games

MDPI

Article

On the Query Complexity of Black-Peg AB-Mastermind

Mourad El Ouali [1], Christian Glazik [2,*], Volkmar Sauerland [2] and Anand Srivastav [2]

[1] Polydisciplinary Faculty Ouarzazate, University Ibn Zohr, Agadir 80000, Morocco; meo@informatik.uni-kiel.de

[2] Department of Computer Science, Kiel University, 24118 Kiel, Germany; vsa@informatik.uni-kiel.de (V.S.); asr@informatik.uni-kiel.de (A.S.)

* Correspondence: cgl@informatik.uni-kiel.de

Received: 8 November 2017; Accepted: 28 December 2017; Published: 2 January 2018

Abstract: Mastermind is a two players zero sum game of imperfect information. Starting with Erdős and Rényi (1963), its combinatorics have been studied to date by several authors, e.g., Knuth (1977), Chvátal (1983), Goodrich (2009). The first player, called "codemaker", chooses a secret code and the second player, called "codebreaker", tries to break the secret code by making as few guesses as possible, exploiting information that is given by the codemaker after each guess. For variants that allow color repetition, Doerr et al. (2016) showed optimal results. In this paper, we consider the so called Black-Peg variant of Mastermind, where the only information concerning a guess is the number of positions in which the guess coincides with the secret code. More precisely, we deal with a special version of the Black-Peg game with n holes and $k \geq n$ colors where no repetition of colors is allowed. We present upper and lower bounds on the number of guesses necessary to break the secret code. For the case $k = n$, the secret code can be algorithmically identified within less than $(n-3)\lceil \log_2 n \rceil + \frac{5}{2}n - 1$ queries. This result improves the result of Ker-I Ko and Shia-Chung Teng (1985) by almost a factor of 2. For the case $k > n$, we prove an upper bound of $(n-2)\lceil \log_2 n \rceil + k + 1$. Furthermore, we prove a new lower bound of n for the case $k = n$, which improves the recent $n - \log\log(n)$ bound of Berger et al. (2016). We then generalize this lower bound to k queries for the case $k \geq n$.

Keywords: Mastermind; combinatorial problem; permutation; algorithm; complexity

MSC: 91A46

1. Introduction

In this paper, we deal with *Mastermind*, which is a popular board game that in the past three decades has become interesting from an algorithmic point of view. Mastermind is a two-player board game invented in 1970 by the postmaster and telecommunication expert Mordecai Meirowitz. The original version of Mastermind consists of a board with twelve (or ten, or eight) rows containing four holes and pegs of six different colors. The idea of the game is that the codemaker chooses a secret color combination of n pegs from k possible colors and the codebreaker has to identify the code by a sequence of queries and corresponding information that is provided by the codemaker. All queries are also color combinations of n pegs. Information is given about the number of correctly positioned colors and further correct colors, respectively. Mathematically, the codemaker selects a vector $y \in [k]^n$ and the codebreaker gives in each iteration a query in form of a vector $x \in [k]^n$. The codemaker replies with a pair of two numbers, called black(x, y) and white(x, y), respectively. The first one is the number of positions in which both vectors x and y coincide and the second one is the number of additional pegs with a right color but a wrong position:

$$\begin{aligned}\text{black}(x,y) &= |\{i \in [n] \mid x(i) = y(i)\}|,\\ \text{white}(x,y) &= \max_{\sigma \in S_n} |\{i \in [n] \mid y(i) = x(\sigma(i))\}|\\ &\quad - \text{black}(x,y).\end{aligned}$$

The Black-Peg game is a special version of Mastermind, where answers are provided by black information only. A further version is the so-called AB game, a.k.a. Bulls and Cows, in which all colors within a code must be distinct. Actually, this version is supposed to be much older than the commercial variant of Mastermind. It is an interesting open question whether both variants are of the same complexity for the codebreaker or if one version is significantly harder because, in the AB game, the space of possible solutions as well as the space of possible queries are both restricted. In this paper, we deal with a special combination of the Black-Peg game and the AB game, where both the secret vector and the guesses must be composed of pairwise distinct colors ($k \geq n$) and the answers are given by the black information only. In a breakthrough paper, Doerr et al. [1] state that $k = n$ is the most popular case in research, where the white information is redundant for the AB game.

Related Works: The study of Mastermind in its different variants has a long lasting history in combinatorial game theory. In 1963, several years before the invention of Mastermind as a commercial board game, Erdős and Rényi [2] analyzed the same problem with two colors. One of the earliest analyses of this game after its commercialization dealing with the case of four pegs and six colors was done by Knuth [3]. He presented a strategy that identifies the secret code in at most five guesses. For the AB game with four pegs, it is known that at least seven guesses are required in the worst case [4]. Ever since the work of Knuth, the general case of arbitrary many pegs and colors has been intensively investigated in combinatorics and computer science literature. In the field of complexity, Stuckman and Zhang [5] showed that it is $\mathcal{NP}$-complete (most likely impossible in polynomial time) to determine if a sequence of queries and answers is satisfiable. Concerning the approximation aspect, there are many works regarding different methods [5–16]. The Black-Peg game was first introduced by Chvátal for the case $k = n$. He gave a deterministic adaptive strategy that uses $2n\lceil \log_2 k \rceil + 4n$ guesses. Later, Goodrich [17] improved the result of Chvátal for arbitrary n and k to $n\lceil \log_2 k \rceil + \lceil (2 - 1/k)n \rceil + k$ guesses. Moreover, he proved in the same paper that this kind of game is $\mathcal{NP}$-complete. A further improvement to $n\lceil \log_2 n \rceil + k - n + 1$ for $k > n$ and $n\lceil \log_2 n \rceil + k$ for $k \leq n$ was done by Jäger and Peczarski [18]. Doerr et al. [1] provided a randomized codebreaker strategy that only needs $\mathcal{O}(n \log \log n)$ queries in expectation. They also showed that this asymptotic order even holds for up to $n^2 \log \log n$ colors, if both black and white information is allowed. For the AB game, Jäger and Peczarski [19] proved exact worst-case numbers of guesses for fixed $n \in \{2, 3, 4\}$ and arbitrary k. Concerning the combination of both variants, Black-Peg game and AB game, for almost three decades, the work due to Ker-I Ko and Shia-Chung Teng [20] was the only contribution that provides an upper bound for the case $k = n$. They presented a strategy that identifies the secret permutation in at most $2n \log_2 n + 7n$ guesses and proved that the corresponding counting problem is $\#\mathcal{P}$-complete.

Our Contribution: In this paper, we consider the Black-Peg game without color repetition. We first present a deterministic polynomial-time algorithm that identifies the secret permutation in less than $(n-3)\lceil \log_2 n \rceil + \frac{5}{2}n - 1$ queries in the case $k = n$ and in less than $(n-2)\lceil \log_2 n \rceil + k + 1$ queries in the case $k > n$. In a conference version (extended abstract) "Improved Approximation Algorithm for the Number of Queries Necessary to Identify a Permutation", upper bounds of this paper have been presented with some sketches of the proofs [21]. Our result for the case $k = n$ improves the result of Ker-I Ko and Shia-Chung Teng [20] by almost a factor of 2. Furthermore, we analyze the worst-case performance of query strategies for both variants of the Game and give a new lower bound of n queries for the case $k = n$, which improves the recently presented lower bound of $n - \log \log(n)$ by Berger et al. [22]. We note, however, that the corresponding asymptotic bound of $O(n)$ is long-established. For $k \geq n$, we generalize this lower bound to k. Both lower bounds even hold if the codebreaker is allowed to use repeated colors in his guesses.

2. Upper Bounds on the Number of Queries

We first consider Black-Peg Mastermind with $k = n$ and the demand for pairwise distinct colors in both the secret code and all queries, i.e., we deal with permutations in S_n.

2.1. Black-Peg AB-Mastermind, Case $k = n$

For convenience, we will use the term permutation for both, a mapping in S_n and its one-line representation as a vector. Our algorithm for finding the secret permutation $y \in S_n$ includes two main phases that are based on two ideas. In the first phase, the codebreaker guesses an initial sequence of n permutations that has a predefined structure. In the second phase, the structure of the initial sequence and the corresponding information by the codemaker enable us to identify correct *components* y_i of the secret code one after another, each by using a binary search. Recall that, for two codes $w = (w_1, \ldots, w_n)$ and $x = (x_1, \ldots, x_n)$, we denote by $\mathrm{black}(w, x)$ the number $|\{i \in [n] \mid w_i = x_i\}|$ of components in which w and x are equal. We denote the mapping x restricted to the set $\{s, \ldots, \ell\}$ with $(x_i)_{i=s}^{\ell}$, $s, \ell \in [n]$.

Phase 1. Consider the n permutations, $\sigma^1, \ldots, \sigma^n$, which are defined as follows: σ^1 corresponds to the identity map and, for $j \in [n-1]$, we obtain σ^{j+1} from σ^j by a circular shift to the right. For example, if $n = 4$, we have $\sigma^1 = (1, 2, 3, 4)$, $\sigma^2 = (4, 1, 2, 3)$, $\sigma^3 = (3, 4, 1, 2)$ and $\sigma^4 = (2, 3, 4, 1)$. Within those n permutations, every color appears exactly once at every position and, thus, we have

$$\sum_{j=1}^{n} \mathrm{black}(\sigma^j, y) = n. \tag{1}$$

The codebreaker guesses $\sigma^1, \ldots, \sigma^{n-1}$ and obtains the additional information $\mathrm{black}(\sigma^n, y)$ from Equation (1).

Phase 2. The strategy of the second phase identifies the values of y one after another. This is done by using two binary search routines, called FINDFIRST and FINDNEXT, respectively. The idea behind both binary search routines is to exploit the information that, for $1 \le i, j \le n-1$, we have $\sigma_i^j = \sigma_{i+1}^{j+1}$, $\sigma_i^n = \sigma_{i+1}^1$, $\sigma_n^j = \sigma_1^{j+1}$ and $\sigma_n^n = \sigma_1^1$, while, except for an infrequent special case, FINDFIRST is used to identify the first correct component of the secret code, and FINDNEXT identifies the remaining components in the main loop of the algorithm. Actually, FINDFIRST would also be able to find the remaining components but requires more guesses than FINDNEXT (twice as many in the worst case). On the other hand, FINDNEXT only works if at least one value of y is already known such that we have to identify the value of one secret code component in advance.

Identifying the First Component: Equation (1) implies that either $\mathrm{black}(\sigma^j, y) = 1$ holds for all $j \in [n]$ or that we can find a $j \in [n]$ with $\mathrm{black}(\sigma^j, y) = 0$.

In the first case, which is infrequent, we can find one correct value of y by guessing at most $\lfloor \frac{n}{2} \rfloor + 1$ modified versions of some initial guess, say σ^1. Namely, if we define a guess σ by swapping a pair of components of σ^1, we will obtain $\mathrm{black}(\sigma, y) = 0$, if and only if one of the swapped components has the correct value in σ^1.

In the frequent second case, we find the first component by FINDFIRST in at most $2\lceil \log_2 n \rceil$ guesses. The routine FINDFIRST is outlined as Algorithm 1 and works as follows. In the given case, we can either find a $j \in [n-1]$ with $\mathrm{black}(\sigma^j, y) > 0$ but $\mathrm{black}(\sigma^{j+1}, y) = 0$ and set $r := j + 1$, or we have $\mathrm{black}(\sigma^n, y) > 0$ but $\mathrm{black}(\sigma^1, y) = 0$ and set $j := n$ and $r := 1$. We call such an index j an *active* index. Now, for every $\ell \in \{2, 3, \ldots, n\}$, we define the code

$$\sigma^{j,\ell} := \left((\sigma_i^j)_{i=1}^{\ell-1}, \sigma_1^r, (\sigma_i^r)_{i=\ell+1}^{n} \right),$$

and call the peg at position ℓ in $\sigma^{j,\ell}$ the pivot peg. Note that notations of the form $(\sigma)_{i=a}^{b}$ substitute $\sigma_a, \sigma_{a+1}, \ldots, \sigma_b$ and vanish in the case $a > b$. From the information $\sigma_i^j = \sigma_{i+1}^r$ for $1 \le i \le n-1$, we conclude that $\sigma^{j,\ell}$ is actually a new permutation as required. The fact that $\mathrm{black}(\sigma^r, y) = 0$

implies that the number of correct pegs up to position $\ell-1$ in σ^j is either black$(\sigma^{j,\ell}, y)$ (if $y_\ell \neq \sigma_1^r$) or black$(\sigma^{j,\ell}, y) - 1$ (if $y_\ell = \sigma_1^r$). For our algorithm, we will only need to know if there exists one correct peg in σ^j up to position $\ell - 1$. The question is cleared up, if black$(\sigma^{j,\ell}, y) \neq 1$. On the other hand, if black$(\sigma^{j,\ell}, y) = 1$, we can define a new guess $\rho^{j,\ell}$ by swapping the pivot peg with a wrong peg in $\sigma^{j,\ell}$. We define

$$\rho^{j,\ell} := \begin{cases} \left((\sigma_i^j)_{i=1}^{\ell}, \sigma_1^r, (\sigma_i^r)_{i=\ell+2}^{n} \right), & \text{if } \ell < n, \\ \left(\sigma_1^r, (\sigma_i^j)_{i=2}^{n-1}, \sigma_1^j \right), & \text{if } \ell = n. \end{cases}$$

Algorithm 1: Routine FINDFIRST

input : Code y and an active index $j \in [n]$
output: Left most correct peg position in σ^j

```
if j = n then r := 1;
else r := j + 1;
a := 1;
b := n;                                          // b is also the position to be found
while b > a do
    ℓ := ⌈(a+b)/2⌉;                                                  // pivot position
    Guess σ^{j,ℓ} := ( (σ_i^j)_{i=1}^{ℓ-1}, σ_1^r, (σ_i^r)_{i=ℓ+1}^{n} );
    s := black(σ^{j,ℓ}, y);
    if s = 1 then
        if ℓ < n then ρ^{j,ℓ} := ( (σ_i^j)_{i=1}^{ℓ}, σ_1^r, (σ_i^r)_{i=ℓ+2}^{n} );
        else ρ^{j,ℓ} := ( σ_1^r, (σ_i^j)_{i=2}^{n-1}, σ_1^j );
        Guess ρ^{j,ℓ};
        s := black(ρ^{j,ℓ}, y);
    if s > 0 then b := ℓ − 1;
    else a := ℓ;
return b;
```

For the case $\ell = n$, we may assume that we applied our query procedure for an $\ell' < \ell$ already, proving that the first $\ell' - 1$ pegs in σ^j are wrong, particularly σ_1^j. Now, we obtain black$(\rho^{j,\ell}, y) > 0$, if and only if the pivot peg had a wrong color in $\sigma^{j,\ell}$ meaning that there is one correct peg in σ^j in the first $\ell - 1$ places. Thus, we can find the position m of the leftmost correct peg in σ^j by a binary search as outlined in Algorithm 1.

Identifying a Further Component: For the implementation of FINDNEXT (Algorithm 2), we deal with a partial solution vector x that satisfies $x_i \in \{0, y_i\}$ for all $i \in [n]$. We call the (indices of the) non-zero components of the partial solution *fixed*. They indicate the components of the secret code that have already been identified. The (indices of the) zero components are called *open*. Whenever FINDNEXT makes a guess σ, it requires knowing the number of open components in which the guess coincides with the secret code, i.e., the number

$$\text{black}(\sigma, y, x) := \text{black}(\sigma, y) - \text{black}(\sigma, x).$$

Note that the term black(σ, x) is known by the codebreaker and not greater than black(σ, y). After the first component of y has been found and fixed in x, there exists a $j \in [n]$ such that black$(\sigma^j, y, x) = 0$. As long as we have open components in x, we can either find a $j \in [n-1]$ with black$(\sigma^j, y, x) > 0$ but black$(\sigma^{j+1}, y, x) = 0$ and set $r := j + 1$, or we have black$(\sigma^n, y, x) > 0$ but black$(\sigma^1, y, x) = 0$ and set $j := n$ and $r := 1$. Again, we call such an index j an *active* index. Let j be an active index and

r its related index. Let c be the color of some component of y that is already identified and fixed in the partial solution x. With ℓ_j and ℓ_r, we denote the position of color c in σ^j and σ^r, respectively. The peg with color c serves as a pivot peg for identifying a correct position m in σ^j that is not fixed yet. There are two possible modes for the binary search that depend on the fact if $m \leq \ell_j$. The mode is indicated by a Boolean variable *leftS* and determined by lines 5 to 10 of FINDNEXT. Clearly, $m \leq \ell_j$ if $\ell_j = n$. Otherwise, the codebreaker guesses

$$\sigma^{j,0} := \left(c, (\sigma_i^j)_{i=1}^{\ell_j-1}, (\sigma_i^j)_{i=\ell_j+1}^{n}\right).$$

By the information $\sigma_i^j = \sigma_{i+1}^r$ we obtain that $(\sigma_i^j)_{i=1}^{\ell_j-1} \equiv (\sigma_i^r)_{i=2}^{\ell_j}$. We further know that every open color has a wrong position in σ^r. For that reason, $\text{black}(\sigma^{j,0}, y, x) = 0$ implies that $m \leq \ell_j$.

Algorithm 2: Routine FINDNEXT

input : Code y, partial solution $x \neq 0$ and an active index $j \in [n]$
output: Position of a correct open component in σ^j

1 **if** $j = n$ **then** $r := 1$;
2 **else** $r := j + 1$;
3 Choose a color c of identified peg (a value c of some non-zero component of x);
4 Let ℓ_j and ℓ_r be the positions with color c in σ^j and σ^r, respectively;
5 **if** $\ell_j = n$ **then** *leftS* := true;
6 **else**
7 Guess $\sigma^{j,0} := \left(c, (\sigma_i^j)_{i=1}^{\ell_j-1}, (\sigma_i^j)_{i=\ell_j+1}^{n}\right)$;
8 $s := \text{black}(\sigma^{j,0}, y, x)$;
9 **if** $s = 0$ **then** *leftS* := true;
10 **else** *leftS* := false;
11 **if** *leftS* **then**
12 $a := 1$;
13 $b := \ell_j$;
14 **else**
15 $a := \ell_r$;
16 $b := n$;
17 **while** $b > a$ **do**
18 $\ell := \lceil \frac{a+b}{2} \rceil$; `// position for peg c`
19 **if** *leftS* **then** $\sigma^{j,\ell} := \left((\sigma_i^j)_{i=1}^{\ell-1}, c, (\sigma_i^r)_{i=\ell+1}^{\ell_j}, (\sigma_i^j)_{i=\ell_j+1}^{n}\right)$;
20 **else** $\sigma^{j,\ell} := \left((\sigma_i^r)_{i=1}^{\ell_r-1}, (\sigma_i^j)_{i=\ell_r}^{\ell-1}, c, (\sigma_i^r)_{i=\ell+1}^{n}\right)$;
21 Guess $\sigma^{j,\ell}$;
22 $s := \text{black}(\sigma^{j,\ell}, y, x)$;
23 **if** $s > 0$ **then** $b := \ell - 1$;
24 **else** $a := \ell$;
25 **return** b;

The binary search for the exact value of m is done in the interval $[a, b]$, where m is initialized as n and $[a, b]$ as

$$[a, b] := \begin{cases} [1, \ell_j], & \text{if } \textit{leftS}, \\ [\ell_r, n], & \text{else} \end{cases}$$

(lines 11 to 16 of FINDNEXT). In order to determine if there is an open correct component on the left side of the current center ℓ of $[a,b]$ in σ^j, we can define a case dependent permutation:

$$\sigma^{j,\ell} := \begin{cases} \left((\sigma_i^j)_{i=1}^{\ell-1}, c, (\sigma_i^r)_{i=\ell+1}^{\ell_j}, (\sigma_i^j)_{i=\ell_j+1}^{n} \right), & \text{if } \textit{leftS}, \\ \left((\sigma_i^r)_{i=1}^{\ell_r-1}, (\sigma_i^j)_{i=\ell_r}^{\ell-1}, c, (\sigma_i^r)_{i=\ell+1}^{n} \right), & \text{else.} \end{cases}$$

In the first case, the first $\ell-1$ components of $\sigma^{j,\ell}$ coincide with those of σ^j. The remaining components of $\sigma^{j,\ell}$ cannot coincide with the corresponding components of the secret code if they have not been fixed yet. This is because the ℓ-th component of $\sigma^{j,\ell}$ has the already fixed value c, components $\ell+1$ to ℓ_j coincide with the corresponding components of σ^r, which satisfies $\text{black}(\sigma^r, y, x) = 0$, and the remaining components have been checked to be wrong in this case (cf. former definition of *leftS* in line 5 and line 9, respectively). Thus, there is a correct open component on the left side of ℓ in σ^j, if and only if $\text{black}(\sigma^{j,\ell}, y, x) \neq 0$. In the second case, the same holds for similar arguments. Now, if there is a correct open component to the left of ℓ, we update the binary search interval $[a,b]$ by $[a,\ell-1]$. Otherwise, we update $[a,b]$ by $[\ell,b]$.

The Main Algorithm. The main algorithm is outlined as Algorithm 3. It starts with an empty partial solution and finds the components of the secret code y one-by-one. Herein, the vector v does keep records about the number of open components in which the permutations $\sigma^1,\ldots,\sigma^n$ equal y and is, thus, initialized by $v_i := \text{black}(\sigma^i, y)$, $i \in [n-1]$ and $v_n := n - \sum_{i=1}^{n-1} v_i$. As mentioned above, the main loop always requires an active index. For that reason, if $v = \mathbb{1}_n$ in the beginning, we fix one solution peg in σ^1 and update x and v, correspondingly. Every call of FINDNEXT in the main loop augments x by a correct solution value. Since one call of FINDNEXT requires at most $1 + \lceil \log_2 n \rceil$ guesses, Algorithm 3 does not need more than $(n-3)\lceil \log_2 n \rceil + \frac{5}{2}n - 1$ queries for $n \geq 10$ (inclusive $n-1$ initial guesses, $\lfloor \frac{n}{2} \rfloor + 1$ guesses to find the first correct peg, $n-3$ calls of FINDNEXT and 2 final queries) to break the secret code.

Algorithm 3: Algorithm for Permutations

1 Let y be the secret code and set $x := (0,0,\ldots,0)$;
2 Guess the permutations σ^i, $i \in [n-1]$;
3 Initialize $v \in \{0,1,\ldots,n\}^n$ by $v_i := \text{black}(\sigma^i, y)$, $i \in [n-1]$, $v_n := n - \sum_{i=1}^{n-1} v_i$;
4 **if** $v = \mathbb{1}_n$ **then**
5 $j := 1$;
6 Find the position m of the correct peg in σ^1 by at most $\lfloor \frac{n}{2} \rfloor + 1$ further guesses;
7 **else**
8 Call $m :=$ FINDFIRST(y, j) for an active index $j \in [n]$ to find the position m of the correct peg in σ^j by at most $2\lceil \log_2 n \rceil$ further guesses;
9 $x_m := \sigma_m^j$;
10 $v_j := v_j - 1$;
11 **while** $|\{i \in [n] \mid x_i = 0\}| > 2$ **do**
12 Use v to choose an active index $j \in [n]$; // $(v_j > 0,\ v_{j+1} = 0)$
13 $m :=$ FINDNEXT(y, x, j);
14 $x_m := \sigma_m^j$;
15 $v_j := v_j - 1$;
16 Make at most two more guesses to find the remaining two unidentified colors;

Example 1. *We consider the case $n = k = 8$ and suppose that the secret code y is*

$$7 \quad 1 \quad 4 \quad 3 \quad 2 \quad 8 \quad 5 \quad 6.$$

Figure 1 (upper panel) shows n possible initial queries. We illustrate the procedure FINDNEXT and further suppose that we have already identified the positions of three colors indicated in the partial solution x:

$$\bullet \quad \bullet \quad \bullet \quad \bullet \quad 2 \quad \bullet \quad 5 \quad 6.$$

From the n_3 values in Figure 1, we see that $\text{black}(\sigma_3, y, x) = 1$ and $\text{black}(\sigma_4, y, x) = 0$, so we choose 3 as our active index applying FINDNEXT with the highlighted initial queries, σ^3 and σ^4. Choosing the already identified color 2 as a pivot color, FINDNEXT does its binary search to identify the next correct peg as demonstrated in the lower panel of Figure 1. Since the information n_3 for query σ^a is 0 (cf. lines 7–9 of Algorithm 2), all correctly placed but unidentified pegs in σ^3 are in the first four places. Thus, we can apply a binary search for the leftmost correct peg in the first four places of query σ^3 using the pivot peg. Here, the binary search is done by queries σ^b and σ^c and identifies the peg with color 7 (in general, the peg that is left to the leftmost pivot position for which n_3 is non-zero). If the response to σ^a would have been greater than 0, we would have found analogously a new correct peg among the last four places of σ^3.

	queries								n_1	n_2	n_3
σ^1	1	2	3	4	5	6	7	8	0	0	0
σ^2	8	1	2	3	4	5	6	7	2	0	2
σ^3	7	8	1	2	3	4	5	6	3	2	1
σ^4	6	7	8	1	2	3	4	5	1	1	0
σ^5	5	6	7	8	1	2	3	4	0	0	0
σ^6	4	5	6	7	8	1	2	3	0	0	0
σ^7	3	4	5	6	7	8	1	2	1	0	1
σ^8	2	3	4	5	6	7	8	1	1	0	1

	queries								n_1	n_2	n_3
σ^a	2	7	8	1	3	4	5	6	2	2	0
σ^b	7	8	2	1	3	4	5	6	3	2	1
σ^c	7	2	8	1	3	4	5	6	3	2	1

Figure 1. Upper panel: initial queries σ^j with associated responses $n_1 = \text{black}(\sigma^j, y)$, coincidences with a partial solution $n_2 = \text{black}(\sigma^j, x)$, and the difference of both n_3. Lower panel: binary search queries to extend the partial solution. The highlighted subsequences correspond to the subsequences of the selected initial queries.

2.2. More Colors Than Positions

Now, we consider the variant of Black-Peg Mastermind where $k > n$ and color repetition is forbidden. Let $y = (y_1, \ldots, y_n)$ be the code that must be found. We use the same notations as above.

Phase 1. Consider the k permutations $\overline{\sigma}^1, \ldots, \overline{\sigma}^k$, where $\overline{\sigma}^1$ corresponds to the identity map on $[k]$ and for $j \in [k-1]$, we obtain $\overline{\sigma}^{j+1}$ from $\overline{\sigma}^j$ by a circular shift to the right. We define k codes $\sigma^1, \ldots, \sigma^k$ by $\sigma^j = (\overline{\sigma}^j_i)_{i=1}^n$, $j \in [k]$. For example, if $k = 5$ and $n = 3$, we have $\sigma^1 = (1,2,3)$, $\sigma^2 = (5,1,2)$, $\sigma^3 = (4,5,1)$, $\sigma^4 = (3,4,5)$ and $\sigma^5 = (2,3,4)$. Within those k codes, every color appears exactly once at every position, and, thus, we have

$$\sum_{j=1}^{k} \text{black}(\sigma^j, y) = n,$$

similar to Equation (1). Since $k > n$, this implies

Lemma 1. *There is a* $j \in [k]$ *with* $\text{black}(\sigma^j, y) = 0$.

Phase 2. Having more colors than holes, we can perform our binary search for the next correct position without using a pivot peg. The corresponding simplified version of FINDNEXT is outlined as Algorithm 4. Using that version of FINDNEXT also allows to simplify our main algorithm (Algorithm 3) by adapting lines 2 and 3, and, due to Lemma 1, skipping lines 4–10, as FINDNEXT can be already applied to find the first correct peg. Thus, for the required number of queries to break the secret code, we have: the initial $k-1$ guesses, a call of the modified FINDNEXT for all but the last two positions (at most $\lceil \log_2 n \rceil$ guesses per position) and one or two final guesses. This yields the modified Mastermind Algorithm breaking the secret code in at most $(n-2)\lceil \log_2 n \rceil + k + 1$ queries.

Algorithm 4: Routine FINDNEXT for $k > n$

input : Code y, partial solution $x \neq 0$ and an active index $j \in [k]$
output: Position m of a correct open component in σ^j

1 **if** $j = n$ **then** $r := 1$;
2 **else** $r := j + 1$;
3 $a := 1, b := n$;
4 **while** $b > a$ **do**
5 $\quad \ell := \lceil \frac{a+b}{2} \rceil$; `// mid position of current interval`
6 $\quad$ Guess $\sigma := \left((\sigma_i^r)_{i=1}^{\ell-1}, (\sigma_i^j)_{i=\ell}^{n} \right)$;
7 $\quad s := \text{black}(\sigma, y, x)$;
8 $\quad$ **if** $s > 0$ **then** $a := \ell$;
9 $\quad$ **else** $b := \ell - 1$;
10 **return** a;

3. Lower Bounds on the Number of Queries

In the following, we consider the case that the secret code has no repetition but arbitrary questions are allowed. Note that the lower bounds for that case especially hold true for Black-Peg AB-Mastermind, since the codebreaker will not be able to detect a secret code with less attempts, if the set of allowed queries is restricted to the corresponding subset. Similar to the upper bounds, we prove the respective lower bounds on the necessary number of queries by construction.

3.1. Black-Peg AB-Mastermind, Case $k = n$

In each iteration, the worst case for the code breaker is simulated by allowing the codemaker to replace his secret code with another permutation from the remaining feasible search space. For $m \in \mathbb{N}$, we denote the m-th query of the code breaker with x^m and the m-th secret code adaption of the codemaker with y^m. The remaining feasible search space R_m consists of all permutations that agree with the first m pairs of queries and answers:

$$R_0 := S_n,$$
$$R_m := \{\sigma \in S_n \mid \text{black}(y^j, x^j) = \text{black}(\sigma, x^j) \text{ for all } j \in [m]\}, \text{ for } m > 0.$$

Now, a simple strategy of the codemaker is to reply every query x^m, $m \in \mathbb{N}$, with the smallest possible number

$$b_m := \min_{\sigma \in R_{m-1}} \text{black}(\sigma, x^m),$$

choosing his new secret code $y^m \in R_{m-1}$ such that $\text{black}(y^m, x^m) = b_m$. We obtain our lower bound on the necessary number of queries by proving the following Lemma.

Lemma 2. *It holds that* $b_m \leq m$ *for all* $m < n$.

In particular, none of the first $n-1$ queries will be answered with n. Thus, the secret code cannot be identified with less than n queries.

Proof. Assuming that our claim is wrong, we fix the smallest number $m \in [n-1]$ with $b_m > m$. Let

$$D := \{c \in [n] \mid (x^m)^{-1}(c) = (y^m)^{-1}(c)\}$$

be the set of colors that are correctly placed in the current query with respect to the current secret code. For every $i \in [n]$, let $C_i \subseteq [n]$ be the set of all colors that do not occur at position i in any of the former $m-1$ queries nor in the current secret code, i.e.,

$$C_i := \{c \in [n] \mid c \neq x^\ell(i) \text{ for all } \ell \in [m]\}.$$

The intersections $C_i \cap D$, $i \in [n]$ are not empty since $|D| = b_m \geq m+1$ but at most m of the n colors are missing in C_i. This fact will enable us to determine a new feasible secret code $z \in R_{m-1}$ such that $\text{black}(z, x^j) = b_j$ for all $j \in [m-1]$ but $\text{black}(z, x^m) < b_m$, a contradiction to the minimality of b_m. The new secret code z is constructed from y^m by changing the colors of some components that coincide with x^m, choosing the new color at a given position i from $C_i \cap D$. The precise procedure is outlined as Algorithm 5. Starting with any position i_1 where y^m and x^m have the same color, we choose another color $c_1 \in C_{i_1} \cap D$. Since $c_1 \in D$, there must be another position i_2 such that $y^m(i_2) = c_1 = x^m(i_2)$. Thus, for $s > 1$, we can iteratively determine positions i_s where y^m and x^m have the same color, c_{s-1}, and choose a new color $c_s \in C_{i_s} \cap D$ (while loop, lines 5–9). The iteration stops, if the chosen color c_s corresponds with a color that appears in y^m at some position i_t, $t < s$, that has been considered before (indicated by the set A). Note that the iteration must terminate with $2 \leq s \leq m+1$, since A is empty in the beginning, and $|D| = m+1$. The set of chosen colors $\{c_\ell \mid t \leq \ell \leq s\}$ is equal to the set of colors $\{y^m(i_\ell) \mid t \leq \ell \leq s\}$ at the corresponding positions in y^m. Hence, the new secret code z (defined in lines 10–11) is again a permutation. Now, let $j \in [m-1]$ be the number of some former query. Since $z \in R_{m-1} \subseteq R_{j-1}$, the definition of b_j implies $\text{black}(z, x^j) \geq b_j$. However, $\text{black}(z, x^j) \leq b_j$ also holds since $\text{black}(y^m, x^j) = b_j$ ($y^m \in R_m$), and, for each position i with $z(i) \neq y^m(i)$, we have $z(i) \neq x^j(i)$ ($z(i) \in C_i$). Furthermore, the construction of z immediately yields $\text{black}(z, x^m) < \text{black}(y^m, x^m) = b_m$, since z is obtained from y^m by changing some pegs that coincided in y^m and x^m. Thus, z is indeed a secret permutation in R_{m-1} that contradicts the minimality of b_m. □

Algorithm 5: Secret code adaption, $k = n$

```
1  s := 1;
2  A := ∅;
3  Choose position i_1 ∈ [n] with y^m(i_1) = x^m(i_1);
4  Choose color c_1 ∈ C_{i_1} ∩ D;
5  while c_s ∉ A do
6  |  A := A ∪ {y^m(i_s)};
7  |  s := s + 1;
8  |  i_s := (y^m)^{-1}(c_{s-1});
9  |  Choose color c_s ∈ C_{i_s} ∩ D;
10 Find the unique t < s with y^m(i_t) = c_s;
11 z := y^m;
12 for ℓ := t to s do z(i_ℓ) := c_ℓ;
```

3.2. More Colors Than Positions

Considering the case $k > n$, we adapt the codemaker strategy from the former subsection, i.e., in each turn m, the codemaker chooses the new secret code y^m such that the answer is the smallest possible answer b_m. We easily obtain a lower bound of k queries by the following:

Lemma 3. *It holds that $b_m < n$ for all $m < k$.*

Proof. Assume for a moment that there exists an $m < k$ with $b_m = n$. Like before, let

$$C_i := \{c \in [n] \mid c \neq x^\ell(i) \text{ for all } \ell \in [m]\}.$$

Similar to Algorithm 5, we now replace certain entries of y^m by elements of the corresponding C_i. The detailed procedure is described in Algorithm 6.

Algorithm 6: Secret code adaption, $k > n$

```
s := 1;
i_1 := 1;
A := ∅;
B := {c ∈ [k] | For all i ∈ [n] : y^k(i) ≠ c};
Choose color c_1 ∈ C_1;
while c_s ∉ A ∪ B do
    A := A ∪ {y^m(i_s)};
    s := s + 1;
    i_s := (y^m)^{-1}(c_{s-1});
    Choose color c_s ∈ C_{i_s};
if c_s ∈ A then
    Find the unique t < s with y^m(i_t) = c_s;
else t := 1;
z := y^m;
for ℓ := t to s do z(i_ℓ) := c_ℓ;
```

We start with position one and choose a color $c_1 \in C_{i_1}$. As soon as we have $c_s \in B$, we construct z by starting with y^k and then replacing the color $y^m(i_\ell)$ by the color c_{i_ℓ} for any $\ell \leq s$. The set of chosen colors $\{c_\ell \mid \ell \leq s\}$ is equal to the set of colors $\{y^m(i_\ell) \mid \ell \leq s\}$ except for c_s, which only appears in the first set and $y^m(i_\ell)$, which only appears in the second. Since $c_s \in B$, we know that z has no color occurring twice.

If the iteration stops because of $c_s \in A$, the procedure is identical to the one in Algorithm 5. Thus, in both cases, we find that $\text{black}(z, x^m) < b_m$ and $\text{black}(z, x^\ell) = b_\ell$ for any $\ell \in [m-1]$, in contradiction to the minimality of b_m. □

4. Discussion

We present deterministic algorithms for the identification of a secret code in "Black-Peg AB-Mastermind" as well as a "cheating algorithm" for the codemaker. Our constructive algorithms yield new upper and lower bounds on the necessary number of queries. A challenge of the considered Mastermind variant is that no color repetition is allowed for a query while most strategies for other Mastermind variants exploit the property of color repetition. We improve the recent lower bound of Berger et al. [22] and show that the worst case number of queries for Black-Peg AB-Mastermind with $k = n$ is at least n, another matter than the asymptotic bound of $O(n)$, which is long-established. Ko and Teng [20] conjecture that this number is actually $\Omega(n \log n)$, a proof of which would close the gap to the upper bound, answering the question of whether the AB game is harder than the general game. The lower bound proof of Berger et al. is derived by solely considering the search space partition with respect to the number of coincidences with the very first query. On the other hand, our algorithmic proof does not exploit any structure property of the remaining search space. For both reasons, we expect at least some room for improvements of the lower bound. Our corresponding general lower bound for Black-Peg AB-Mastermind (case $k \geq n$) is k. In the future, we will keep both bounds in focus, but the real challenge is to prove or disprove the conjecture of Ko and Teng. It would also be

interesting to examine the impact of further restrictions concerning the answers by the codemaker. We conjecture that our binary search approach will also work if the codemaker answers a query by only indicating if there is at least one black peg.

Acknowledgments: Christian Glazik and Volkmar Sauerland contributed to this work while supported by DFG Cluster of Excellence 80. We also would like to acknowledge financial support by Land Schleswig-Holstein within the funding programme Open Access Publikationsfonds.

Author Contributions: Mourad El Ouali invented the codebreaker strategies that yield the upper bounds. Christian Glazik invented the codemaker strategies and proved the corresponding lower bounds. Volkmar Sauerland implemented and tested the codebreaker strategies. All authors contributed to the manuscript and approved the version submitted to *games*.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Doerr, B.; Doerr, C.; Spöhel, R.; Thomas, H. Playing Mastermind with Many Colors. *J. ACM* **2016**, *63*, 1–23.
2. Erdős, P.; Rényi, C. On Two Problems in Information Theory. *Publ. Math. Inst. Hung. Acad. Sci.* **1963**, *8*, 229–242.
3. Knuth, D.E. The computer as a master mind. *J. Recreat. Math.* **1977**, *9*, 1–5.
4. Francis, J. Strategies for playing MOO, or "Bulls and Cows". 2010. Available online: https://pdfs.semanticscholar.org/d839/f794cccd174790b0cde695d3626f00caf7e1.pdf (accessed on 21 December 2017).
5. Stuckman, J.; Zhang, G. Mastermind is NP-Complete. *INFOCOMP J. Comput. Sci.* **2006**, *5*, 25–28.
6. Bergmann, L.; Goossens, D.; Leus, R. Efficient solutions for Mastermind using genetic algorithms. *Comput. Op. Res.* **2009**, *36*, 1880–1885.
7. Chen, Z.; Cunha, C.; Homer, S. Finding a Hidden Code by Asking Questions. In Proceedings of the 2nd Conference on Computing and Combinatorics (COCOON 1996), Hong Kong, China, 17–19 June 1996; pp. 50–56.
8. Chvátal, V. Mastermind. *Combinatorica* **1983**, *3*, 325–329.
9. Doerr, B.; Winzen, C. Playing Mastermind with Constant-Size Memory. In Proceedings of the 29th Symposium on Theoretical Aspects of Computer Science (STACS 2012), Paris, France, 29 February–3 March 2012; pp. 441–452.
10. Focardi, R.; Luccio, F.L. Cracking Bank PINs by Playing Mastermind. In Proceedings of the 5th International Conference on Fun with Algorithms (FUN 2010), Ischia, Italy, 2–4 June 2010; pp. 202–213.
11. Guervós, J.J.M.; Cotta, C.; Gacia, A.M. Improving and Scaling Evolutionary Approaches to the Mastermind Problem. In Proceedings of Applications of Evolutionary Computation (EvoApplications 2011), Torino, Italy, 27–29 April 2011; pp. 103–112.
12. Guervós, J.J.M.; Mora, A.M.; Cotta, C. Optimizing worst-case scenario in evolutionary solutions to the Mastermind puzzle. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2011), New Orleans, LA, USA, 5–8 June 2011; pp. 2669–2676.
13. Goodrich, M.T. The Mastermind Attack on Genomic Data. In Proceedings of the 30th IEEE Symposium on Security and Privacy (SP 2009), Oakland, CA, USA, 17–20 May 2009; pp. 204–218.
14. Jäger, G.; Peczarski, M. The number of pessimistic guesses in Generalized Mastermind. *Inf. Process. Lett.* **2009**, *109*, 635–641.
15. Kalisker, T.; Camens, D. Solving Mastermind Using Genetic Algorithms. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003), Chicago, IL, USA, 12–16 July 2003; pp. 1590–1591.
16. Koyama, K.; Lai, T.W. An optimal Mastermind strategy. *J. Recreat. Math.* **1993**, *25*, 251–256.
17. Goodrich, M.T. On the algorithmic complexity of the Mastermind game with black-peg results. *Inf. Process. Lett.* **2009**, *109*, 675–678.
18. Jäger, G.; Peczarski, M. The number of pessimistic guesses in Generalized Black-peg Mastermind. *Inf. Process. Lett.* **2011**, *111*, 933–940.
19. Jäger, G.; Peczarski, M. The worst case number of questions in Generalized AB game with and without white-peg answers. *Discret. Appl. Math.* **2015**, *184*, 20–31.
20. Ko, K.; Teng, S. On the Number of Queries Necessary to Identify a Permutation. *J. Algorithms* **1986**, *7*, 449–462.

21. El Ouali, M.; Sauerland, V. Improved Approximation Algorithm for the Number of Queries Necessary to Identify a Permutation. In Proceedings of the 24th International Workshop on Combinatorial Algorithms (IWOCA 2013), Rouen, France, 10–12 July 2013; pp. 443–447.
22. Berger, A.; Chute, C.; Stone, M. Query Complexity of Mastermind Variants. *arXiv* **2016**, arXiv:1607.04597.