Hardness of Mastermind

Giovanni Viglietta

University of Pisa, Italy,
viglietta@gmail.com

Abstract. Mastermind is a popular board game released in 1971, where a codemaker chooses a secret pattern of colored pegs, and a codebreaker has to guess it in several trials. After each attempt, the codebreaker gets a response from the codemaker containing some information on the number of correctly guessed pegs. The search space is thus reduced at each turn, and the game continues until the codebreaker is able to find the correct code, or runs out of trials.

In this paper we study several variations of #MSP, the problem of computing the size of the search space resulting from a given (possibly fictitious) sequence of guesses and responses. Our main contribution is a proof of the **#P**-completeness of #MSP under parsimonious reductions, which settles an open problem posed by Stuckman and Zhang in 2005, concerning the complexity of deciding if the secret code is uniquely determined by the previous guesses and responses. Similarly, #MSP stays **#P**-complete under Turing reductions even with the promise that the search space has at least k elements, for any constant k. (In a regular game of Mastermind, $k = 1$.)

All our hardness results hold even in the most restrictive setting, in which there are only two available peg colors, and also if the codemaker's responses contain less information, for instance like in the so-called single-count (black peg) Mastermind variation.

Keywords: Mastermind; code-breaking; game; counting; search space.

1 Introduction

Mastermind at a glance. *Mastermind* is a code-breaking board game released in 1971, which sold over 50 million sets in 80 countries. The Israeli postmaster and telecommunication expert Mordecai Meirowitz is usually credited for inventing it in 1970, although an almost identical paper-and-pencil game called *bulls and cows* predated Mastermind, perhaps by more than a century [1].

The classic variation of the game is played between a *codemaker*, who chooses a secret sequence of four colored pegs, and a *codebreaker*, who tries to guess it in several attempts. There are six available colors, and

Fig. 1. A Mastermind box published by Pressman Toy Corporation in 1981, foreshadowing the game's computational hardness.

the secret code may contain repeated colors. After each attempt, the codebreaker gets a *rating* from the codemaker, consisting in the number of correctly placed pegs in the last guess, and the number of pegs that have the correct color but are misplaced. The rating does not tell which pegs are correct, but only their amount. These two numbers are communicated by the codemaker as a sequence of smaller black pegs and white pegs, respectively (see Figure 1, where the secret code is concealed behind a shield, and each guess is paired with its rating). If the codebreaker's last guess was wrong, he guesses again, and the game repeats until the secret code is found, or the codebreaker reaches his limit of ten trials. Ideally, the codebreaker plans his new guesses according to the information he collected from the previous guesses. Table 1 depicts a complete game of Mastermind, where colors are encoded as numbers between zero and five, and the codebreaker finally guesses the code at his sixth attempt.

Table 1. A typical game of Mastermind.

Secret code: ⓪ ① ② ③	
Guess	Rating
④ ④ ① ①	○
③ ② ② ④	● ○
⓪ ③ ⓪ ④	● ○
⑤ ⑤ ③ ④	○
① ② ⓪ ③	● ○ ○ ○
⓪ ① ② ③	● ● ● ●

Previous work. Recently, Focardi and Luccio pointed out the unexpected relevance of Mastermind in real-life security issues, by showing how certain API-level bank frauds, aimed at disclosing user PINs, can be interpreted as an extended Mastermind game played between an insider and the bank's computers [5]. On the other hand, Goodrich suggested some applications to genetics of the Mastermind variation in which scores consist of black pegs only, called *single-count (black peg) Mastermind* [6].

As a further generalization of the original game, we may consider (n, c)-*Mastermind*, where the secret sequence consists of n pegs, and there are c available colors. Chvátal proved that the codebreaker can always determine the secret code in (n, c)-Mastermind after at most $2n \log_2 c + 4n + \lceil \frac{c}{n} \rceil$ guesses, each computable in polynomial time, via a simple divide-and-conquer strategy [3]. This upper bound was later lowered by a constant factor in [2], while Goodrich also claimed to be able to lower it for single-count (black peg) Mastermind, hence using even less information [6]. Unfortunately, after a careful inspection, Goodrich's method turns out to outperform Chvátal's several techniques given in [3] asymptotically (as n grows, and c is a function of n) only if $n^{1-\varepsilon} < c < (3 + \varepsilon) n \log_2 n$, for every $\varepsilon > 0$.

However, despite being able to guess any secret code with an efficient strategy, the codebreaker may insist on really minimizing the number of trials, either in the worst case or on average. Knuth proposed a heuristic that exhaustively searches through all possible guesses and ratings, and greedily picks a guess that will minimize the number of eligible solutions, in the worst case [8]. This is practical and worst-case optimal for standard $(4, 6)$-Mastermind, but infeasible and suboptimal for even slightly bigger instances. The size of the solution space is employed as an ideal quality indicator also in other heuristics, most notably those based on genetic algorithms [7].

In order to approach the emerging complexity theoretic issues, Stuckman and Zhang introduced the Mastermind Satisfiability Problem (MSP) for (n, c)-Mastermind, namely the problem of deciding if a given sequence of guesses and ratings has indeed a solution, and proved its **NP**-completeness [10]. Similarly, Goodrich showed that also the analogous satisfiability problem for single-count (black peg) Mastermind is **NP**-complete [6].

Interestingly, Stuckman and Zhang observed that the problem of detecting MSP instances with a unique solution is Turing-reducible to the problem of producing an eligible solution. However, the determination of the exact complexity of the first problem is left open [10].

Our contribution. In this paper we study #MSP, the *counting problem* associated with MSP, i.e., the problem of computing the number of solutions that are compatible with a given set of guesses and ratings. We do this for standard (n, c)-Mastermind, as well as its single-count variation with only black peg ratings, and the analogous single-count variation with only white peg ratings, both in general and restricted to instances with a fixed number of colors c.

Our main theorem states that, in all the aforementioned variations of Mastermind, #MSP is either trivially polynomial or **#P**-complete under *parsimonious reductions*. Capturing the true complexity of #MSP is an improvement on previous results (refer to [6,10]) because:

- Evaluating the size of the search space is a natural and recurring subproblem in several heuristics, whereas merely deciding if a set of guesses has a solution seems a more fictitious problem, especially because in a real game of Mastermind we already know that our previous guesses and ratings *do* have a solution.
- The reductions we give are parsimonious, hence they yield stronger versions of all the previously known **NP**-completeness proofs for MSP and its variations. Moreover, we obtain the same hardness results even for $(n, 2)$-Mastermind, whereas all the previous reductions used unboundedly many colors (see Corollary 1).
- Our main theorem enables simple proofs of a wealth of complexity-related corollaries, including the hardness of detecting unique solutions, which was left open in [10] (see Corollary 3).

Paper structure. In Section 2 we define #MSP and its variations. Section 3 contains a statement and proof of our main result, Theorem 1, and an example of reduction. In Section 4 we apply Theorem 1 to several promise problems with different assumptions on the search space, and finally in Section 5 we suggest some directions for further research.

2 Definitions

Codes and ratings. For (n, c)-Mastermind, let the set $\mathbb{Z}_c^n$ be the *code space*, whose elements are *codes* of n numbers ranging from 0 to $c - 1$. Following Chvátal, we define two *metrics* on the code space [3]. If $x = (x_1, \cdots, x_n)$ and $y = (y_1, \cdots, y_n)$ are two codes, let $\alpha(x, y)$ be the number of subscripts i with $x_i = y_i$, and let $\beta(x, y)$ be the largest $\alpha(x, \tilde{y})$, with $\tilde{y}$ running through all the permutations of y. As observed in [10], $n-\alpha(x, y)$ and $n - \beta(x, y)$ are indeed distance functions, respectively on $\mathbb{Z}_c^n$ and

$\mathbb{Z}_c^n/S_n$ (i.e., the code space where codes are equivalent up to reordering of their elements).

Given a secret code $s \in \mathbb{Z}_c^n$ chosen by the codemaker, we define the *rating* of a guess $g \in \mathbb{Z}_c^n$, for all the three variants of Mastermind we want to model.

- For standard Mastermind, let $\rho(s,g) = (\alpha(s,g), \beta(s,g) - \alpha(s,g))$.
- For single-count black peg Mastermind, let $\rho_b(s,g) = \alpha(s,g)$.
- For single-count white peg Mastermind, let $\rho_w(s,g) = \beta(s,g)$.

A guess is considered correct in single-count white peg (n,c)-Mastermind whenever its rating is n, therefore the secret code has to be guessed only up to reordering of the numbers. As a consequence, the codebreaker can always guess the code after $c-1$ attempts: He can determine the number of pegs of each color via monochromatic guesses, although this is not an optimal strategy when c outgrows n. On the other hand, order does matter in both other variants of Mastermind, where the guess has to coincide with the secret code for the codebreaker to win.

Satisfiability problems. Next we define the Mastermind Satisfiability Problem for all three variants of Mastermind.

Problem 1. MSP (respectively, MSP-Black, MSP-White).
Input: (n, c, Q), where Q is a finite set of queries of the form (g, r), where $g \in \mathbb{Z}_c^n$ and r is a rating.
Output: Yes if there exists a code $x \in \mathbb{Z}_c^n$ such that $r = \rho(x, g)$ (respectively, $r = \rho_b(x, g)$, $r = \rho_w(x, g)$) for all $(g, r) \in Q$. No otherwise.

MSP and MSP-Black are known to be **NP**-complete problems [6,10]. We shall see in Corollary 1 how MSP-White is **NP**-complete, as well.

Further, we may want to restrict our attention to instances of Mastermind with a fixed number of colors. Thus, for every constant c, let (c)-MSP be the restriction of MSP to problem instances with exactly c colors (i.e., whose input is of the form $(\cdot, c, \cdot)$). Similarly, we define (c)-MSP-Black and (c)-MSP-White.

Counting problems. All the above problems are clearly in **NP**, thus it makes sense to consider their *counting versions*, namely #MSP, #MSP-Black, $\#(c)$-MSP, and so on, which are all **#P** problems [11]. Basically, these problems ask for the size of the solution space after a number of guesses and ratings, i.e., the number of codes that are coherent with all the guesses and ratings given as input.

Recall that reductions among **#P** problems that are based on oracles are called *Turing reductions* and are denoted with $\leqslant_{\mathrm{T}}$, while the more specific reductions that map problem instances preserving the number of solutions are called *parsimonious reductions*, and are denoted with $\leqslant_{\mathrm{pars}}$. Each type of reduction naturally leads to a different notion of **#P**-completeness: For instance, #2-SAT is **#P**-complete under Turing reductions, while #3-SAT is **#P**-complete under parsimonious reductions [9]. Problems that are **#P**-complete under parsimonious reductions are *a fortiori* **NP**-complete, while it is unknown whether all **NP**-complete problems are **#P**-complete, even under Turing reductions [4].

3 Counting Mastermind solutions

Next we give a complete classification of the complexities of all the counting problems introduced in Section 2.

Theorem 1.

a) #MSP, #MSP-Black *and* #MSP-White *are* **#P***-complete under parsimonious reductions.*

b) #(c)-MSP *and* #(c)-MSP-Black *are* **#P***-complete under parsimonious reductions for every $c \geqslant 2$.*

c) #(c)-MSP-White *is solvable in deterministic polynomial time for every $c \geqslant 1$.*

(Notice that #(1)-MSP and #(1)-MSP-Black are trivially solvable in deterministic linear time.)

Lemma 1. *For every $c \geqslant 1$,* #(c)-MSP-White *is solvable in deterministic polynomial time.*

Proof. In $\mathbb{Z}_c^n / S_n$ there are only $\binom{n+c-1}{c-1} = \Theta(n^{c-1})$ possible codes to check against all the given queries, hence the whole process can be carried out in polynomial time, for any constant c. □

Lemma 2. *For every $c \geqslant 1$,*

$$\#(c)\text{-MSP} \leqslant_{\mathrm{pars}} \#(c+1)\text{-MSP},$$
$$\#(c)\text{-MSP-Black} \leqslant_{\mathrm{pars}} \#(c+1)\text{-MSP-Black}.$$

Proof. Given the instance (n, c, Q) of #(c)-MSP (respectively, #(c)-MSP-Black), we convert it into $(n, c+1, Q \cup \{(g, r)\})$, where g is a sequence of n consecutive c's, and $r = (0, 0)$ (respectively, $r = 0$). The new query (g, r) implies that the new color c does not occur in the secret code, hence the number of solutions is preserved and the reduction is indeed parsimonious. □

Lemma 3. #3-SAT $\leqslant_{\text{pars}}$ #MSP-White.

Proof. Given a 3-CNF Boolean formula φ with v variables and m clauses, we map it into an instance of MSP-White (n, c, Q). For each clause C_i of φ, we add three fresh *auxiliary variables* a_i, b_i, c_i. For each variable x (including auxiliary variables), we define two colors x and $\bar{x}$, representing the two possible truth assignments for x. We further add the *mask color* $*$, thus getting $c = 2v + 6m + 1$ colors in total. We let $n = v + 3m$ (we may safely assume that $n \geqslant 5$), and we construct Q as follows.

1) Add the query $\Big((*, *, *, \cdots, *), 0\Big)$.
2) For each variable x, add the query $\Big((x, x, \bar{x}, \bar{x}, *, *, *, \cdots, *), 1\Big)$.
3) For each clause $C_i = \{\ell_1, \ell_2, \ell_3\}$ (where each literal may be positive or negative), add the query $\Big((\ell_1, \ell_2, \ell_3, a_i, b_i, *, *, *, \cdots, *), 3\Big)$.
4) For each clause C_i, further add the query $\Big((\overline{a_i}, b_i, c_i, *, *, *, \cdots, *), 2\Big)$.

By (1), the mask color does not occur in the secret code; by (2), each variable occurs in the secret code exactly once, either as a positive or a negative literal. Moreover, by (3), at least one literal from each clause must appear in the secret code. Depending on the exact number of literals from C_i that appear in the code (either one, two or three), the queries in (3) and (4) always force the values of the auxiliary variables a_i, b_i and c_i. (Notice that, without (4), there would be two choices for a_i and b_i, in case exactly two literals of C_i appeared in the code.) As a consequence, the reduction is indeed parsimonious. □

Lemma 4. #3-SAT $\leqslant_{\text{pars}}$ #(2)-MSP-Black.

Proof. We proceed along the lines of the proof of Lemma 3, with similar notation. We add the same auxiliary variables a_i, b_i, c_i for each clause C_i, and we construct the instance of (2)-MSP-Black $(2n, 2, Q)$, where $n = v + 3m$. This time we encode literals as *positions* in the code: For each variable x, we allocate two specific positions x and $\bar{x}$, so that $g_x = 1$ (respectively, $g_{\bar{x}} = 1$) in code $g = (g_1, \cdots, g_{2n})$ if and only if variable x is assigned the value true (respectively, false). Notice that, in contrast with Lemma 3, we are not using a mask color here. Q is constructed as follows.

1) Add the query $\Big((0, 0, 0, \cdots, 0), n\Big)$.
2) For each variable x, add the query (g, n), where $g_j = 1$ if and only if $j \in \{x, \bar{x}\}$.

3) For each clause $C_i = \{\ell_1, \ell_2, \ell_3\}$, add the query $(g, n+1)$, where $g_j = 1$ if and only if $j \in \{\ell_1, \ell_2, \ell_3, a_i, b_i\}$. (Without loss of generality, we may assume that ℓ_1, ℓ_2 and ℓ_3 are occurrences of three mutually distinct variables [14].)
4) For each clause C_i, further add the query $(g, n+1)$, where $g_j = 1$ if and only if $j \in \{\overline{a_i}, b_i, c_i\}$.

By (1), every solution must contain n times 0 and n times 1, in some order. The semantics of (2), (3) and (4) is the same as that of the corresponding steps in Lemma 3, hence our construction yields the desired parsimonious reduction. Indeed, observe that, if altering k bits of a binary code increases its rating by r, then exactly $\frac{k+r}{2}$ of those k bits are set to the right value. In (2), altering $k = 2$ bits of the code in (1) increases its rating by $r = 0$, hence exactly one of those bits has the right value, which means that $s_x \neq s_{\bar{x}}$ in any solution s. Similarly, in (3) (respectively, (4)), $k = 5$ (respectively, $k = 3$) and $r = 1$, hence exactly three (respectively, two) of the bits set to 1 are correct (cf. the ratings in Lemma 3). □

Lemma 5. #3-SAT $\leqslant_{\text{pars}}$ #(2)-MSP.

Proof. We replicate the construction given in the proof of Lemma 4, but we use the proper ratings: Recall that the ratings of MSP are pairs of scores (black pegs and white pegs). The first score (black pegs) has the same meaning as in MSP-Black, and we maintain these scores unchanged from the previous construction. By doing so, we already get the desired set of solutions, hence we merely have to show how to fill out the remaining scores (white pegs) without losing solutions.

Referring to the proof of Lemma 4, we change the rating in (1) from n to $(n, 0)$, because every 0 in the guess is either correct at the correct place, or redundant.

The rating in (2) is changed from n to $(n, 2)$. Indeed, let y be any other variable (distinct from x), so that $g_y = g_{\bar{y}} = 0$. Then, exactly one between g_y and $g_{\bar{y}}$ is a misplaced 0, which can be switched with the misplaced 1 from either g_x or $g_{\bar{x}}$. All the other 0's in g are either correct at the correct place, or redundant.

Similarly, the rating in (3) (respectively, (4)) changes from $n+1$ to $(n+1, 4)$ (respectively, $(n+1, 2)$). Indeed, exactly two (respectively, one) 1's are in a wrong position in g. If either $g_x = 1$ or $g_{\bar{x}} = 1$ is wrong, then both g_x and $g_{\bar{x}}$ are wrong and of opposite colors, hence they can be switched. Once again, all the other 0's in g are either correct at the correct place, or redundant. □

Proof (of Theorem 1). All the claims easily follow from Lemma 1, Lemma 2, Lemma 3, Lemma 4, Lemma 5, and the **#P**-completeness of #3-SAT under parsimonious reductions [9]. □

Example. As an illustration of Lemma 5, we show how the Boolean formula $(x \vee \neg y \vee z) \wedge (\neg x \vee y \vee w) \wedge (y \vee \neg z \vee \neg w)$ is translated into a set of queries for (2)-MSP. For visual convenience, 0's and 1's are represented as white and black circles, respectively.

x	$\overline{x}$	y	$\overline{y}$	z	$\overline{z}$	w	$\overline{w}$	a_1	$\overline{a}_1$	b_1	$\overline{b}_1$	c_1	$\overline{c}_1$	a_2	$\overline{a}_2$	b_2	$\overline{b}_2$	c_2	$\overline{c}_2$	a_3	$\overline{a}_3$	b_3	$\overline{b}_3$	c_3	$\overline{c}_3$	Rating
○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	(13, 0)
●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	(13, 2)
○	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	(13, 2)
○	○	○	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	(13, 2)
○	○	○	○	○	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	(13, 2)
○	○	○	○	○	○	○	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	(13, 2)
○	○	○	○	○	○	○	○	○	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	(13, 2)
○	○	○	○	○	○	○	○	○	○	○	○	●	●	○	○	○	○	○	○	○	○	○	○	○	○	(13, 2)
○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	●	○	○	○	○	○	○	○	○	○	○	(13, 2)
○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	●	○	○	○	○	○	○	○	○	(13, 2)
○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	●	○	○	○	○	○	○	(13, 2)
○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	●	○	○	○	○	(13, 2)
○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	●	○	○	(13, 2)
○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	●	(13, 2)
●	○	○	●	●	○	○	○	●	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	(14, 4)
○	●	●	○	○	○	●	○	○	○	○	○	○	○	●	○	●	○	○	○	○	○	○	○	○	○	(14, 4)
○	○	●	○	○	●	○	●	○	○	○	○	○	○	○	○	○	○	○	○	●	○	●	○	○	○	(14, 4)
○	○	○	○	○	○	○	○	○	●	●	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	(14, 2)
○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	●	○	●	○	○	○	○	○	○	○	(14, 2)
○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	●	○	●	○	(14, 2)
x	$\overline{x}$	y	$\overline{y}$	z	$\overline{z}$	w	$\overline{w}$	a_1	$\overline{a}_1$	b_1	$\overline{b}_1$	c_1	$\overline{c}_1$	a_2	$\overline{a}_2$	b_2	$\overline{b}_2$	c_2	$\overline{c}_2$	a_3	$\overline{a}_3$	b_3	$\overline{b}_3$	c_3	$\overline{c}_3$	Rating

The solutions to both problems are exactly ten, and are listed below.

x	y	z	w
T	T	T	T
T	T	T	F
T	T	F	T
T	T	F	F
T	F	F	T
F	T	T	T
F	T	T	F
F	F	T	F
F	F	F	T
F	F	F	F

x	$\overline{x}$	y	$\overline{y}$	z	$\overline{z}$	w	$\overline{w}$	a_1	$\overline{a}_1$	b_1	$\overline{b}_1$	c_1	$\overline{c}_1$	a_2	$\overline{a}_2$	b_2	$\overline{b}_2$	c_2	$\overline{c}_2$	a_3	$\overline{a}_3$	b_3	$\overline{b}_3$	c_3	$\overline{c}_3$
●	○	●	○	●	○	●	○	○	●	●	○	○	●	○	●	●	○	○	●	●	○	●	○	●	○
●	○	●	○	●	○	○	●	○	●	●	○	○	●	●	○	●	○	●	○	○	●	●	○	○	●
●	○	●	○	○	●	●	○	●	○	●	○	●	○	○	●	●	○	○	●	○	●	●	○	○	●
●	○	●	○	○	●	○	●	●	○	●	○	●	○	●	○	●	○	●	○	○	●	○	●	●	○
●	○	○	●	○	●	●	○	○	●	●	○	○	●	●	○	●	○	●	○	●	○	●	○	●	○
○	●	●	○	●	○	●	○	●	○	●	○	●	○	○	●	○	●	●	○	●	○	●	○	●	○
○	●	●	○	●	○	○	●	●	○	●	○	●	○	○	●	●	○	○	●	○	●	●	○	○	●
○	●	○	●	●	○	○	●	○	●	●	○	○	●	●	○	●	○	●	○	●	○	●	○	●	○
○	●	○	●	○	●	●	○	●	○	●	○	●	○	○	●	●	○	○	●	●	○	●	○	●	○
○	●	○	●	○	●	○	●	●	○	●	○	●	○	○	●	●	○	○	●	○	●	●	○	○	●

We remark that, in order to determine the values of the auxiliary variables a_i, b_i and c_i when a solution to the Boolean satisfiability problem is given, it is sufficient to check how many literals of C_i are satisfied. a_i is true if and only if exactly one literal is satisfied, b_i is false if and only if all three literals are satisfied, and c_i is true if and only if $a_i = b_i$.

4 Related results

We describe some applications of Theorem 1 to several complexity problems.

Corollary 1. (2)-MSP, (2)-MSP-Black *and* MSP-White *are* **NP**-*complete.*

Proof. Parsimonious reductions among **#P** problems are *a fortiori* Karp reductions among the corresponding **NP** problems. □

So far, we made no assumptions on the queries in our problem instances, which leads to a more general but somewhat fictitious theory. Since in a real game of Mastermind the codebreaker's queries are guaranteed to have at least a solution (i.e., the secret code chosen by the codemaker), more often than not the codebreaker is in a position to exploit this information to his advantage. However, we show that such information does not make counting problems substantially easier.

Corollary 2. #(2)-MSP, #(2)-MSP-Black *and* #MSP-White, *with the promise that the number of solutions is at least* k*, are all* **#P**-*complete problems under Turing reductions, for every* $k \geqslant 1$.

Proof. Let #Match be the problem of counting the matchings of any size in a given graph, which is known to be **#P**-complete under Turing reductions [12]. Let Π_k be the problem #(2)-MSP (respectively, #(2)-MSP-Black, #MSP-White) restricted to instances with at least k solutions, and let us show that #Match $\leqslant_{\mathrm{T}}$ Π_k. Given a graph G, if it has fewer than k edges, we can count all the matchings in linear time. Otherwise, there must be at least k matchings (each edge e yields at least the matching $\{e\}$), so we parsimoniously map G into an instance of Π_k via Theorem 1, we call an oracle for Π_k, and output its answer. □

The following result, for $k = 1$, settles an issue concerning the determination of MSP instances with unique solution, which was left unsolved in [10]. We actually prove more: Even if a solution is given as input, it is hard to determine if it is unique. Therefore, not only solving Mastermind puzzles is hard, but *designing* puzzles around a solution is also hard.

Corollary 3. *For every $k \geqslant 1$, the problem of deciding if an instance of* (2)-MSP*,* (2)-MSP-Black *or* MSP-White *has strictly more than k solutions is* **NP***-complete, even if k solutions are explicitly given as input.*

Proof. Not only do the parsimonious reductions given in Theorem 1 preserve the number of solutions, but they actually yield an explicit polynomial-time computable transformation of solutions (cf. the remark at the end of Section 3). Hence, the involved **#P**-complete problems are also **ASP**-complete as function problems, and their decision k-**ASP** counterparts are accordingly **NP**-complete [14]. □

Remarkably, even if the codebreaker somehow knows that his previous queries are sufficient to uniquely determine the solution, he still has a hard time finding it.

Corollary 4. *The promise problem of finding the solution to an instance of* (2)-MSP*,* (2)-MSP-Black *or* MSP-White*, when the solution itself is known to be unique, is* **NP***-hard under randomized Turing reductions.*

Proof. It is known that SAT $\leqslant_{\mathbf{RP}}$ USAT, where USAT is the promise version of SAT whose input formulas are known to have either zero or one satisfying assignments [13]. Let f be the composition of this reduction with the parsimonious one from Boolean formulas to instances of (2)-MSP (respectively, (2)-MSP-Black, MSP-White) given by Theorem 1. Our Turing reduction proceeds as follows: Given a Boolean formula φ, compute $f(\varphi)$ and submit it to an oracle that finds a correct solution s of (2)-MSP (respectively, (2)-MSP-Black, MSP-White) when it is unique. Then output Yes if and only if s is indeed a solution of $f(\varphi)$, which can be checked in polynomial time. □

5 Further research

In Lemma 1 we showed that #(c)-MSP-White is solvable in polynomial time when c is a constant, while in Lemma 3 we proved that it becomes **#P**-complete when $c = 2n + 1$. By making the code polynomially longer and filling the extra space with a fresh color, we can easily prove that also #($\Theta(\sqrt[k]{n})$)-MSP-White is **#P**-complete, for every constant k. An obvious question arises: What is the lowest order of growth of $c(n)$ such that #($c(n)$)-MSP-White is **#P**-complete?

We observed that #MSP is a subproblem of several heuristics aimed at optimally guessing the secret code, but is #MSP really inherent in the game? Perhaps the hardness of Mastermind is not captured by #MSP or even MSP, and there are cleverer, yet unknown, ways to play.

Problem 2. MASTERMIND.
Input: (n, c, Q, k), where (n, c, Q) is an instance of MSP, and $k \geqslant 0$.
Output: YES if the codebreaker has a strategy to guess the secret code in at most k attempts, using information from Q. NO otherwise.

Notice that MASTERMIND belongs to **PSPACE**, due to the polynomial upper bound on the length of the optimal strategy given by Chvátal [3]. Our question is whether MASTERMIND is **PSPACE**-complete.

To make the game more fun to play for the codemaker, whose role is otherwise too passive, we could let him change the secret code at every turn, coherently with the ratings of the previous guesses of the codebreaker. As a result, nothing changes for the codebreaker, except that he may perceive to be quite unlucky with his guesses, but the codemaker's game becomes rather interesting: By Corollary 3, even deciding if he has a non-trivial move is **NP**-complete, but he can potentially force the codebreaker to always work in the worst-case scenario, and make him pay for his mistakes. We call this variation *adaptive Mastermind*.

References

1. `http://en.wikipedia.org/wiki/Mastermind_(board_game)`.
2. Z. Chen, C. Cunha, and S. Homer. Finding a hidden code by asking questions. In *Proceedings of COCOON'96*, 50–55, 1996.
3. V. Chvátal. Mastermind. *Combinatorica*, 3:325–329, 1983.
4. M. Dyer, L. A. Goldberg, C. Greenhill, and M. Jerrum. On the relative complexity of approximate counting problems. In *Proceedings of APPROX'00*, 108–119, 2000.
5. R. Focardi and F. L. Luccio. Cracking bank PINs by playing Mastermind. In *Proceedings of FUN'10*, 202–213, 2010.
6. M. T. Goodrich. On the algorithmic complexity of the Mastermind game with black-peg results. *Information Processing Letters*, 109:675–678, 2009.
7. T. Kalisker and D. Camens. Solving Mastermind using genetic algorithms. In *Proceedings of GECCO'03*, 1590–1591, 2003.
8. D. E. Knuth. The computer as Master Mind. *Journal of Recreational Mathematics*, 9:1–6, 1976–77.
9. C. H. Papadimitriou. *Computational complexity*. Addison-Wesley Publishing Company, Inc., 1994.
10. J. Stuckman and G.-Q. Zhang. Mastermind is NP-complete. *INFOCOMP Journal of Computer Science*, 5:25–28, 2006.
11. L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.
12. L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8:410–421, 1979.
13. L. G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.
14. T. Yato. Complexity and completeness of finding another solution and its application to puzzles. Master's thesis, University of Tokyo, 2003.