Hardness of Mastermind

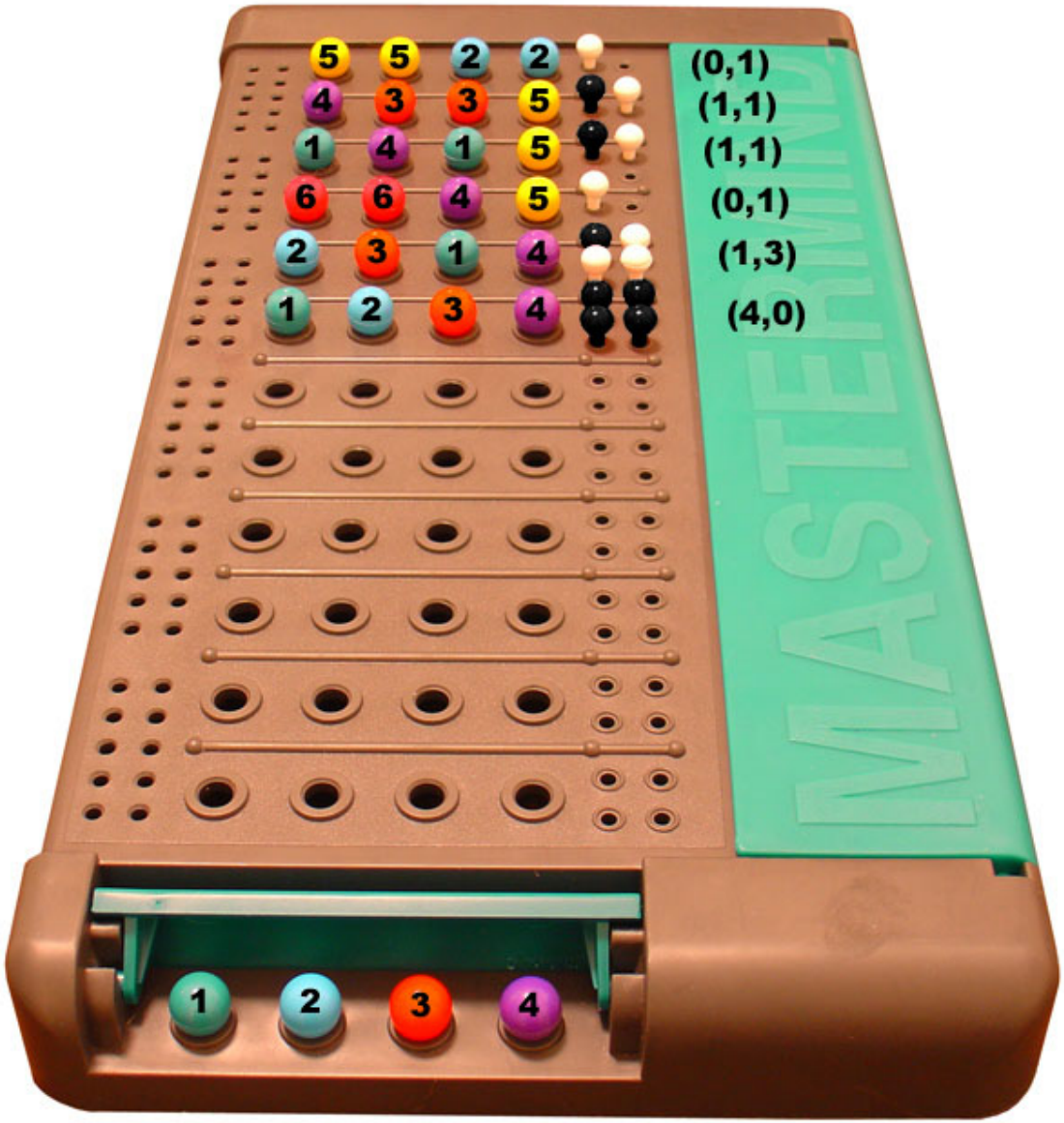
Giovanni Viglietta

Department of Computer Science, University of Pisa, Italy

Pisa - January 19$^{\text{th}}$, 2011

"Easy to learn. Easy to play. But not so easy to win."

Mastermind commercial, 1981



Mastermind is played on a board with colored pegs. A *codemaker* chooses a secret sequence of colors, and a *codebreaker* has to guess it in several attempts.

Mastermind at a glance

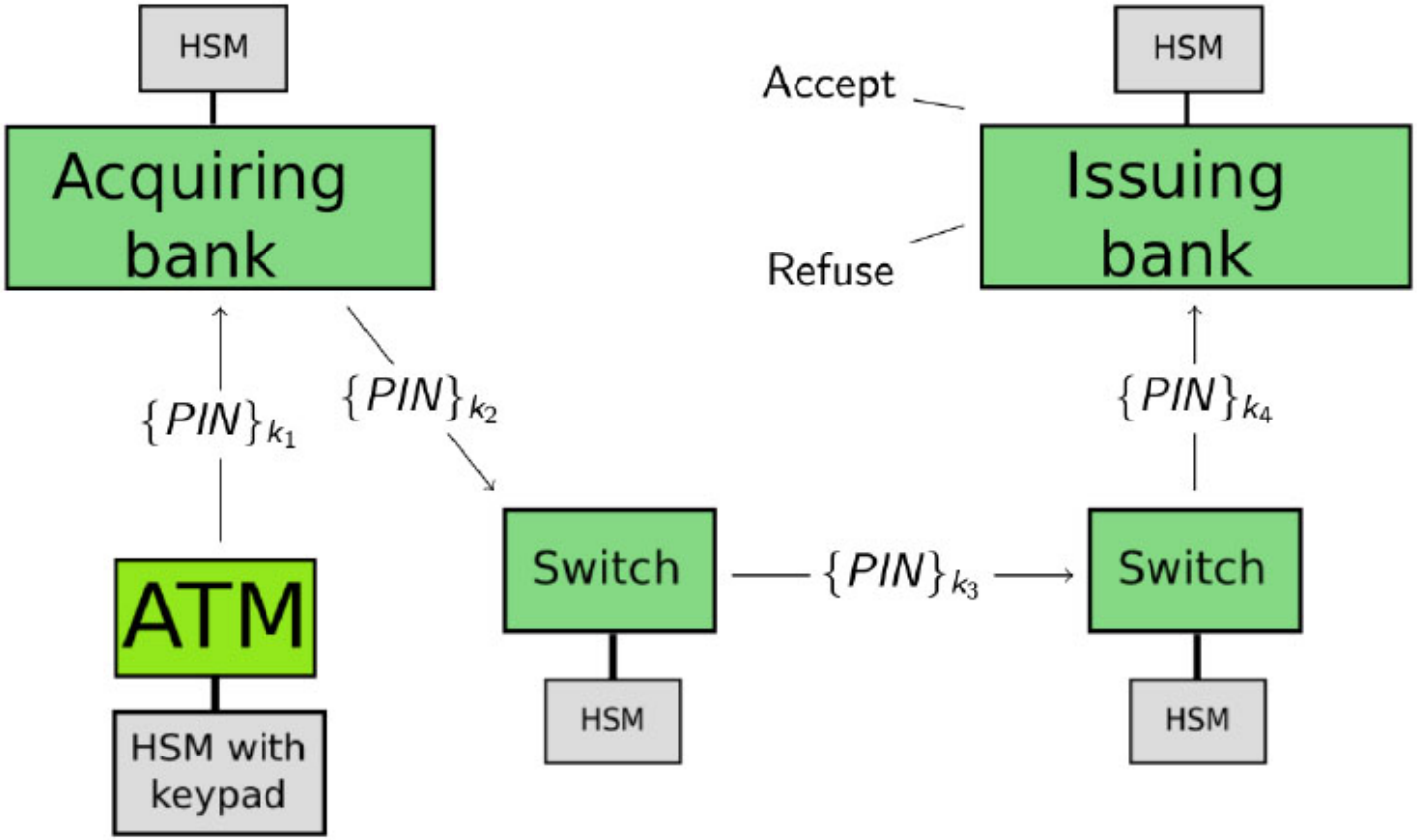
- After each guess, the codemaker responds with some black and white pegs.
  - Black pegs represent correct pegs in the codebreaker's guess, which are also well-placed.
  - White pegs represent pegs in the codebreaker's guess which are correct but misplaced.
  - Black and white pegs don't mark the positions of the correct pegs in the codebreaker's guess, but only their amount.

Mastermind at a glance

- After each guess, the codemaker responds with some black and white pegs.
  - Black pegs represent correct pegs in the codebreaker's guess, which are also well-placed.
  - White pegs represent pegs in the codebreaker's guess which are correct but misplaced.
  - Black and white pegs don't mark the positions of the correct pegs in the codebreaker's guess, but only their amount.
- In the classic game from 1970 the code's length is 4 and there are 6 available colors, but complexity theorists deal with the generalized (n, c)-Mastermind with n positions and c colors.
- The codemaker's response is then encoded as a pair (b, w) with $0 \leqslant b + w \leqslant n$.
  - $n - b$ is the Hamming distance between guess and secret code.
  - $n - b - w$ is also a distance on the space of unordered n-tuples.
- As a result, the solution space after a sequence of attempts can be regarded as an intersection of spheres in some metric space over the strings.

Mastermind in bank frauds

- The relevance of Mastermind in real-life security issues was pointed out in 2010 by *Focardi* and *Luccio*.
- User PINs travelling in a bank network are decrypted and re-encrypted at every switch.



Mastermind in bank frauds

- An insider of the bank who gains access to some switch is able to issue several PIN verification API calls.
- By subtly manipulating certain offsets and public parameters, and checking for acceptance or refusal, he can eventually deduce the user PIN, digit by digit.
- This kind of attack is performed exactly as an extended Mastermind game played between the insider and the bank's computers.

A feasible heuristic

- A systematic study of Mastermind was carried out by *Chvátal*, in a 1983 paper dedicated to Erdős on his 70^{th} birthday.

- Chvátal suggested a simple divide-and-conquer strategy for the codebreaker to guess the code in $2n\lceil \log c \rceil + 4n + \lceil \frac{c}{n} \rceil$ attempts. Each guess can be computed in polynomial time.

- This bound was subsequently lowered by a constant factor, with an improvement on the same basic idea.

A feasible heuristic

- A systematic study of Mastermind was carried out by *Chvátal*, in a 1983 paper dedicated to Erdős on his 70^{th} birthday.

- Chvátal suggested a simple divide-and-conquer strategy for the codebreaker to guess the code in $2n\lceil \log c \rceil + 4n + \lceil \frac{c}{n} \rceil$ attempts. Each guess can be computed in polynomial time.

- This bound was subsequently lowered by a constant factor, with an improvement on the same basic idea.

- But playing perfectly is still hard: the classic $(4, 6)$-Mastermind is solvable within 5 guesses, while Chvátal's algorithm guesses 18 times.

Exhaustive searches

- Another thread of heuristics was started in 1976 by *Knuth*, who devised a worst-case optimal (w.r.t. the number of guesses) greedy strategy to beat $(4, 6)$-Mastermind.

- Every step of the strategy is a brute-force search among all possible guesses and all possible responses of the codemaker.
- The heuristic is based on choosing the guess that will most reduce the size of the solution space, in the worst case.
- This is practical and optimal for $(4, 6)$-Mastermind, but still infeasible and suboptimal in general.
- Several other approaches were adopted, most notably genetic algorithms, achieving different performance tradeoffs.

Satisfiability

The following is a sub-problem of most Knuth-like strategies:

Mastermind Satisfiability Problem (MSP)

Input: (n, c, Q), *where* Q *is any sequence of guesses and responses in* (n, c)*-Mastermind.*
Output: YES *if there actually exists a code which is compatible with all the queries in* Q, NO *otherwise.*

Satisfiability

The following is a sub-problem of most Knuth-like strategies:

Mastermind Satisfiability Problem (MSP)

Input: (n, c, Q), *where* Q *is any sequence of guesses and responses in* (n, c)*-Mastermind.*
Output: YES *if there actually exists a code which is compatible with all the queries in* Q, NO *otherwise.*

- In 2005 *Stuckman* and *Zhang* proved that MSP is NP-complete.
- In 2009 *Goodrich* proved the same result for a variant of Mastermind where the codemaker only responds with black pegs.

Uniqueness of solution

The following observation could be derived from general theorems, but Stuckman and Zhang gave a simpler proof that inherently relies on the rules of Mastermind:

Observation (Stuckman-Zhang, 2005)

MSP instances with a unique solution can be determined in polynomial time using an oracle which finds a satisfying solution for general MSP instances.

Uniqueness of solution

The following observation could be derived from general theorems, but Stuckman and Zhang gave a simpler proof that inherently relies on the rules of Mastermind:

Observation (Stuckman-Zhang, 2005)

MSP instances with a unique solution can be determined in polynomial time using an oracle which finds a satisfying solution for general MSP instances.

- This motivates the question: how hard is to determine if a sequence of queries has a unique solution, and how hard is to find it?
- Such questions naturally arise in most cryptography-related studies.
- In order to tackle the problem, we must turn to the *counting version* of MSP.

Counting problems

- #P is the set of counting problems associated to decision problems in NP.
  - #SAT: how many variable assignments satisfy a given Boolean formula?
  - #HAM: how many Hamiltonian cycles are there in a given graph?

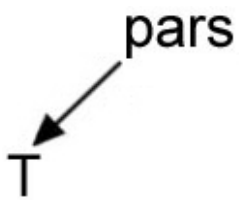
Counting problems

- #P is the set of counting problems associated to decision problems in NP.
  - #SAT: how many variable assignments satisfy a given Boolean formula?
  - #HAM: how many Hamiltonian cycles are there in a given graph?
- Several polynomial-time reductions can be defined between problems A and B in #P:
- **Parsimonious:** maps instances of A into instances of B with the same number of witnesses.

pars

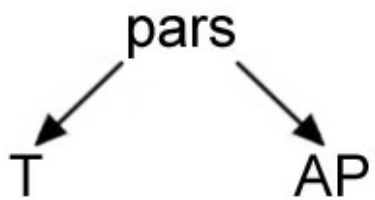
Counting problems

- #P is the set of counting problems associated to decision problems in NP.
  - #SAT: how many variable assignments satisfy a given Boolean formula?
  - #HAM: how many Hamiltonian cycles are there in a given graph?
- Several polynomial-time reductions can be defined between problems A and B in #P:
- **Parsimonious:** maps instances of A into instances of B with the same number of witnesses.
- **Turing:** uses an oracle for B to solve A.



Counting problems

- #P is the set of counting problems associated to decision problems in NP.
  - #SAT: how many variable assignments satisfy a given Boolean formula?
  - #HAM: how many Hamiltonian cycles are there in a given graph?
- Several polynomial-time reductions can be defined between problems A and B in #P:
- **Parsimonious:** maps instances of A into instances of B with the same number of witnesses.
- **Turing:** uses an oracle for B to solve A.
- **Approximation Preserving:** transforms a randomized approximation scheme for B into one for A.



#P-completeness

- #P-complete problems can be defined with respect to every type of reduction.
- Cook's proof of the NP-hardness of SAT actually constructs parsimonious reductions:

Corollary

#SAT is #P-complete under parsimonious reductions.

#P-completeness

- #P-complete problems can be defined with respect to every type of reduction.
- Cook's proof of the NP-hardness of SAT actually constructs parsimonious reductions:

Corollary

#SAT is #P-complete under parsimonious reductions.

- #SAT $\leqslant_{pars}$ #3-SAT,
  where the clauses of 3-SAT have at most 3 literals.

#P-completeness

- #P-complete problems can be defined with respect to every type of reduction.
- Cook's proof of the NP-hardness of SAT actually constructs parsimonious reductions:

Corollary

#SAT is #P-complete under parsimonious reductions.

- #SAT $\leqslant_{pars}$ #3-SAT,
  where the clauses of 3-SAT have at most 3 literals.

Theorem

If A is NP-complete, #A is #P-complete under AP-reductions.

- It's unknown whether the same holds for Turing reductions.

Detecting unique solutions

Unique Satisfiability (USAT)

Input: *a CNF formula* φ*.*

Output: YES *if* φ *is satisfied by a unique assignment,* NO *if* φ *is not satisfiable. Otherwise any answer is fine.*

Detecting unique solutions

Unique Satisfiability (USAT)

Input: *a CNF formula* φ.
Output: YES *if* φ *is satisfied by a unique assignment,* NO *if* φ *is not satisfiable. Otherwise any answer is fine.*

Theorem (Valiant-Vazirani, 1986)

SAT $\leqslant_{RP}$ *USAT. Thus USAT*$\notin$*RP unless RP=NP.*

Corollary

Given a Boolean formula with a unique solution, computing its solution is NP-hard under randomized reductions.

Detecting unique solutions

Unique Satisfiability (USAT)

Input: *a CNF formula* φ.
Output: YES *if* φ *is satisfied by a unique assignment,* NO *if* φ *is not satisfiable. Otherwise any answer is fine.*

Theorem (Valiant-Vazirani, 1986)

SAT $\leqslant_{RP}$ *USAT. Thus USAT*$\notin$*RP unless RP=NP.*

Corollary

Given a Boolean formula with a unique solution, computing its solution is NP-hard under randomized reductions.

Although we know there exists an obscure AP-reduction from #SAT to #MSP, in order to apply the above theorems we need a *parsimonious* one.

Restricted Mastermind

- Inspired by Goodrich, we define two variants of MSP.
  - In *MSP-BLACK* the codemaker responds only with black pegs.
  - In *MSP-WHITE* the codemaker responds with $(b + w)$ white pegs whenever in MSP he would have responded (b, w). The codebreaker has to guess the code up to reordering of the pegs.

Restricted Mastermind

- Inspired by Goodrich, we define two variants of MSP.
  - In *MSP-BLACK* the codemaker responds only with black pegs.
  - In *MSP-WHITE* the codemaker responds with $(b + w)$ white pegs whenever in MSP he would have responded (b, w). The codebreaker has to guess the code up to reordering of the pegs.
- *c-MSP* is always played with c colors, while n is still a variable.
  - Similarly for *c-MSP-BLACK* and *c-MSP-WHITE*.

Preliminary results

Observation

In (n, c)-Mastermind restricted to white pegs, the codebreaker guesses the code after $c - 1$ attempts.

- He tries all possible colors... Although this is suboptimal.

Observation

$\#c$-MSP-WHITE $\in$ FP.

- There are only $\binom{n+c-1}{c-1} = \Theta(n^{c-1})$ possible codes to check.

Observation

$\#(c-1)$-MSP $\leqslant_{pars}$ $\#c$-MSP.

- Add the guess $ccc\cdots$ with response $(0, 0)$.
- This holds also for $\#c$-MSP-BLACK and $\#c$-MSP-WHITE.

Hardness of Mastermind

- The following statement is the strongest possible with respect to color numbers:

Theorem

$$\#3\text{-}SAT \leqslant_{pars} \begin{cases} \#MSP\text{-}WHITE \\ \#2\text{-}MSP\text{-}BLACK \\ \#2\text{-}MSP. \end{cases}$$

- Stuckman's, Zhang's and Goodrich's reductions for MSP and MSP-BLACK don't help, since they're not parsimonious.

$\#3\text{-SAT} \leqslant_{pars} \#\text{MSP-WHITE}$

- Let $\varphi \in$ 3-CNF be given.
- Add color $*$ and add the query
  - $***\cdots$ wth score (0).
  - ($*$ is used as a *mask color* which never occurs.)

$\#3\text{-SAT} \leqslant_{pars} \#\text{MSP-WHITE}$

- Let $\varphi \in$ 3-CNF be given.
- Add color $*$ and add the query
  - $****\cdots$ wth score (0).
  - ($*$ is used as a *mask color* which never occurs.)
- For every variable x, add colors x and $\bar{x}$, and add the query
  - $xx\bar{x}\bar{x}****\cdots$ with score (1).
  - (Only one between x and $\bar{x}$ will be in the correct code, representing the truth value of variable x.)

#3-SAT $\leqslant_{pars}$ #MSP-WHITE

- Let $\varphi \in$ 3-CNF be given.
- Add color $*$ and add the query
  - $***\cdots$ wth score (0).
  - ($*$ is used as a *mask color* which never occurs.)
- For every variable x, add colors x and $\bar{x}$, and add the query
  - $xx\bar{x}\bar{x}***\cdots$ with score (1).
  - (Only one between x and $\bar{x}$ will be in the correct code, representing the truth value of variable x.)
- For each clause (x) in φ, add the query
  - $x***\cdots$ with score (1).

#3-SAT $\leqslant_{pars}$ #MSP-WHITE

- Let $\varphi \in$ 3-CNF be given.
- Add color $*$ and add the query
  - $***\cdots$ wth score (0).
  - ($*$ is used as a *mask color* which never occurs.)
- For every variable x, add colors x and $\bar{x}$, and add the query
  - $xx\bar{x}\bar{x}***\cdots$ with score (1).
  - (Only one between x and $\bar{x}$ will be in the correct code, representing the truth value of variable x.)
- For each clause (x) in φ, add the query
  - $x***\cdots$ with score (1).
- For each clause $C_i = (x \vee y)$ in φ, add the fresh colors c_i and $\bar{c}_i$, and the queries
  - $c_i c_i \bar{c}_i \bar{c}_i ***\cdots$ with score (1),
  - $xyc_i ***\cdots$ with score (2).
  - (At least one between x and y has to be true.)
  - Of course, if $\neg x$ or $\neg y$ occur in the clause, we use colors $\bar{x}$ and $\bar{y}$ in the query.

#3-SAT $\leqslant_{pars}$ #MSP-WHITE

- For each clause $C_i = (x \vee y \vee z)$ in φ, add fresh colors $c_i, c_i', \bar{c}_i, \bar{c}_i'$, and the queries
  - $c_i c_i \bar{c}_i \bar{c}_i * * * \cdots$ with score (1),
  - $c_i' c_i' \bar{c}_i' \bar{c}_i' * * * \cdots$ with score (1),
  - $xyz c_i c_i' * * * \cdots$ with score (3).

#3-SAT $\leqslant_{pars}$ #MSP-WHITE

- For each clause $C_i = (x \vee y \vee z)$ in φ, add fresh colors $c_i, c_i', \bar{c}_i, \bar{c}_i'$, and the queries
  - $c_i c_i \bar{c}_i \bar{c}_i * * * \cdots$ with score (1),
  - $c_i' c_i' \bar{c}_i' \bar{c}_i' * * * \cdots$ with score (1),
  - $xyz c_i c_i' * * * \cdots$ with score (3).
  - This is still not parsimonious, because the values of x, y, z don't force those of c_i and c_i'. So further add the fresh colors a_i and $\bar{a}_i$, and the queries
  - $a_i a_i \bar{a}_i \bar{a}_i * * * \cdots$ with score (1),
  - $c_i \bar{c}_i' a_i * * * \cdots$ with score (2).

#3-SAT $\leqslant_{pars}$ #MSP-WHITE

- For each clause $C_i = (x \vee y \vee z)$ in φ, add fresh colors $c_i, c_i', \bar{c}_i, \bar{c}_i'$, and the queries
  - $c_i c_i \bar{c}_i \bar{c}_i * * * \cdots$ with score (1),
  - $c_i' c_i' \bar{c}_i' \bar{c}_i' * * * \cdots$ with score (1),
  - $xyz c_i c_i' * * * \cdots$ with score (3).
  - This is still not parsimonious, because the values of x, y, z don't force those of c_i and c_i'. So further add the fresh colors a_i and $\bar{a}_i$, and the queries
  - $a_i a_i \bar{a}_i \bar{a}_i * * * \cdots$ with score (1),
  - $c_i \bar{c}_i' a_i * * * \cdots$ with score (2).
- Let $n = \frac{c-1}{2}$.
- The increase in input size is at most quadratic.

#3-SAT $\leqslant_{pars}$ #3-MSP-BLACK

- We will use only colors 1 and 0 for true and false, and the mask color $*$.
  - Add $***\cdots$ with score (0).

#3-SAT $\leqslant_{pars}$ #3-MSP-BLACK

- We will use only colors 1 and 0 for true and false, and the mask color $*$.
  - Add $***\cdots$ with score (0).
- We adapt the previous reduction by transforming colors into *positions* in the code.
- For each pair of colors x and $\bar{x}$, add the query

$\cdot$	$\cdot$	$\cdot$	x	$\bar{x}$	$\cdot$	$\cdot$	$\cdot$	Score
$*$	$*$	$*$	1	1	$*$	$*$	$*$	(1)

#3-SAT $\leqslant_{pars}$ #3-MSP-BLACK

- We will use only colors 1 and 0 for true and false, and the mask color $*$.
  - Add $***\cdots$ with score (0).
- We adapt the previous reduction by transforming colors into *positions* in the code.
- For each pair of colors x and $\bar{x}$, add the query

$\cdot$	$\cdot$	$\cdot$	x	$\bar{x}$	$\cdot$	$\cdot$	$\cdot$	Score
$*$	$*$	$*$	1	1	$*$	$*$	$*$	(1)

- Convert all other queries of the form $x\bar{y}z***\cdots$ with score (k) into the query

$\cdot$	$\cdot$	x	$\bar{x}$	$\cdot$	y	$\bar{y}$	$\cdot$	z	$\bar{z}$	$\cdot$	$\cdot$	Score
$*$	$*$	1	$*$	$*$	$*$	1	$*$	1	$*$	$*$	$*$	(k)

$\#3\text{-SAT} \leqslant_{pars} \begin{cases} \#2\text{-MSP-BLACK} \\ \#2\text{-MSP} \end{cases}$

- We have to remove the mask color and use only 1 and 0. Let $2n$ be the length of the code in the previous reduction. Start with the query
  - $000\cdots$ with scores (n) and $(n, 0)$.
  - (The code is made of n 0's and n 1's.)

$\#3\text{-SAT} \leqslant_{pars} \begin{cases} \#2\text{-MSP-BLACK} \\ \#2\text{-MSP} \end{cases}$

- We have to remove the mask color and use only 1 and 0. Let $2n$ be the length of the code in the previous reduction. Start with the query
  - $000\cdots$ with scores (n) and $(n, 0)$.
  - (The code is made of n 0's and n 1's.)
- For every variable x, add the query

$\cdot$	$\cdot$	$\cdot$	x	$\bar{x}$	$\cdot$	$\cdot$	$\cdot$	Score 1	Score 2
0	0	0	1	1	0	0	0	(n)	$(n, 2)$

$\#3\text{-SAT} \leqslant_{pars} \begin{cases} \#2\text{-MSP-BLACK} \\ \#2\text{-MSP} \end{cases}$

- We have to remove the mask color and use only 1 and 0. Let $2n$ be the length of the code in the previous reduction. Start with the query
  - $000\cdots$ with scores (n) and $(n, 0)$.
  - (The code is made of n 0's and n 1's.)
- For every variable x, add the query

$\cdot$	$\cdot$	$\cdot$	x	$\bar{x}$	$\cdot$	$\cdot$	$\cdot$	Score 1	Score 2
0	0	0	1	1	0	0	0	(n)	$(n, 2)$

- For all other queries, do the following conversion:

$\cdot$	$\cdot$	x	$\cdot$	$\cdot$	$\bar{y}$	$\cdot$	$\cdot$	Score 1	Score 2
$*$	$*$	1	$*$	$*$	1	$*$	$*$	(k)	
0	0	1	0	0	1	0	0	$(n+1)$	$(n+1, 2k-2)$

  - Caveat: the number of 1's is always $2k-1$.
- Check that these scores work...

Example

The formula $(x \vee \neg y \vee z) \wedge (y \vee \neg z)$ yields the queries

x	$\bar{x}$	y	$\bar{y}$	z	$\bar{z}$	c_1	$\bar{c}_1$	c_1'	$\bar{c}_1'$	a_1	$\bar{a}_1$	c_2	$\bar{c}_2$	Score
0	0	0	0	0	0	0	0	0	0	0	0	0	0	$(7,0)$
1	1	0	0	0	0	0	0	0	0	0	0	0	0	$(7,2)$
0	0	1	1	0	0	0	0	0	0	0	0	0	0	$(7,2)$
0	0	0	0	1	1	0	0	0	0	0	0	0	0	$(7,2)$
0	0	0	0	0	0	1	1	0	0	0	0	0	0	$(7,2)$
0	0	0	0	0	0	0	0	1	1	0	0	0	0	$(7,2)$
0	0	0	0	0	0	0	0	0	0	1	1	0	0	$(7,2)$
0	0	0	0	0	0	0	0	0	0	0	0	1	1	$(7,2)$
1	0	0	1	1	0	1	0	1	0	0	0	0	0	$(8,4)$
0	0	0	0	0	0	1	0	0	1	1	0	0	0	$(8,2)$
0	0	1	0	0	1	0	0	0	0	0	0	1	0	$(8,2)$
x	$\bar{x}$	y	$\bar{y}$	z	$\bar{z}$	c_1	$\bar{c}_1$	c_1'	$\bar{c}_1'$	a_1	$\bar{a}_1$	c_2	$\bar{c}_2$	Score

Example

The satisfying assignments for the formula are 5:

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

The solutions of the queries are also 5:

x	$\bar{x}$	y	$\bar{y}$	z	$\bar{z}$	c_1	$\bar{c}_1$	c_1'	$\bar{c}_1'$	a_1	$\bar{a}_1$	c_2	$\bar{c}_2$
0	1	0	1	0	1	1	0	1	0	1	0	1	0
0	1	1	0	1	0	1	0	1	0	1	0	1	0
1	0	0	1	0	1	1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	0	1	0	1	0	0	1
1	0	1	0	1	0	1	0	0	1	0	1	1	0

Size assumptions

- In a real game of Mastermind we would *know* that our queries are satisfiable. Can we use this information to compute the size of the solution space?
  - In general, is it easier to compute the number of solutions, with the knowledge that they're at least k?

Size assumptions

- In a real game of Mastermind we would *know* that our queries are satisfiable. Can we use this information to compute the size of the solution space?
  - In general, is it easier to compute the number of solutions, with the knowledge that they're at least k?
- Let #MATCH be the problem of counting all the matchings (perfect and imperfect) in a given graph.

Lemma (Valiant, 1979)

#MATCH is #P-complete under Turing reductions.

Size assumptions

- In a real game of Mastermind we would *know* that our queries are satisfiable. Can we use this information to compute the size of the solution space?
  - In general, is it easier to compute the number of solutions, with the knowledge that they're at least k?
- Let #MATCH be the problem of counting all the matchings (perfect and imperfect) in a given graph.

Lemma (Valiant, 1979)

#MATCH is #P-complete under Turing reductions.

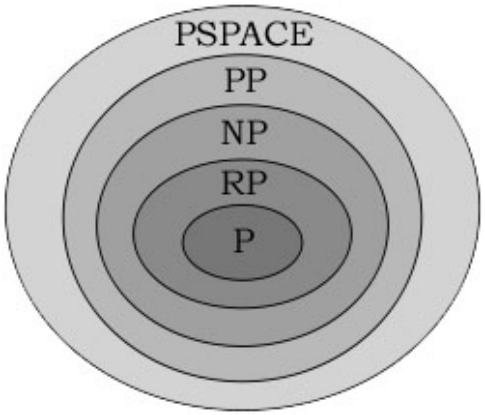
- Then $\#\text{SAT} \leqslant_T \#\text{MATCH} \leqslant_{pars} \#\text{SAT} \leqslant_{pars} \#\text{MSP}$.
- Moreover, there are only a finite number of graphs with less than k edges, and the others have at least k matchings.

Theorem

#MSP restricted to instances with at least k solutions is #P-hard.

Conclusions

- If there is a randomized algorithm to determine if a given set of Mastermind queries has a unique solution, then RP=NP.
- Computing the solution of a sequence of Mastermind queries, under the assumption that it's unique, is NP-hard under randomized reductions.
- Even assuming that a sequence of Mastermind queries has at least k solutions, they still can't be counted in polynomial time, unless P=PP.
- The same holds if the colors used are only 2, and also for the variants of Mastermind where the codemaker responds only with black or white pegs.



Further work

- Solving MSP was a sub-step of several heuristics.
  - But was it really necessary? Can't Mastermind be solved in other ways?

Further work

- Solving MSP was a sub-step of several heuristics.
  - But was it really necessary? Can't Mastermind be solved in other ways?

MASTERMIND

Input: (n, c, Q, k).
Output: YES *if the codebreaker has a strategy to guess the code within* k *attempts, given the set of queries* Q. NO *otherwise.*

- We pretend that the codemaker can change his code during the game, but coherently with his previous answers. Thus we conjecture that MASTERMIND is PSPACE-complete.
  - Caveat: by Chvátal's method, k can be polynomially bounded.

References for Mastermind

- Chvátal, *Mastermind*, 1983.
- Focardi, Luccio, *Cracking bank PINs by playing Mastermind*, 2010.
- Goodrich, *On the Algorithmic Complexity of the Mastermind Game with Black-Peg Results*, 2009.
- Knuth, *The Computer as a Master Mind*, 1976.
- Stuckman, Zhang, *Mastermind is NP-Complete*, 2005.

References for counting problems

- Dyer et al., *On the Relative Complexity of Approximate Counting Problems*, 2000.
- Valiant, *The Complexity of Computing the Permanent*, 1979.
- Valiant, *The Complexity of Enumeration and Reliability Problems*, 1979.
- Valiant, Vazirani, *NP is as easy as detecting unique solutions*, 1986.