Defeating Mastermind

By Justin Dowell

Executive Summary

The purpose of this paper is to describe the game Mastermind, the problems involved in making a guess towards the solution, discussion of various previous algorithms, and the introduction of two new algorithms.

Mastermind is a game played by two players, a 'Code Master' and a 'Code Breaker'. The game is played using effectively unlimited pegs of six colors on a board containing eleven rows of four holes. These holes are used for the Code Breaker's guessing, and a matching number of holes are beside the rows providing space for the Code Master's reply information, which helps the Code Breaker determine the code behind hidden on the eleventh row. The information returned consists of a black key for each correctly colored peg in a correct hole, a white key for each correctly colored peg in an incorrect hole (not including pegs that received black keys), and no key for an incorrectly colored peg.

The goal for the Code Breaker is to deduce the hidden code and make the guess exactly matching it, receiving a reply of black keys equal to the number of holes ((4,0) in the classic four-hole game). If the Code Breaker cannot do this with the provided number of spaces for guesses to be played, the Code Master wins instead.

This paper focuses on the problem of the Code Breaker, and the various algorithms that are can be used to determine a near-optimal next guess for the Code Breaker to use, along with what methodologies they follow and why.

Much work has been done on the subject of creating algorithms that recommend guesses for the Code Breaker, such as the MiniMax strategy (Knuth, 1977), Entropy strategy (Bestavros and Belal, 1986), Irving's strategy (Irving, 1979), Simple strategy (Kooi, 2005), MaxParts strategy (Kooi, 2005), Optimal Depth strategy (Koyama and Lai, 1993), and Optimal Mean strategy (Koyama and Lai, 1993). This paper includes two new strategies that compare favorably to the previous work.

Some statistical information about the algorithms is shown below, including total number of guesses to solve every game, mean average guesses to solve a game, standard deviation of game length, first guess recommended by an algorithm, maximum game length, and the number of games that last a given number of turns in the remaining columns.

						Games completed in N turns								
Algorithm	#Guesses	Mean	StdDev	First Move	Max Turns	1	2	3	4	5	6	7	8	9
MiniMax	5778	4.458	0.607	AABB	5	1	6	54	572	663	0	0	0	0
MaxEnt	5723	4.416	0.631	ABCD	6	1	4	70	613	596	12	0	0	0
Irving	5696	4.395	0.619	AABC	6	1	10	54	645	583	3	0	0	0
Simple	7471	5.765	1.049	AAAA	9	1	4	25	108	305	602	196	49	6
MaxParts	5668	4.373	0.649	AABC	6	1	12	72	635	569	7	0	0	0
WideDev	5685	4.387	0.627	AABC	6	1	11	59	643	579	3	0	0	0
LongRect	5681	4.383	0.623	AABC	6	1	11	57	650	575	2	0	0	0
OptDepth	????	4.341	?.???	AABC	5	1	?	?	?	?	0	0	0	0
OptMean	????	4.340	?.???	AABC	6	1	?	?	?	?	1	0	0	0

Based on the statistical information gathered on all of the algorithms, it is my recommendation to use MaxParts when attempting to win a game in a low number of turns, and MiniMax if the number of turns cannot exceed five. OptMean and OptDepth both provide perfect information, but are prohibitively expensive (several orders of magnitude of time longer) to calculate guesses for. For those in possession of the preprocessed guesses of either optimal strategy, they should replace MaxParts and MiniMax respectively.

Problem Description

Mastermind is a game played by two players, a 'Code Master' and a 'Code Breaker'. The game is played using effectively unlimited pegs of six colors on a board containing eleven rows of four holes. These holes are used for the Code Breaker's guessing, and a matching number of holes are beside the rows providing space for the Code Master's reply information, which helps the Code Breaker determine the code behind hidden on the eleventh row. The information returned consists of a black key for each correctly colored peg in a correct hole, a white key for each correctly colored peg in an incorrect hole (not including pegs that received black keys), and no key for an incorrectly colored peg.

The goal for the Code Breaker is to deduce the hidden code and make the guess exactly matching it, receiving a reply of black keys equal to the number of holes ((4,0) in the classic four-hole game). If the Code Breaker cannot do this with the provided number of spaces for guesses to be played, the Code Master wins instead.

This paper focuses on the problem of the Code Breaker, and the various algorithms that are can be used to determine a near-optimal next guess for the Code Breaker to use, along with what methodologies they follow and why. The paper also introduces two new algorithms that perform comparably well.

Analysis Technique

Mastermind is a game played by two players, a 'Code Master' and a 'Code Breaker'. The game is played using effectively unlimited pegs of six colors on a board containing eleven rows of four holes. Next to all rows but the final are an equal number of smaller holes. The Code Master initiates the game by placing any arrangement of the colored pegs behind the shield before the final row (henceforth known as the hidden code), after which the Code Breaker must attempt to guess the hidden code by placing pegs into the holes on the other side of the screen (thus retaining the history of the game so far). After each guess, the Code Master provides information based on the accuracy of the Code Breaker's guess. This information is recorded to the side of the rows in the smaller holes, and consists of two parts: A black 'Key' is awarded for each peg that is the same color in the same position as in the hidden code. A white key is awarded for each peg that is the correct color, but not in the correct place. White keys are not awarded for pegs that have been awarded black keys. No keys are awarded for pegs of incorrect colors. The information returned in the form of keys is not in any particular order, and can be simplified by simply expressing the number of black and white keys. For example, if the Code Breaker guesses "ABBB" when the code is "BBAA", one black key and two white keys are returned. I will write these arrangements of keys as (Black, White) and refer to them as 'replies'. In the example, the guess has a reply of (1, 2).

The goal for the Code Breaker is to deduce the hidden code and make the guess exactly matching it, receiving a reply of black keys equal to the number of holes ((4,0) in the classic four-hole

game). If the Code Breaker cannot do this with the provided number of spaces for guesses to be played, the Code Master wins instead.

In an example game, pretend the hidden code is encoded using 'ROYGBV' as colors. The hidden code is chosen to be ROYY.

					Keys	
Hidden Code	R	O	Y	Y	Blk	Wht
Guess 10						
Guess 9						
Guess 8						
Guess 7						
Guess 6	R	O	Y	Y	4	0
Guess 5	R	O	V	V	2	0
Guess 4	O	R	O	O	0	2
Guess 3	R	O	O	O	2	0
Guess 2	G	G	B	B	0	0
Guess 1	R	R	B	B	1	0

The Code Breaker decides to guess RRBB first to get a feel for what colors are in the hidden code (Guess 1). The reply of (1, 0) tells her that one of these pegs is correct. She wants to narrow down which color was correct, so tries guessing GGBB and gets (0, 0) (Guess 2). She now knows that not only are there no blues or greens in the hidden code, but that a red goes in one of the first two spaces. In order to figure out what other colors are in the code and which spot the red peg goes in, she next guesses ROOO and receives a reply of (2, 0) (Guess 3). The red is in the correct space, and there's an orange somewhere in the rest of the code. In order to figure out where, her next guess of OROO switches the red peg's position. She receives a reply of (0, 2) (Guess 4), so knows that the orange and red pegs are no longer in the correct place- the first two colors in the hidden code must be 'RO', in that order. Next, she tries to determine what other colors are in the hidden code by guessing ROVV (Guess 5). She gets a reply of (2, 0), so knows the only remaining code could be ROYY, because all the other colors (G, B, V) are eliminated. Making the guess of ROYY, she gets a reply of (4, 0) and wins the game (Guess 6).

While two optimal strategies for the specific game of four holes and six colors have been found (Koyama and Lai, 1993), they were discovered using an exhaustive, depth-first search. Other algorithms, such as the ones discussed in this paper, look only a single step in advance.

While the algorithms perform significantly differently in terms of their measurements of a better guess, they share the same information-gathering process. They also share the same optimization for the first guess. The generalized algorithm is as follows:

1. Generate a Master List, which contains all possible Mastermind guesses. (AAAA, AAAB, AAAC, ... AAAF, AABA, …, AAFF, …, FFFF).

2. Generate a Code List, which is all remaining possible codes that could still be the solution. On the first turn, this is equal to the Master List (as all codes remain possible), shrinks after each turn, and by the end of the game is reduced to a single element (the solution).

3. Generate a Guess List, which is equal to the Master List (see explanation below) for all turns but the first. The first guess differs from later guesses because no information about the hidden code is known therefore many initial guesses have identical measurable statistics. More specifically, these are guesses where pegs have been placed into different order, or where pegs of a certain color are replaced with another color. Consequently, the first turn's Guess List can be reduced to a few elements that represent the generic-colored, ordered arrangements of peg patterns.

4. Iterate through the Guess List, performing the following:

    a) Place the elements of the Code List into different partitions based on the replies they would generate if they were the solution, given this particular element of the Guess List was the guess.

    b) Derive algorithm-specific statistics based on the reply-partitions, such as number of partitions, number of elements in each partition, entropy-based gain from each partition, etc.

    c) Compare the calculated worth of the current guess compared to the best previous stored. Record the better of the two guesses, with ties falling to previously stored. Breaking a tie by simply using the previously stored guess is arbitrary, and does have a slight impact on the results.

5. Recommend the stored guess.

About the Guess List: In the past, there was the suggestion to only use codes that remain as potential hidden codes for the Guess List. This speeds up processing time significantly, as the calculations for any turn are only the number of elements in the Code List squared. However, there can exist a Code List that can be more effectively divided by a guess that is not within its own elements.

The following example is provided by (Kooi, 2005), showing the Code List on the top row and equal Guess List in the first column, with replies in the intersection:

Guess	Hidden Code					
	ABAA	ABAB	ABAF	ABDE	AEAE	AFAE
ABAA	(4, 0)	(3, 0)	(3, 0)	(2, 0)	(2, 0)	(2, 0)
ABAB	(3, 0)	(4, 0)	(3, 0)	(2, 0)	(2, 0)	(2, 0)
ABAF	(3, 0)	(3, 0)	(4, 0)	(2, 0)	(2, 0)	(2, 1)
ABDE	(2, 0)	(2, 0)	(2, 0)	(4, 0)	(2, 0)	(2, 0)
AEAE	(2, 0)	(2, 0)	(2, 0)	(2, 0)	(4, 0)	(3, 0)
AFAE	(2, 0)	(2, 0)	(2, 1)	(2, 0)	(3, 0)	(4, 0)

Looking across the rows, duplicate replies can easily be spotted. Using only the elements of the Code List is insufficient to determine the code with one more guess. However, a guess outside the Code List can do this:

	ABAA	ABAB	ABAF	ABDE	AEAE	AFAE
ABFA	(3, 0)	(2, 1)	(2, 2)	(2, 0)	(1, 1)	(1, 2)

As a result, the MaxEnt strategy described in Bestavros and Belal's (1986) work was changed to check the entire Master List as guesses instead of the Code List. Note that the 'Simple' strategy only chooses elements from the Code List.

The above is the general scheme used by all solver methods. The basis for different algorithms is their individual metrics for judging a best guess. Here are short descriptions of the various algorithms that have been developed:

MiniMax

Invented by Donald Knuth, this was the first Mastermind algorithm. The algorithm attempts to maximize the minimum amount of information a guess could return by comparing the maximal reply-partitions of guesses to see which is smaller. Smaller maximal partitions means the

potential remaining codes are narrowed down further. If the first-largest partitions are of equal size, the second-largest partitions are compared and so on in case of more ties (Knuth, 1977).

MaxEnt

Developed by Azer Bestavros and Ahmed Belal, this algorithm tries to maximize the greatest information gain based on Shannon Entropy. The probability of receiving each partition (its size divided by the number of elements in the Code List) is multiplied with the information in bits that would be gained if the reply generated by the guess led to that partition being the new Code Pool. These values are then summed to get the value of the guess. Higher values mean greater entropy of the original Code List- more information gained (Bestavros and Belal, 1986).

Irving

Invented by Rob Irving, this method bases the worth of a guess on the expected size of the next partition to work with should that guess be chosen. A smaller partition is preferable to a larger one (with the best case being partition of one element). The sizes of partitions are squared, then multiplied by their relative probability of occurring (which is partition size divided by the total of partition sizes) (Irving, 1979).

Simple

The Simple strategy is as its name suggests. The algorithm simply chooses the first available guess that could still be the code. Note that the 'Simple' strategy only chooses elements from the Code List (Kooi, 2005).

MaxParts

Invented by Barteld Kooi, this algorithm operates on the simple concept that the best guess is the one that has as many partitions as possible. In other words, the guess that has the most potential different replies is ranked the highest (Kooi, 2005).

The MaxParts strategy does surprisingly well in comparison to the others (see results). However, it suffers a flaw in the first guess. For the first guess in the classic game, the possible five guesses are shaped as 'AAAA', 'AAAB', 'AABB', 'AABC', and 'ABCD', where the letters are uniquely replaced with colors. While MaxParts chooses the most optimal guess of 'AABC' (Koyama and Lai, 1993) for the first turn while the Guess List is ordered 'alphabetically', it picks a less than optimal guess if the order is reversed, instead choosing 'ABCD'.

My strategies both were attempts to improve upon the MaxParts algorithm by adding a metric for correctly handling the first guess no matter how the Guess List was ordered. I created two new methodologies to this end:

WideDev

A hybrid strategy, this algorithm takes the guess with the maximum number of partitions and lowest standard deviation of partitions. This effectively means taking the 'flattest' distribution of replies. The underlying idea is that a choice that most evenly splits into the most partitions creates a Code List that is most equally handled in the next turn. This is based in part on a suggestion to use the guess with the lowest standard deviation of partition sizes made by (Heeffer and Heeffer, 2007), whose algorithm is excluded.

LongRect

A hybrid strategy of MaxParts and MiniMax, this algorithm takes the guess with the maximum number of partitions and smallest maximal partition. The concept was that the guess with the most partitions and maximum smallest reduction of the Code List would provide a more 'risk free' approach to a guess.

Surprisingly, neither algorithm does as well as MaxParts itself when the Guess List is in ascending order. However, both perform better in comparison when the Guess List is in descending order because of the 'lucky guess' of 'AABC' that no longer is taken by MaxParts. Both of the introduced algorithms correctly select 'AABC' as the first guess in either ordering of the Guess List.

While creating the algorithms, I took note of the relatively little work towards the optimization of the first guess for the general game (with any number of holes and colors). In order to facilitate this, I wrote a function to determine the generic guess patterns. This set is used to directly obtain the first turn's Guess List. The function runs deterministically in polynomial time.

What follows is the statistics and first guess chosen by each algorithm. The table is duplicated in the Executive Summary and Results section. This is when the Guess List is in ascending order. I could not re-obtain results for the Guess List in descending order due to changes in my program. They may be included in future work.

						Games completed in N turns								
Algorithm	#Guesses	Mean	StdDev	First Move	Max Turns	1	2	3	4	5	6	7	8	9
MiniMax	5778	4.458	0.607	AABB	5	1	6	54	572	663	0	0	0	0
MaxEnt	5723	4.416	0.631	ABCD	6	1	4	70	613	596	12	0	0	0
Irving	5696	4.395	0.619	AABC	6	1	10	54	645	583	3	0	0	0
Simple	7471	5.765	1.049	AAAA	9	1	4	25	108	305	602	196	49	6
MaxParts	5668	4.373	0.649	AABC	6	1	12	72	635	569	7	0	0	0
WideDev	5685	4.387	0.627	AABC	6	1	11	59	643	579	3	0	0	0
LongRect	5681	4.383	0.623	AABC	6	1	11	57	650	575	2	0	0	0
OptDepth	????	4.341	?.???	AABC	5	1	?	?	?	?	0	0	0	0
OptMean	????	4.340	?.???	AABC	6	1	?	?	?	?	1	0	0	0

Assumptions

Tie-breaking between guesses that appear of equal value to an algorithm can be resolved by retaining the previously stored guess as opposed to some other process without significantly altering the results. This might be a bold assumption; an optimal strategy may have a way to measure a true 'best guess'. More research on (Koyama and Lai, 1993) is necessary.

Results

What follows is the statistics of and first guess chosen by each algorithm. This is when the Guess List is in ascending order. I could not re-obtain results for the Guess List in descending order due to changes in my program. They may be included in future work.

						Games completed in N turns								
Algorithm	#Guesses	Mean	StdDev	First Move	Max Turns	1	2	3	4	5	6	7	8	9
MiniMax	5778	4.458	0.607	AABB	5	1	6	54	572	663	0	0	0	0
MaxEnt	5723	4.416	0.631	ABCD	6	1	4	70	613	596	12	0	0	0
Irving	5696	4.395	0.619	AABC	6	1	10	54	645	583	3	0	0	0
Simple	7471	5.765	1.049	AAAA	9	1	4	25	108	305	602	196	49	6
MaxParts	5668	4.373	0.649	AABC	6	1	12	72	635	569	7	0	0	0
WideDev	5685	4.387	0.627	AABC	6	1	11	59	643	579	3	0	0	0
LongRect	5681	4.383	0.623	AABC	6	1	11	57	650	575	2	0	0	0
OptDepth	????	4.341	?.???	AABC	5	1	?	?	?	?	0	0	0	0
OptMean	????	4.340	?.???	AABC	6	1	?	?	?	?	1	0	0	0

It appears that MaxParts still performs the best out of the algorithms so far, unless the player is seeking to win for sure in five moves or less, in which case MiniMax is recommended. For those in possession of the optimal strategies, OptMean or OptDepth should be used in replace of MaxParts or MiniMax, respectively.

Issues

I was not able to duplicate the results of having the Guess List sorted in descending order due to changes I had made in my program. While not especially important, it would have been useful to show how MaxParts drops in effectiveness.

Appendices

Future Work:

I could not re-obtain results for the Guess List in descending order due to changes in my program. They may be included in future work.

Alter Bestavros and Belal's (1986) algorithm to use a logarithm base equal to the number of partitions, as suggested in Kooi's (2005) work

Improve the framework such that any arbitrary algorithm weighted by different metrics can be used. A genetic algorithm could evolve the coefficients towards the optimal algorithm.

Allowing a genetic algortithm to create a code (different from the above in that the code itself is being modified, as opposed to the metrics that codes are based off of).

Creation of a score-based algorithm (based on black and white keys most probably received).

Altering the framework in such a way that is it possible to check more than one move ahead.

The big question for future work is: Can optimal guesses be found without looking all the way through the game?

References

Knuth, D.E. (1976). The computer as Mastermind, *Journal of Recreational Mathematics,* 9(1), 1–6.

Irving, R.W. (1978-79). Towards an optimum Mastermind strategy, *Journal of Recreational Mathematics,* 11(2), 81–87.

Koyama, K., and Lai, T.W. (1993). An optimal Mastermind strategy, *Journal of Recreational Mathematics,* 25, 251–256.

Bestavros, A., and Belal, A., (1986). MasterMind: A Game of Diagnostic Strategies, *Bulletin of the Faculty of Engineering*, Alexandria University, Alexandria, Egypt.

Heeffer, A., and Heeffer, H., (2007). NEAR-OPTIMAL STRATEGIES FOR THE GAME OF LOGIK, *ICGA Journal*. doi: 10.1.1.71.9209

Kooi, B., (2005). Yet another mastermind strategy, *ICGA Journal,* 28(1), 13–20.