Implementation of Knuth's and minimax algorithms in Mastermind game

Agnieszka Ostrowska

May 2021

1 Introduction

Mastermind is a popular logic game that over the years has gained several proposals of algorithms that can automatically solve the puzzle. In this paper I will describe the implementation of an algorithm based on systematic searching through all possible moves at a given stage of the game and choosing the one that causes less harm and still will provide quite fast solution. Moreover, I will also present a comparison of the algorithm's operation for different versions of the game (with different number of colours and code length).

2 Mastermind

Mastermind is an reasoning game in which the first player creates the code (codemaker), while the second one tries to guess it (codebreaker) (Knuth, 1977). The secret codes as well as guesses contain four pegs and are generated from the six possible colours. After each move of the codebreaker, the codemaker is giving a feedback to tell how close is codebreaker to solve the puzzle. The rules for feedback are as follows:

> let's say the secret code is: x_1, x_2, x_3, x_4 and the code in the current move is y_1, y_2, y_3, y_4, then:
>
> 1. if $y_n = x_n$, give black peg in the feedback
> 2. if $y_n \neq x_n$ and y_n is in secret code, give white peg in the feedback

So for example, if the secret code is 1432 and the first guess to solve it is 2536, the feedback would be: one *white* – because in both codes there is 2 but in different positions and one *black* – because in both codes there is 3 and in the same spot.

The game ends when the codebreaker guesses the code (so he gets four black pegs as a feedback) created by the codemaker or there are no more moves left as the game restricts possible number of guesses for a codebreaker (there are versions with different limits of moves).

3 Knuth's algorithm

In seventies of XX century, Donald Knuth presented a way for identifying the secret code in the Mastermind game in five or less moves (Knuth, 1977). The main idea is to create a list

S of all possible codes ($4^6 = 1296$) and then narrow down possible codes after each move. As a move it chooses an element of the list S that has the best *worst–case scenario*. The worst–case scenario works as follows:

> let's say our set of still possible answers S_1 consists of twelve codes. Then we choose first code $s_1 \in S_1$ from the list and pretend it is our next guess, and our last actual move is a secret code. Then we compute all possible feedbacks in this scenario and choose the one that was the most frequent, e.g. *white, white* appeared four times. As a score for $s_1 = 12 - 4$ because in this case, no matter what feedback we will get after choosing it as our next move we will eliminate at least eight codes from S_1 (min part from minimax algorithm).
> Next, we repeat it with every $s \in S_1$ and after that choose the code with the highest score as our next move in the game (max part from minimax algorithm).

This scenario guarantees that the secret code will be guessed in five or less moves (it was designed for a version with six colours and four–value code).

4 Minimax algorithm

Minimax is a decision rule which task is to minimise the maximum possible loss (Smed and Hakonen, 2006). It can be also described as a heuristic algorithm in which the *min* part is responsible for minimizing the max score, while the *max* part is trying to maximize the advantage (Luger, 2008); the algorithm will choose the move that will result in the best possible outcome. For example if there is a move that will lead to winning it will choose it. If winning at this stage is not possible, the algorithm will chose the move that makes the least damage to the current state of the game (Smed and Hakonen, 2006).

As an example, it can be applied to games or problems in which a state space can be exhaustively searched. That means players know all the previous moves and which moves are available, so all of the information is on a board and can be displayed in the form of a tree. The root node is the current or initial position, the leaves represent all next possible moves (Smed and Hakonen, 2006).

In the more complicated games, where it is impossible to show all possible moves, *As the leaves of the tree are not final states of the game, it is not possible to give them values that reflect a win or a loss. Instead, each node is given a value according to some heuristic evaluation function* (Luger, 2008, p.153). The taken move does not have to lead to winning. It only is the best possible choice that cause the least harm in this particular state of the game.

5 Mastermind implementation explained

The algorithm[1]starts with generating a list with all possible four–value sets of colours: green, orange, blue, brown, yellow and red. That means at the beginning we have 1296 codes. At this stage, it also creates (a four–value) random secret code that has to be guessed.

There are eight possible moves to take, although algorithm is designed in a way it should solve the puzzle in no more than five moves. First step of the codebreaker is to generate the first guess by filling two spots with one random color and the next two spots with another

[1]https://github.com/ostrowskaaa/Mastermind-minimax-algorithm

random color, e.x. *brown, brown, orange, orange*[2]. Next, the codemaker gives a feedback. With this in mind the codebreaker removes from the list S with all possible codes its move and all codes that would not give the same feedback as the move that has been just done (let's call the narrowed list S_1). It is done by iterating the list with all possible codes (minus the first move) as guesses with the first move as a secret code.

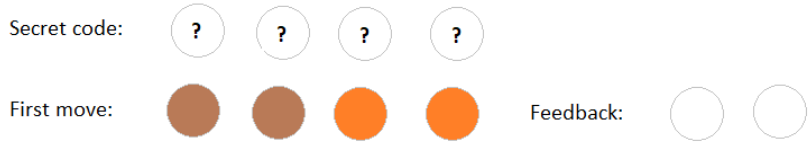

Figure 1: First move.

Figure 2: Example of checking if a combination can still be an answer.

Example
Let's say our first move is *brown, brown, orange, orange* and we get *white, white* as a feedback (see Fig. 1). Now we take the list S with all 1296 of possible codes and try them one by one as a new guess while our first move pretends to be a secret code (see Fig. 2). It is done due to narrow all codes to the list of codes that can still be an answer, so it is not a second move yet. As an example, one of the possible codes is *brown, brown, orange, brown* and the feedback would be *black, black, black*. Because the new feedback in not the same as the first feedback we got (*white, white*), the combination *brown, brown, orange, brown* is removed from the list S.
In the situation from Fig. 1, all of the codes that have more then two brown or orange pegs would be removed from the list S as well as the codes that does not have two brown or two orange or one orange and one brown. It is because the first feedback (*white, white*) indicates there are at two spots filled with the colours used in the first move but are wrongly placed.

After narrowing all the possible codes to the ones which still can be an answer, it is time to make another guess. Here starts the minimax algorithm. It takes the list S_1 with codes

[2]Knuth has proven that choosing a code with only two colours in the first move is an optimal way to solve the game. This is because after such move there will be (in any scenario) 256 or fewer possible codes, while for example code with three colors (e.x. *brown, brown, orange, yellow*) will result in 276 or fewer possible codes.

that have been left and the list S with all possible codes. Now, it takes first possible code from the list S_1 and mark it as a secret code, then it tries codes from the list S one by one as a guess. All feedbacks are collected in another list and when the last code is checked it chooses the most frequent feedback from the list. Then it calculates the score which is the number of possible codes in the list S_1 minus the number that indicates how many times the most frequent feedback appeared. The score with checked code is saved to another list for later. The whole procedure is repeated with the second possible code from the S_1 list and so on as long as there are codes to check. Later, it chooses the code with the highest score as the next move.

So the algorithm can be described simpler as follows:

1. Generate list of all possible codes from the available colors that is four value–long.
2. Generate first random move that is made from two colours.
3. Give a feedback.
4. If the feedback is four blacks, the game is won and algorithm terminates. If not, goes to step 5.
5. Narrow down all possible codes to the ones that can still be an answer.
6. Calculate the best possible move.
7. Give a feedback.
8. If feedback does not contain four black, repeat all the moves from 5 to 7.

This algorithm is not the most computationally efficient implementation as it is making operations on quite big lists of data.

6 Compilation of results with different settings

	Four colours and three–value code	Six colours and three–value code
Moves	3.39	4.23
Time [s]	0.0179	0.1929
Possible codes	$4^3 = 64$	$6^3 = 216$
	Four colours and four–value code	Six colours and four–value code
Moves	3.79	4.6
Time [s]	0.1737	5.1278
Possible codes	$4^4 = 256$	$6^4 = 1296$
	Four colours and five–value code	Six colours and five–value code
Moves	4.13	5.0
Time [s]	2.2889	162.9688
Possible codes	$4^5 = 1024$	$6^5 = 7776$

Table 1: Comparison of average time,number of moves and possible combinations per game.

Each increase in the number of colours or/and length of the code causes growth of the solution space: a) in cases where the code length is increased, the solution space grows exponentially; b) in cases where the number of colours is increased, the solution space grows polynomially; c) in cases where both the number of colours and code lenght is increased, the solution space grows spatially as the graph of the function is in 3D space which makes it hard to define the character of this growth (although this growth is the biggest from these three cases).

The numbers in the Table 1, related to time and moves, have been calculated based on the results of 100 iterations of Mastermind game for each type: four colours and three–value code, six colours and three–value code, four colours and four–value code, six colours and four–value code, four colours and five–value code, six colours and five–value code. The only deviation is the last version with six colours and five–value code which was calculated on 55 iterations. It is because the time for each game was quite high and it was obvious the results will not be good.

The algorithm works well for each situation, in no case does the average number of moves exceeds 5 moves. The only problem is that the growing number and length of possible codes makes the algorithm run time considerably longer, especially when both the number of colours and code length are increased. Analysis of obtained results implicates that increasing the code length increases from 27 (three-value to four–value code), 31 (four-value to five–value code) times the run time of the algorithm with six colours, and from 9 (three-value to four–value code) to 13 (four-value to five–value code) times with four colours. The time extension is related to an exponentially increasing number of codes in the list with all possible codes. However, changing the number of colours for the same length of the code seems to affect the time more drastically as for example for five–value code it increases about 71 times, for four–value code about 29.5 times and for three–value code about 11 times. It can be noted that increasing both the code length and the number of possible colors affects the length of time the most. By increasing the number of colors from four to six and the length of the code from three to four, it extends the algorithm's run time by about 286.5 times. And increasing the number of colors from four to six and the length of the code from four to five, it extends the algorithm's run time by about 938 times.

To sum up, the algorithm in most cases is solving the secret code in less than five moves which makes it possible to say that it copes well with the given problem. Although, increasing the number of colours and the length of the code makes it computationally inefficient. However, its main goal is to minimize the needed moves in order to solve the puzzle and in this case it works well as it guesses the secret code in five or less moves. What is more, Mastermind is a game without time limit so implementing an algorithm that quickly solves a puzzle at the expense of time seems like a good solution.

References

Knuth, D. (1977). The computer as master mind. *Journal of Recreational Mathematics*, 9:1–6.

Luger, G. F. (2008). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Addison-Wesley Publishing Company, USA, 6th edition.

Smed, J. and Hakonen, H. (2006). *Game Trees*, chapter 4, pages 73–95. John Wiley Sons, Ltd.