THE APPLICATION OF DEPTH FIRST SEARCH AND BACKTRACKING IN SOLVING MASTERMIND GAME

Halida Astatin (13507049)
Informatics
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Jalan Ganesa 10 Bandung
e-mail: halida.astatin@gmail.com

ABSTRACT

Mastermind is a two-player code-breaking game in which one player acts as a *code-maker*, and the other acts as *code-breaker*. The *code-maker* creates a combination of four colors out of six possible colors, and the *code-breaker* then has to guess the correct color combination within given turns (usually 8-12 turns). The game ends when the *code-breaker* has guessed the right color combination, or run out of turns. This game is also often played with the computer providing an AI (Artificial Intelligence) to generate color combinations. There are many strategies that can be used to guess the correct color combination, and there are many algorithms that can be applied in finding the right color combination. Using the right strategy, and after several adjustments, we can imply backtracking to solve the game. Backtracking is a problem-solving method which is derived from the Depth-First-Search (DFS) algorithm. In finding a solution using backtracking, once a node is proven to not lead to a solution, it will be pruned and therefore the nodes positioned under said node will not be checked. By doing this, the number of possible solutions that has to be checked can be decreased significantly.

Keywords: Mastermind, DFS, backtracking.

1. INTRODUCTION

Mastermind is a game invented in the 1970s by Mordecai Meirowitz, but it resembles the game *bulls and cows*, a pencil and paper game that may date back a century or more. Throughout time, this game has been modified into many different versions with different game names. Aside from guessing color combinations, the variation of mastermind includes word mastermind which challenges the player to guess a four-letter word with 26 possible letters, number mastermind which uses numbers instead of colors, and grand mastermind, which doesn't only use colors but shapes as well.

Many algorithms can be applied in solving this game, one of which is brute force. To find the right answer, brute force algorithm will test each and every possible answer to see if it is the solution. Using brute force, with the assumption that there are six possible colors and four colors in the combination, there would be 6^4 possible answers that need to be tested. This is highly inefficient, therefore in this paper we will discuss the solution to mastermind game using backtracking which supposedly should solve the problem in a faster, more efficient way.

2. THE MASTERMIND GAME AND THE BACKTRACKING ALGORITHM

Before we begin, we need to fully understand how to play the game and what strategy to imply. Therefore in this chapter, we will discuss further about the gameplay and rules of mastermind game. Also in this chapter, we will review the backtracking algorithm.

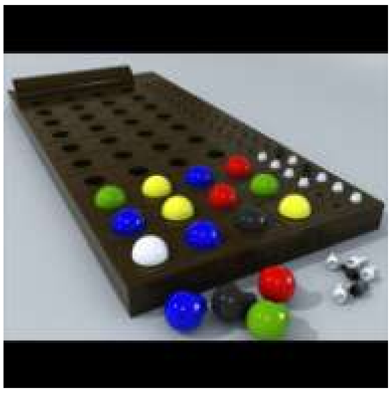

Figure 1 Mastermind Decoding Board, Code Pegs, and Key Pegs

2.1. Mastermind Gameplay and Rules

Mastermind is played in a custom made board called *decoding board*. This board has a shield at one end, covering a row of four holes (this is used to put the right color combination), and more or less twelve guessing rows, depending on how many turns are given to the code-breaker. Each row has four holes for the *code pegs*, and four smaller holes for the *key pegs*. Code pegs are pegs of six different colors with rounded heads which are used to guess the color combination by putting them in the holes on the board. Key pegs are flat-headed pegs, some colored (often black), some white, sized smaller than the code pegs. These key pegs will later be placed in the smaller holes on the board, as the feedback from the code-maker.

First of all, the code-maker chooses a combination of four colors from six possible colors. Duplicates are allowed, which means the combination may even be four pegs of the same color. The chosen combination is placed in the four holes covered by shield, which can be seen by the code-maker but not by the code-breaker. When the color combination has been chosen, the code-breaker can start guessing the pattern, in order and color, within given turns. Each guess is done by placing code pegs on the decoding board. When the guess has been made, the code-maker will then give feedback by placing zero to four key pegs in the provided smaller holes. A colored key peg indicates the existence of a code peg from the guess that is correct in position and color. A white key peg, on the other hand, is given for each color peg that is correct in color, but is placed in the wrong position.

Once the code-maker has given feedback, the code-breaker is allowed to make another guess. Guesses and feedbacks will continue to take turns until either the correct color combination is guessed, or the code-breaker has run out of turns.

2.2. Backtracking Algorithm

There are several algorithms that are widely used to find all or some solution to computational problems; two of which are brute force and backtracking. Both algorithms try to find solutions to a problem by gradually making solution candidates. However, backtracking is somewhat more efficient than brute force. How so? In finding a solution using brute force algorithm, a program will create every possible solution. For each solution, the program will then check whether it fulfills the specification of the requested final solution. Whereas the backtracking algorithm will stop processing a solution candidate once it is proven to not lead to the desired final solution. For instance, given we have an 8-letter word, and the program is requested to form a new word consisting of two vocals and two consonants. Once the program finds a third consonant during its search for solution, then the current candidate and any solution following it is automatically disposed, because it is not likely to fulfill the specification of desired solution.

Nevertheless, backtracking algorithm can only be applied to a certain type of problems. It can only works for problems with solutions that can be searched systematically and gradually. Backtracking can be applied only to problems that admit the concept of "partial candidate solutions" and can be tested relatively fast whether or not it may lead to a valid solution. There are problems that cannot be solved using backtracking, such as finding a value within an unordered table. Nonetheless, when it is applicable, backtracking can be a lot faster than brute force because of the large number of solution candidate it disposes in a single test.

Backtracking algorithm is based on the DFS (depth-first search). The mechanism of backtracking uses the principle of recursion. To solve the entire problem, a solution to the first sub-problem is required. Afterwards, this solution will be used to solve the following sub-problems, recursively. When the current candidate fails, or if it is required to find all of the possible solutions, then the program will backtrack to the preceding node and test the next possible solution. The backtracking process will stop when there are no more possible solutions.

One very significant characteristic of backtracking algorithm is its pruning function. Given the stages of solution finding is represented in the form of tree, pruning will be done to nodes that are not likely to form a valid solution. Once a node has been pruned, its children will automatically be left out of the process, because pruning a node is equal to disposing the whole path following said node.

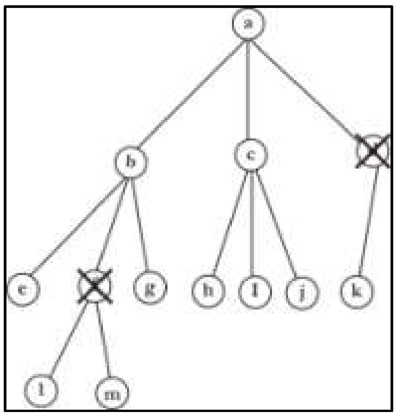

Figure 2 Illustration of Pruning

According to the figure above, nodes d and f are pruned. Consequently, node k, which is a child of node d, also node l and m which are children of node f, will not be processed.

Backtracking algorithm is widely used in the making of computer games such as tic-tac-toe, maze, and chess. It is usually used to build an artificial intelligence (A.I.) for the games. Aside from that, this algorithm is also the most efficient for parsing and many other combinatorial problems. Backtracking is also used in the logic programming language like Prolog, Planner, and Icon.

2.3. Mastermind Solving Strategy

In solving the mastermind game, before determining what algorithm to apply, we need to define the states of the game. Here we observe that, for every guess that is made, there are 14 different feedbacks possible. We will later refer to these feedbacks using these following numbers :

[0] No key pegs
[1] 1 white key peg
[2] 1 colored key peg
[3] 2 white key pegs
[4] 1 colored key peg, 1 white key peg
[5] 2 colored key pegs
[6] 3 white key pegs
[7] 1 colored key peg, 2 white key pegs
[8] 2 colored key pegs, 1 white key peg
[9] 3 colored key pegs
[10] 4 white key pegs
[11] 1 colored key peg, 3 white key pegs
[12] 2 colored key pegs, 2 white key pegs
[13] 4 colored key pegs (SOLVED)

Note that the feedback consisting of 3 colored key pegs and 1 white peg is not possible, because if there are already three pegs in the correct position, then a fourth peg with a correct color but placed in a wrong position is not likely to exist. Later on, we will refer to the six possible colors using the letter **A** to **F**. Now, to obtain a feedback that can give us most useful information to guess the correct color combination, what should be the first guess? In choosing four color pegs, there are 6x6x6x6 = 1296 possible color combinations. But, for any variation of colors and positions, there are only five essentially different combination: AAAA, AAAB, AABB, AABC, and ABCD. These are the five possible moves that can be chosen as the first move. Whenever a guess is made, the given feedback will reduce the number of possible solution. For example, if the code-breaker guessed 'AAAA' and the feedback given was 0 (no key pegs), that would mean that the color A is not in the combination, thus making the number of possible color combination reduced to only 5x5x5x5 = 625 color combinations. Analysis shows that the number of possible solutions left for each case is like so:

Table 1 Number of Solutions Left for Each First Guess x First Mark Case

First Mark	First Guess				
	AAAA	AAAB	AABB	AABC	ABCD
0	625	256	256	81	16
1	x	308	256	276	152
2	500	317	256	182	108
3	x	61	96	222	312
4	x	156	208	230	252
5	150	123	114	105	96
6	x	x	16	44	136
7	x	27	36	84	132
8	x	24	32	40	48
9	20	20	20	20	20
10	x	x	1	2	9
11	x	x	x	4	8
12	x	3	4	5	6
13	1	1	1	1	1

As we can conclude from the given table, the first guess that can reduce most number of solution is AABB, with a maximum solution left of 256. Therefore, strategically the first move to make is always AABB.

3. APPLICATION OF BACKTRACKING TO SOLVE MASTERMIND GAME

To create a solution to mastermind game using backtracking, we will later on use these conventions:

- The six possible colors will be stored in an array of colors [A..F]
- For every guesses made, feedbacks will be counted as scores. Each colored peg will be valued 2 while white pegs will be valued 1 each.

In order to be able to apply the backtracking algorithm to solve Mastermind, we need to first adjust the condition of the game. If we use an empty board as the start of the backtracking process, which means that the root of the solution tree is an empty board, the backtracking process might be a little too complex. Therefore, as a first step, we need to determine which color combination to use as the start of the backtracking process. This is done by finding the best valued first move, using the pattern AABB but with every variation of color. Color combinations that are marked with two white pegs (scored 2) will be regarded as its inverse (for example AABB will be regarded as BBAA), and considered marked with two colored pegs (scored 4).

Once the root of the tree has been decided, the tree is later expanded in two ways, by checking every possible positions (for example, AABB will be expanded to BABB, ABBB, AAAB, and AABA), then by trying every possible color in one same position (AABB will be expanded to BABB, CABB, DABB, EABB, and FABB). Combination that has been checked will not be checked twice. In expanding the tree, positions are changed from the left, while colors are changed based on its order in the array. Colors that are involved in guesses scored 0 will be discarded from the array. Color combinations that gives less score will automatically be pruned from the solution tree.

As a sample case, let us review the process of finding a solution if the code to find is **FCBE**. Based on the backtracking algorithm, the process of solving the game is as follows:

1. **Determine the root of the tree**
   In this case, the scores for the possible first moves are:

Table 2 Scores for Possible First Moves in Case FCBE

First Move	Mark	Score
AABB	one colored peg	2
CCDD	one colored peg	2
EEFF	two white pegs	2

From the table we can see that every first moves scores the same. But note that the last possible first move, that is EEFF, is marked with two white pegs. This means that we can regard the combination as FFEE, marked with two colored pegs, with a score of 4, making it the best valued first move.

2. **Expand the root by position**
   From the selected root, we can start expanding it by position. The first new node will be AFEE, which is marked with a white peg and a colored peg, scored 3. This is proven to be lower than the previous node, which means this node will be pruned. We move to the next new node, FAEE, which is scored 4, or equals to the current score. So we will continue to expand this node, with an additional condition that in the next stage, the score has to be higher than the current score. The result of this stage is illustrated in the following figure:

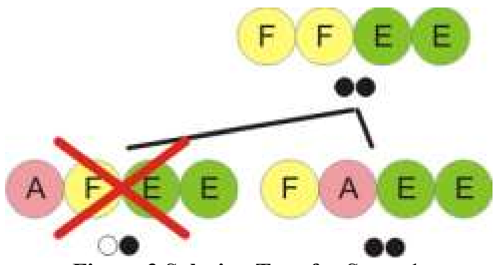

Figure 3 Solution Tree for Stage 1

3. **Expand the node by color**
   From the selected node, we can start expanding it by color. The first new node that is generated on this stage would be FBEE. This node is marked with two colored pegs scored 4, which does not fulfill the new condition given. So we prune this node and move on to the next node, which is FCEE. This node gets a total score of 6 from three colored pegs. This is greater than the current score, so this node will be expanded. The next node will be expanded by position, and the additional condition will not be applied. The result of this stage is illustrated in the following figure:

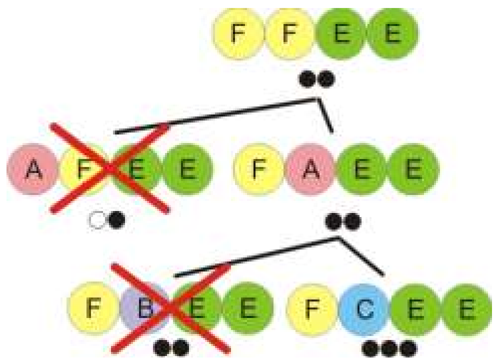

Figure 4 Solution Tree in Stage 2

4. **Expand the node by position**
   From the node FCEE, we check every position. The first node to be generated will be ACEE, which will not get a score higher than or equal to the previous score that has been obtained. The next node, FAEE, faces the same problem. It is at the next node, FCAE, that we find a node marked with three colored pegs, scored 6 or equal to the current score. This will be the node we take to expand on the next stage. The current solution tree is as follows:

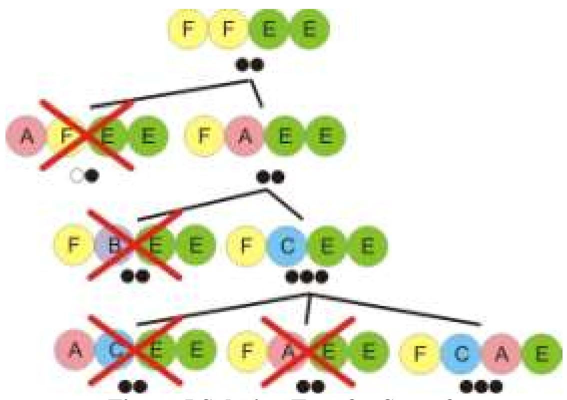

Figure 5 Solution Tree for Stage 3

5. **Expand the root by color**
   In this stage, the additional condition that the score has to be higher than the previous score is applied. The current node, FCAE, will be expanded in which the color in the third position will be changed with every color. In this stage, the solution is found in the first trial, that is FCBE. Because the solution is found, the searching will be stopped, and the nodes will not be expanded any further.

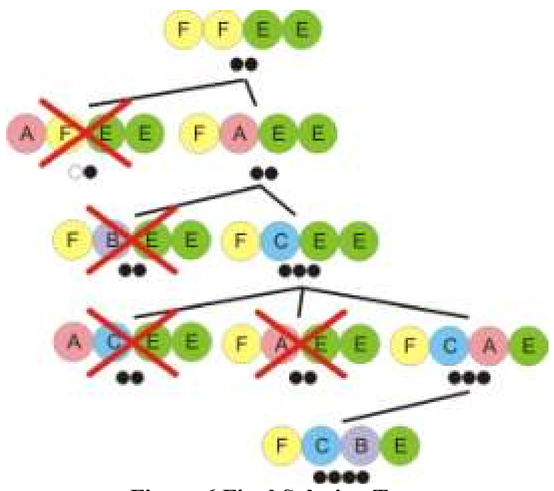

Figure 6 Final Solution Tree

4. CONCLUSION

The backtracking algorithm is commonly used to solve problems that admits the concept of partial candidate solutions. It is highly efficient when applied in the right type of problem. With a few adjustments, this algorithm can solve the mastermind game. However, the solution using backtracking is not the best solution for this game. There are still many cases in which the algorithm fails to finish the game within given turn. The best solution so far is the five-guess algorithm created by Donald Knuth, and there is also a method by Kenji Koyama and Tony W. Lai which can finish the game within an average of 4.34 turns, with a worst case of six turns.

REFERENCES

[1] Munir, Rinaldi, “Strategi Algoritmik”, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2007, 125-149.

[2] http://en.wikipedia.org/wiki/Mastermind_(board_game) (accessed December 29th, 2009, 10:40 PM)

[3] http://www.tnelson.demon.co.uk/mastermind/ (accessed December 31st, 2009, 6:34 PM)

[4] http://en.wikipedia.org/wiki/Backtracking (accessed January 2nd, 2010, 1:24 AM)

[5] http://programmingpraxis.com/2009/11/20/master-mind-part-2/ (accessed January 2nd, 2010, 2:31 AM)