Chess Puzzle Mate in N-Moves Solver with Branch and Bound Algorithm

Ryan Ignatius Hadiwijaya / 13511070
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13511070@std.stei.itb.ac.id

***Abstract*—Chess is popular game played by many people in the world. Aside from the normal chess game, there is a special board configuration that made chess puzzle. Chess puzzle has special objective, and one of that objective is to checkmate the opponent within specifically number of moves. To solve that kind of chess puzzle, branch and bound algorithm can be used for choosing moves to checkmate the opponent. With good heuristics, the algorithm will solve chess puzzle effectively and efficiently.**

***Index Terms*—branch and bound, chess, problem solving, puzzle.**

I. Introduction

Chess is a board game played between two players on opposite sides of board containing 64 squares of alternating colors. Each player plays with different color of pieces; white and black pieces. But, each player has the same 16 pieces from the start : 1 king, 1 queen, 2 rooks, 2 bishops, 2 knights, and 8 pawns.

Figure 1 - starting chess configuration

Chess always start with this pieces configuraton. The player with the white pieces always moves first. Then, each player make 1 move alternately until the end of the game. Chess game will end if one of the following case occur :

1. Any player checkmate the opponent. If this occur, then that player will win and the opponent will lose.
2. Any player resign / give up. If this occur, then that player is lose.
3. Draw. Draw occur on any of the following case :
   a. Stalemate. Stalemate occur when player doesn't have any legal moves in his/her turn and his/her king isn't in checked.
   b. Both players agree to draw.
   c. There are not enough pieces on the board to force a checkmate.
   d. Exact same position is repeated 3 times
   e. Fifty consecutive moves with neither player has moved a pawn or captured a piece.

One of the chess variant is chess puzzle, which is a chess board configuration set by composer (and often present very artificial looking positions) present to (human) solvers with particular task to be achieved. One of that task is to checkmate the opponent in a pre-determined number of moves. This paper will focus on solving chess puzzle with checkmate in pre-determined number of moves.

II. Basic Theory

A. Branch and Bound Algorithm

Branch and bound algorithm is based on Breadth First Search (BFS) algorithm with least cost search. Each node has a value to represent cost of that node. Cost of the each node will determine the next node to expand. Branch and bound algorithm works as follow :

1. Put the root node on the queue. If the root contains the solution, then the solution is found, stop.
2. If queue is empty, then there is no solution, stop.
3. If queue isn't empty, choose node i from the queue which have least cost.
4. If node i contains the solution, then solution is found, stop. If node i is not contain solution, then expand that node and all its children. If node i doesn't have a child, back to step 2.

5. For each child j from node i, calculate cost of the node j, and put all the children on the queue.
6. Back to step 2.

B. Moves

To solve chess problem, first we must know how each piece moves. Each of the 6 different kinds of pieces moves differently. Pieces cannot move through other pieces (though the knight can jump over other pieces), and can never move onto a square with one of their own pieces. However, pieces can be moved to take the place of an opponent's piece which is then captured. Pieces are generally moved into positions where they can capture other pieces (by landing on their square and then replacing them), defend their own pieces in case of capture, or control important squares in the game.

Each pieces move describe as below :

1. King

The king is the most important piece, but is one of the weakest. The king can only move one square in any direction - up, down, left, right, or diagonally. The king may never move himself into check (where he could be captured).

2. Queen

The queen is the most powerful piece. She can move in any one straight direction - forward, backward, sideways, or diagonally - as far as possible as long as she does not move through any of her own pieces. And, like with all pieces, if the queen captures an opponent's piece her move is over.

3. Rook

The rook may move as far as it wants, but only forward, backward, and to the sides.

4. Bishop

The bishop may move as far as it wants, but only diagonally. Each bishop starts on one color (light or dark) and must always stay on that color.

5. Knight

Knights move in a very different way from the other pieces – going two squares in one direction, and then one more move at a 90 degree angle, just like the shape of an "L". Knights are also the only pieces that can move over other pieces.

6. Pawn

Pawns are unusual because they move and capture in different ways: they move forward, but capture diagonally. Pawns can only move forward one square at a time, except for their very first move where they can move forward two squares. Pawns can only capture one square diagonally in front of them. They can never move or capture backwards. If there is another piece directly in front of a pawn he cannot move past or capture that piece.

C. Chess Notation

Chess has a notation to describe each of the moves. Each of the piece has a prefix different from each other pieces. The notation of each piece :

King – K
Queen – Q
Rook – R
Knight – N
Bishop – B

Pawn doesn't have prefix for its notation. This prefix is then followed by the destination square with format column ['A'-'H'] and row ['1'-'8'].

Example of the chess notation is :

1. e4 e5
2. Nf3 Nc6
3. Bc4 Be7
4. 0 – 0

D. Calculating the best moves

There is an easy system that most players use to keep track of the relative value of each chess piece:

- A pawn is worth 1
- A knight is worth 3
- A bishop is worth 3
- A rook is worth 5
- A queen is worth 9
- A king is infinitely valuable

The normal chess game usually consider using this value very much. The player can use this to make decisions while playing, helping know when to capture, exchange, or make other moves. So, the player will not make a disadvantageous move.

But, this solver needs to produce moves to checkmate the opponent. So, that value doesn't mean anything much. And different approach must be made to quickly solve this problem.

To solve this problem, basically we just need to try all the possible moves until we find the move to checkmate the opponent. But, this approach is too slow and needs large memory. So, heuristic is needed for efficiency. To make the good heuristic, we must consider the checkmate condition of the opponent. Checkmate happens when the opponent king is put into check and cannot get out of check in his/her turn. There are only three ways a king can get out of check: move out of the, block the check with another piece, or capture the piece threatening the king.

The first heuristic is to consider move which check the opponent first. This is because the opponent king must be put in check to checkmate him/her. Also, from the chess puzzle example in [], we can see that most of the solutions check the opponent repeatedly until checkmate.

The second heuristic is to consider move which reduce number of legal moves of the opponent king. The opponent king mustn't have any legal moves to checkmate him/her. Fewer of legal moves of the opponent king has mean it is more likely to reduce it to zero on the next move.

The third heuristic is to consider move with piece order : queen, rook, bishop, knight, pawn, king. This order is made by strength of the pieces. It is more likely to checkmate with queen rather than with pawn (although in some problem, the solution needs pawn to checkmate the opponent).

The fourth heuristic is to never move to the same board configuration.

III. IMPLEMENTATION

A. Chess Data Structure

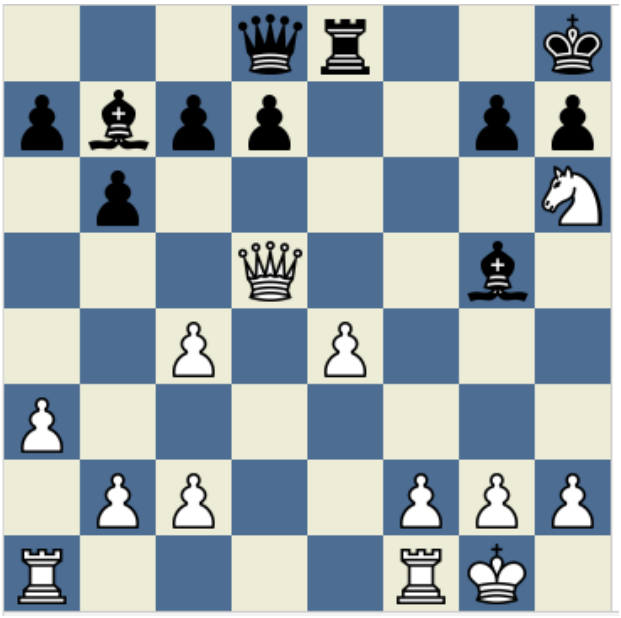

Figure 2 – example of chess puzzle mate in two

Before solving the problem, we need to make data structures to represent chess board. There are some implementation of the data structure to represent chess board, in example :

1. 2D dimensional array or matrix of size 8 x 8 which represent each square in chess. This matrix store information about which piece is standing on that square. Information stored as integer which has meaning :

Matrix value	Information
0	Empty
1	White King
2	White Queen
3	White Rook
4	White Knight
5	White Bishop
6	White Pawn
7	Black King
8	Black Queen
9	Black Rook
10	Black Knight
11	Black Bishop
12	Black Pawn

Table1 – piece representation in number

So, board in figure [2] will be represented as 8x8 array which has value :

0	0	0	8	9	0	0	7
12	11	12	12	0	0	12	12
0	12	0	0	0	0	0	4
0	0	0	2	0	0	11	0
0	0	6	0	6	0	0	0
6	0	0	0	0	0	0	0
0	6	6	0	0	6	6	6
3	0	0	0	0	3	1	0

Table 2 – board representation with 8x8 array

2. Piece List : 1D dimensional array of size 32 which represent each piece position. The position can be represent with two number digit, first represent the column and second the row. Each index represent piece in the table :

Index	Piece
0	White King
1	White Queen
2	White Rook A
3	White Rook H
4	White Knight B
5	White Knight G
6	White Bishop C
7	White Bishop F
8	White Pawn A
9	White Pawn B
10	White Pawn C
11	White Pawn D
12	White Pawn E
13	White Pawn F
14	White Pawn G
15	White Pawn H

Index	Piece
16	Black King
17	Black Queen
18	Black Rook A
19	Black Rook H
20	Black Knight B
21	Black Knight G
22	Black Bishop C
23	Black Bishop F
24	Black Pawn A
25	Black Pawn B
26	Black Pawn C
27	Black Pawn D
28	Black Pawn E
29	Black Pawn F
30	Black Pawn G
31	Black Pawn H

Table 3 – piece list in number

Index	Position
16	88
17	48
18	58
19	00
20	00
21	00
22	27
23	75
24	17
25	26
26	37
27	47
28	00
29	00
30	77
31	87

Table 4 – board representation with piece list

So, board in figure [2] will represented as :

Index	Position
0	71
1	45
2	11
3	61
4	86
5	00
6	00
7	00
8	13
9	22
10	32
11	34
12	54
13	62
14	72
15	82

B. Branch and Bound Algorithm Implementation

To solve the problem using branch and bound algorithm, program use heuristics as define in the theory above. The implementation of choosing moves store on the priority queue. Each move has a value which calculated by each heuristic. Each move start with value = 0. From heuristic 1, add 50 value if the move check the opponent and 100 value if the move double check the opponent. From heuristic 2, add value of (100 – 2 * (number of opponent king moves)^2). From heuristic 3, if two or more moves have the same values, then priority queue is sorted by the value of the piece. From heuristic 4, each move is recorded at map tree so the same moves never considered again.

From the diagram [2] above, using above implementation, the first move to be considered is Qg8+. That move check the opponent so the value get +50. That move also makes the opponent king number of move reduce to the zero. So, that value get +100 and the final value of that move is 150. The second move to be considered is Ng7+ which check the opponent and reduce number of opponent king move to 1 and get final value of 148.

From the first move, the solution then can be reached by the second move. The example of implementation in figure [2] is :

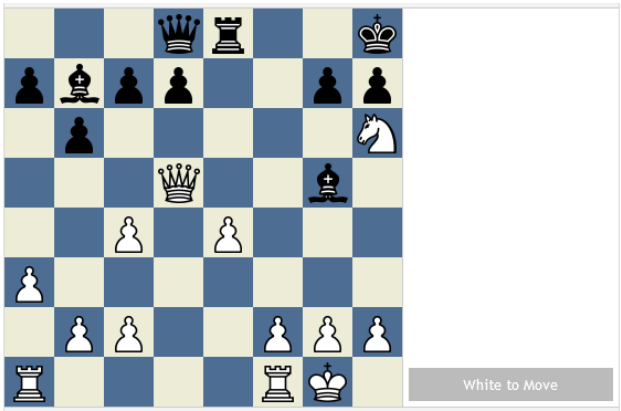

Figure 3 – Initial chess problem

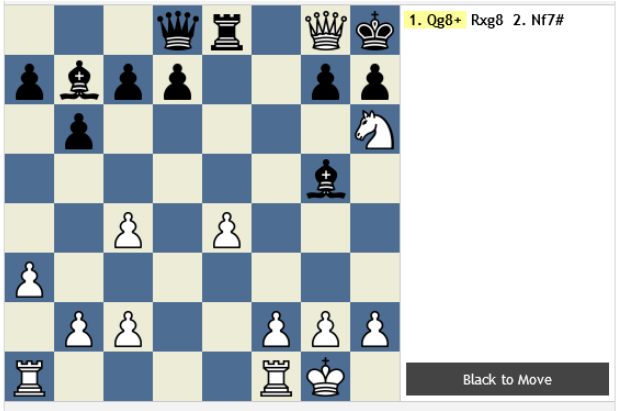

Figure 4 – Move 1 white

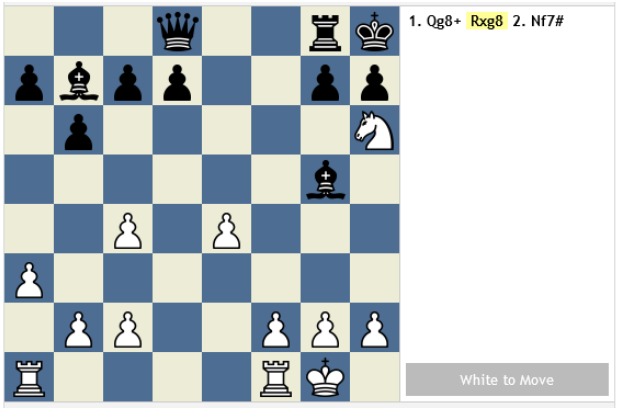

Figure 5 – Move 1 black

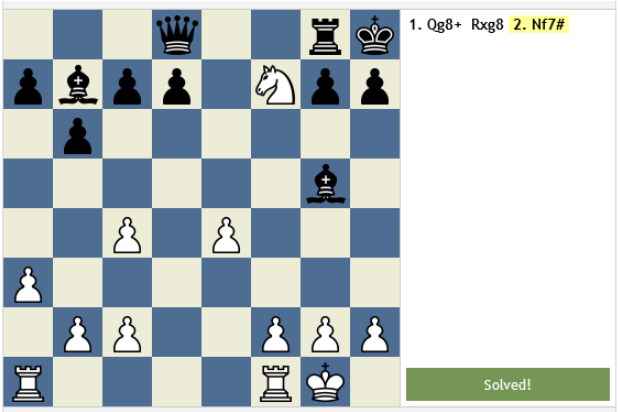

Figure 6 – Move 2 white

IV. Analysis

A. Data Structure

With 8x8 array representation, we can know the condition of each square. If we want to know what piece is on that square, we can use the array with O(1) complexity time to search. But, it has to iterate all the square to find a piece on the board.

With piece list representation, we can know each of the piece location with O(1) search time. But, we need to iterate all the piece to know which piece is standing on the square.

This paper use two of the data structures as define above. Though it is quite waste of memory size, but using that two data structures at once can faster the search of the move. This is because we use each of the data structures advantage to cover the other disadvantage. We can use piece list to quickly determine the position of the piece we want to consider, and use 8x8 array to quickly determine the piece on the board to determine that piece movement and attacking square.

The heuristic this paper use is good enough for most of the puzzle, including all the puzzles in [3]. But, in some of the puzzle, the first heuristic can go wrong. Because, the solution may be not to check the enemy in the beginning.

B. Branch and Bound Algorithm

With chess puzzle example in figure [2], we can solve that using normal BFS or Branch and Bound. With normal BFS, there are 43 moves created on the first move. Each of the moves generate many other moves leading to many moves that are far from checkmating the opponent. But, with Branch and Bound Algorithm using implementation above, we just need to expand two nodes to reach the solution. So, branch and bound algorithm reduce the size of the search tree and faster the search dramatically because the search only expand move that are more likely to be the solution first.

V. Conclusion

Chess puzzle mate in n-moves can be solved effectively end efficiently using branch and bound algorithm. The algorithm performance is depend on the heuristic used. Using good heuristic will make the huge difference in search time.

References

[1] Munir, Rinaldi. 2009. *Diktat Kuliah Strategi Algoritma*. Program Studi Teknik Informatika STEI ITB.
[2] http://www.chess.com
[3] http://www.chess.com/forum/view/more-puzzles/300-checkmate-puzzles-puzzles-1---50
[4] http://chesspuzzles.com
[5] http://chessprogramming.wikispaces.com

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Desember 2013

Ryan Ignatius / 13511070