Optimization of Tic-Tac-Toe Game Status Analysis Using Boolean Algebra Approach

Muhammad Hazim Ramadhan Prajoda - 13523009
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
hazimmuhammad28@gmail.com, 13523009@std.stei.itb.ac.id

***Abstract*—Boolean algebra approach uses a representation of game states in logical equations; in optimizing the tic-tac-toe game status, it transforms the board into a bitboard that represents as bits. This conversion allows it to be more efficient in performing bitwise operations. It allows to minimize computational complexity while ensuring accuracy.**

***Keywords*—Boolean algebra, optimization, tic-tac-toe, game status**

I. Introduction

Tic-tac-toe, a widely known game, dates to ancient civilizations. Its historical roots are far deeper than any lesson. Frankly, the origins of the term "tic-tac-toe" have never been fully determined. Some theories propose that it comes from the sound of a pencil sketching on paper, while others suggest that "tic-tac-toe" represents the trio of moves essential to win [1]. There are other names for tic-tac-toe, such as "noughts and crosses" in different parts of the world, and even an evidence of similar games dates to ancient Egypt, where it was known as "Seega" in that era. The games used pebbles and stones as game pieces, it can be said that the Ancient Egyptians can be credited with the laying foundation for the origins of tic-tac-toe.

Tic-tac-toe presents interesting computational challenges in game state analysis. The traditional approach to analyzing game states involves checking rows, columns, and diagonals through arithmetic operations. Although, this method may not be optimal for all scenarios. With the advancement of computer science, alternative approaches such as Boolean algebra have emerged as potential optimization techniques for game state analysis.

This research focuses on optimizing the game state analysis of tic-tac-toe using Boolean algebra principles. By transforming the traditional board representation into a bitboard format, we can leverage efficient bitwise operations for pattern matching and winner detection. The research compares the performance of traditional arithmetic-based methods with Boolean algebra approaches across various game scenarios, from simple row wins to complex diagonal patterns.

The value of this research goes beyond just improving tic-tac-toe itself. It also can be made to applied to more complex game analysis systems and pattern recognition problem.

II. Theoretical Basis

A. Tic-Tac-Toe Game

The game of tic-tac-toe has a set of rules that must be followed by every player. Before starting on those rules, we must first understand how to set up the tic-tac-toe board. At its heart, tic-tac-toe is made up of a 3x3 grid, as shown below:

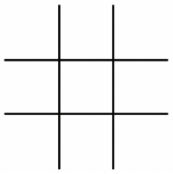
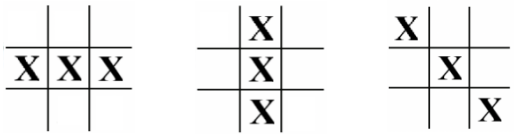


Figure 2.1. The Board of Tic-tac-toe

As shown in **Fig. 2.1.**, there are nine blank spots to be filled by the two players alternately. Each player will use a unique symbol, typically an X or an O.

The goal of this game is to get the three of your symbols in a row which can be horizontal, vertical, or even diagonal.



(a) (b) (c)

Figure 2.2. Ways to Win in Tic-tac-toe: (a) Horizontal, (b) vertical, (c) Diagonal

The opposing player must prevent the other player from getting the winning row as shown in **Fig. 2.2.** The players can't override the spots that are already filled in the board.

B. Boolean Algebra

1. Definition

Boolean algebra is a branch of algebra that only have two possible outcomes, 1 or 0. In another way of saying, the variables can only have two options, true or false. Boolean Algebra was found by George Boole where he saw that set and propositional logic have similar properties.

The definition of Boolean algebra is as follows: Say that B is a set that defined by 2 binary operators, (+) & (·), and single unary operator (’). Say that 0 and 1 are different elements from set B.

So, the tuple <B, +, ·, ’, 0, 1> are called Boolean algebra if for every a, b, c ∈ B the following axiom holds:

1. Identity
   (i) a + 0 = a
   (ii) a · 1 = a
2. Commutative
   (i) a + b = b + a
   (ii) a · b = b · a
3. Distributive
   (i) a · (b + c) = (a · b) + (a · c)
   (ii) a + (b · c) = (a + b) · (a + c)
4. Complement

For every a ∈ B, there's a unique element a' ∈ B thus:
   (i) a + a' = 1
   (ii) a · a' = 0

From the axioms above, it can be concluded that both set algebra and propositional logic algebra are instances of Boolean algebra or a subset of Boolean algebra as they fulfilled all axioms mentioned. From that, we can analogously say that the (+) operation in Boolean algebra is analogous to the 'or' operation in propositional algebra, while the (·) operation is the 'and' equivalent. The same instances also goes to the other operator and elements such as ('), 0, and 1, that are equivalent to (~), F, and T. It can be concluded that <B, ∨, ∧, ~, F, T> are Boolean algebra.

2. Two Valued Boolean Algebra

Two valued Boolean algebra is defined on a set of two elements, B = {0, 1} with the binary operator (+) & (·), and an unary operator ('). The two valued Boolean algebra also fulfils the 4 axioms above, the rules for binary and unary operators is as follows:

Table 2.1. Binary and Unary Operator in Two Valued Boolean Algebra (source from [3])

a	b	a · b
0	0	0
0	1	0
1	0	0
1	1	1

a	b	a + b
0	0	0
0	1	1
1	0	1
1	1	1

a	a'
0	1
1	1

3. Laws Of Boolean Algebra

In Boolean Algebra, a set of laws have been invented to help reduce the number of logic gates needed to perform a particular logic operation[5], The laws of Boolean algebra is as follows:

Table 2.2. The Laws of Boolean Algebra

1. Identity Laws: (i) a + 0 = a (ii) a · 1 = a	2. Idempotent Laws: (i) a + a = a (ii) a · a = a
3. Complement Laws: (i) a + a' = 1 (ii) a · a' = 0	4. Dominance Laws: (i) a · 0 = 0 (ii) a + 1 = 1
5. Involution Law: (i) (a')' = a	6. Absorption Laws: (ii) a + ab = a (iii) a(a + b) = a
7. Commutative Laws: (iv) a + b = b + a (v) ab = ba	8. Associative Laws: (i) a + (b + c) = (a + b) + c (ii) a · a = a
9. Distributive Laws: (i) a + (bc) = (a + b)(a + c) (ii) a(b + c) = ab + bc	10. De Morgan Laws: (i) (a + b)' = a'b' (ii) (ab)' = a' + b'
11. 0/1 Laws: (i) 0' = 1 (ii) 1' = 0	-

4. Boolean Functions

Boolean functions are functions that involve variables (also known as literals) that take on binary values (0 or 1, or TRUE/FALSE). These functions are fundamental to digital logic and computer design.

Here's an example of Boolean functions:

$$f(x) = x$$

This function only has one literal, that is x, that means the function depends on the single value of the variable x.

$$f(x) = x'yz$$

This function has three literals: x', y, and z. This function will be only 1 (True) if x=0, y=1, and z=1 at the same time.

$$f(x, y) = x + y'$$

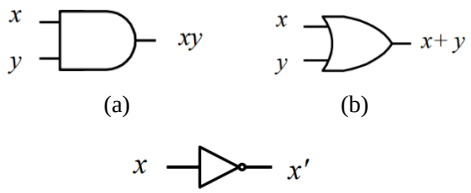
This function has two literals: x and y'. This function will be 1 (True) if either x=1, or y=0.

5. Logic Gate

One of the way to represents Boolean functions is using logic gate. Logic gates are electronic implementations of basic Boolean functions. Each gate processes one or more binary inputs to produce a single binary output based on its specific Boolean operation.

There are there basic logic gates: AND gates, OR gates, and NOT gates.

AND gates are the equivalent to the operator (·) in Boolean algebra, OR gates are (+), and (') for NOT gates.



(a) (b)

(c)

Figure 2.3. Basic Gates: (a) AND Gate, (b) OR Gate, and (c) NOT Gate (source from [3])

While the other gates are derived from the basic gates, there are universal gates: the NAND gate (a combination of AND followed by NOT) and the NOR gate (a combination of OR followed by NOT). There is also a special gate: the XOR gate (a combination of AND, OR, and NOT).

(a) (b)

Figure 2.4. Universal Gates: (a) NAND Gate, (b) NOR Gate (source from [3])

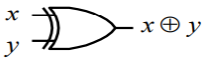


Figure 2.5. XOR Gate (source from [3])

C. Game State Analysis

Game state analysis in Tic-Tac-Toe involves evaluating the current board configuration to determine if there's a winner or if the game continues. This analysis is crucial for both gameplay and artificial intelligence implementations.

The game of Tic-Tac-Toe has a relatively small state space compared to other board games. With 9 cells and 3 possible states for each cell (empty, X, O), the game presents a finite but significant number of possible configurations. The first player has 9 possible moves, followed by 8 possible moves for the second player, and so on. This leads to:

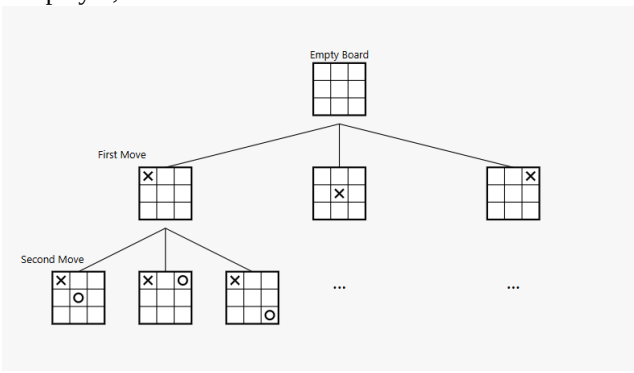


Figure 2.6. State Space Tree of Tic-Tac-Toe

Traditional pattern recognition in Tic-Tac-Toe focuses on checking 8 possible winning combinations: three horizontal lines, three vertical lines, and two diagonal lines, as shown previously in **Figure 2.6**. Each line requires checking three positions to determine if a player has achieved victory. These winning patterns can be represented both in traditional array format and in binary format for Boolean operations.

Several key factors affect the performance of state analysis implementations. These include the number of operations required per check, memory access patterns, cache efficiency, possibilities for early termination, and potential for parallelization. The choice of implementation method can significantly impact these factors, leading to varying levels of performance in different scenarios. This theoretical foundation helps explain why certain approaches may perform better than others under specific circumstances, providing the basis for optimizing game state analysis through methods such as Boolean algebra.

III. IMPLEMENTATION

A. Traditional Method Implementation

The traditional method only utilizes two simple implementations. An array-based approach for representing the game board of tic-tac-toe and an arithmetic operations for detecting the winner. The value 0 is used for representing empty cell, while the values of 1 and -1 are used for representing the X and O symbol respectively.

This implementation consists of checking three main patterns found in tic-tac-toe game. Row checking, column checking, and diagonal checking. Every checking will determine if the total equals of row, column, or diagonal is equal to 3 (X wins) or -3 (O wins).

The row checking process examines each horizontal line by summing the values. When all three cells in a row contain X (value 1), their sum equals 3, indicating X wins. Conversely, three O symbols (value -1) sum to -3, indicating O wins. The column checking follows the same principle but examines vertical alignments instead. For diagonal checking, the method sums both the main diagonal (top-left to bottom-right) and anti-diagonal (top-right to bottom-left), applying the same win condition logic of checking for sums of 3 or -3.

For a clearer understanding, here's the code for the traditional method:

```python
class TicTacToeTraditional:
    def __init__(self):
        self.board = np.zeros((3, 3), dtype=int)

    def check_winner_traditional(self):
        # Check rows and columns
        for i in range(3):
            row_sum = sum(self.board[i])
            col_sum = sum(self.board[:,i])
            if row_sum == 3 or col_sum == 3:
                return 1
            if row_sum == -3 or col_sum == -3:
                return -1

        # Check diagonals
        diag1 = self.board[0][0] + self.board[1][1] + self.board[2][2]
        diag2 = self.board[0][2] + self.board[1][1] + self.board[2][0]
        if diag1 == 3 or diag2 == 3:
            return 1
        if diag1 == -3 or diag2 == -3:
            return -1
        return 0
```

Figure 3.1. Traditional Method Implementation

B. Boolean Method Implementation

Different than the traditional method, the Boolean algebra method transforms the game board into a bitboard representation where cell's state is represented as bits. It converts the traditional board into two separate bitboards for X and O as in the following implementation:

```python
    def board_to_bitboard(self, board):
        # Optimized bitboard conversion using bit shifting
        x_board = 0
        o_board = 0
        # Process all cells in one pass
        for i in range(9):
            row, col = divmod(i, 3)
            if board[row][col] == 1:
                x_board |= 1 << i
            elif board[row][col] == -1:
                o_board |= 1 << i
        return x_board, o_board
```

Figure 3.2. Converting Board Game into Bitboard

This makes it so that every cell on the board will be

represented as a bit. For example, a binary pattern of 111000000 represents three marks in the first row. This conversion allows the program to use efficient bitwise operations for pattern matching. Each player (X and O) gets their own bitboard, which tracks their positions on the board using 1s for occupied cells and 0s for unoccupied cells.

After converting the board game into bitboard in **Fig. 3.2.** It'll do a pattern matching using pre-defined bit patterns for win conditions.

```
def __init__(self):
        # Optimized pattern organization for faster matching
        self.row_patterns = [
            0b111000000,  # Row 1
            0b000111000,  # Row 2
            0b000000111   # Row 3
        ]
        self.col_patterns = [
            0b100100100,  # Col 1
            0b010010010,  # Col 2
            0b001001001   # Col 3
        ]
        self.diag_patterns = [
            0b100010001,  # Diag 1
            0b001010100   # Diag 2
        ]
```

Figure 3.3. Pre-Defined Pattern for Boolean Algebra Method

The pre-defined patterns represent all possible winning combinations in binary format. Row patterns use consecutive 1s (111) in appropriate positions, column patterns use spaced 1s (100100100), and diagonal patterns arrange 1s according to their respective diagonal positions. This binary representation enables efficient win detection through bitwise operations.

Using the pre-defined pattern in **Fig 3.3.** The program will then utilize the AND operations to check for the winner as in the following:

```
def check_winner_boolean(self, board):
        x_board, o_board = self.board_to_bitboard(board)

        # Check each pattern category separately
        # First check rows (most common win condition)
        for pattern in self.row_patterns:
            if (x_board & pattern) == pattern:
                return 1
            if (o_board & pattern) == pattern:
                return -1

        # Then check columns
        for pattern in self.col_patterns:
            if (x_board & pattern) == pattern:
                return 1
            if (o_board & pattern) == pattern:
                return -1

        # Finally check diagonals
        for pattern in self.diag_patterns:
            if (x_board & pattern) == pattern:
                return 1
            if (o_board & pattern) == pattern:
                return -1

        return 0
```

Figure 3.4. Checking Wins Condition with AND Operation in Boolean Algebra Method

C. Testing Scenarios

1. Test Cases

In this test, there are 5 game scenarios that will be tested. Each test differs from the other to ensure that the method comparison between the traditional approach and Boolean algebra is satisfied. These scenarios range from simple win conditions to more complex board states.

All test case scenarios will be shown in the following:

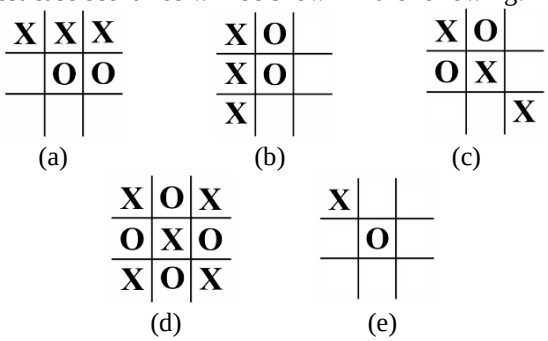


Figure 3.5. Test Case Scenario: (a) Row win patterns, (b) Column win patterns, (c) Diagonal win patterns, (d) Draw game scenarios, (e) Early game states.

For a more clearer understanding for every test case in **Fig. 3.5.** The first test case (a) demonstrates a row win pattern where X completes a horizontal line, representing the most basic winning condition. The second case (b) shows a column win scenario with X achieving victory through vertical alignment. A more complex diagonal win pattern is shown in case (c), featuring strategically placed X and O moves. Case (d) represents a full board draw situation where neither player achieves victory, while case (e) simulates an early game state with minimal moves made.

2. Performance Metrics

The performance metrics that are used in this test include execution time (in seconds); it measures how long each method takes to analyze the game state and determine a winner, memory usage (in MB); it tracks the computational resources required by each approach, standard deviation of measurements; this helps understand the consistency of performance across multiple runs, and percentage improvement calculation; it provides a clear comparison between the two methods. All of this is to ensure that the evaluation of the methods is done comprehensively.

3. Testing Parameters

Testing Parameters To ensure statistical validity, each test case was run for 100,000 iterations to ensures that any minor system fluctuations are averaged out, with 5 trials conducted per case will establish consistency in the results. Various board configurations were used to test different win patterns, ensuring the robustness of the comparison. This testing method reduces the influence of system fluctuations and ensures accurate performance metrics.

IV. ANALYSIS

A. Testing Results

Result of the testing reveals consistent patterns across multiple runs, with slight variations of ±3% in performance metrics.

Here are the following results:

Comprehensive Test Results:

Case	Traditional Time (s)	Traditional Std	Boolean Time (s)	Boolean Std
Row Win Simple	0.134855	0.00711506	0.343146	0.00972725
Column Win Simple	0.142307	0.00542247	0.355041	0.00347737
Diagonal Win Complex	0.504028	0.0133226	0.368633	0.00639232
Full Board Draw	0.507564	0.00919174	0.341384	0.00967547
Early Game State	0.513491	0.010506	0.392646	0.00762509

Figure 4.1. Time Taken for Every Test Case

Speed Improvement (%)	Traditional Memory (KB)	Boolean Memory (KB)
-154.456	32	26.4
-149.488	33.6	28.8
26.8626	38.4	43.2
32.7408	44.8	28.8
23.534	37.6	35.2

Figure 4.2. Speed Improvement and Memory Taken from Every Test Case

From **Fig. 4.1**. and **Fig 4.2.** the overall performance for the simple winning patterns such as row and column wins are consistently won by the traditional method. The traditional method completes row win checks in approximately 0.13 seconds, while the Boolean method requires about 0.32 seconds, showing the traditional method is roughly 143-156% faster for these simple patterns. Similar performance is observed in column win scenarios, where the traditional method maintains its significant speed advantage.

Though the Boolean algebra method shows its strengths when dealing with more complex patterns. In diagonal win scenarios, the Boolean approach demonstrates a 26-29% improvement in execution time compared to the traditional method. This advantage extends to full board analysis situations, where the Boolean method performs 32-33% faster. Even in early game states with scattered pieces, the Boolean approach maintains a 25-27% performance improvement over the traditional method.

B. Pattern-Specific Analysis

The two methods performance vary significantly based on the complexity of the patterns being analyzed. Simple patterns like rows and columns are faster with traditional method cause its straightforward arithmetic operations and direct array access. Its ability to perform summations without any need for data conversion makes it particularly efficient for these basic patterns.

On the other hand, the Boolean method had a more advantages in handling complex patterns. Its bitwise operations prove especially efficient when analyzing diagonal patterns and full board states. The parallel processing capability of bit operations becomes particularly beneficial when multiple pattern checks are required, as seen in the full board analysis and early game states where various winning possibilities need to be evaluated simultaneously.

C. Memory Analysis

Memory usage analysis reveals interesting patterns across different test scenarios. Particularly in simple pattern scenario where Boolean method demonstrates efficient memory utilization of 26.4-28.8 KB, while the traditional method requires slightly more at 32-33.6 KB . This advantage in memory efficiency complements the Boolean method's superior performance in handling complex patterns.

As pattern complexity increases, memory usage shows an interesting trend. The traditional method's memory consumption increases progressively to 44.8 KB for full board analysis. The Boolean method, despite its optimization for complex patterns, maintains relatively stable memory usage between 20.8-43.2 KB across different scenarios. In early game states, both methods show comparable memory efficiency (traditional: 37.6 KB, Boolean: 35.2 KB), indicating that initial overhead costs are similar for both approaches.

These memory measurements demonstrate that while the Boolean method may be slower for simple patterns, it generally maintains better memory efficiency, particularly in complex game states where its performance advantages are most pronounced.

D. Additional Considerations

There are several factors that influence the performance variations observed in this study. The impact at the system level results in an execution time variation of about ±3% among different processes, which demonstrates the robustness of both methods across a wide range of system conditions. Yet the relative performance patterns remain consistent for both methods.

V. Conclusion

In conclusion, the Boolean algebra method is much more efficient for analyzing game status in tic-tac-toe. This method shows significant optimization in more complex game states. The Boolean algebra method is more useful when thorough analysis is required.

Additionally, the Boolean approach demonstrates superior memory efficiency, especially as pattern complexity increases. While the traditional method's memory usage grows progressively, the Boolean method maintains stable and efficient memory consumption across various scenarios.

The Boolean algebra method offers a balanced combination of speed and memory efficiency in complex states, highlighting its potential for broader applications in game analysis and computational problem-solving.

VI. Acknowledgment

First and foremost, praises and thanks to Allah SWT, The Almighty God.

The author would like to express gratitude to God Almighty, as it is by His grace and love that the author was able to complete the discrete mathematics paper assignment on time.

The author also extends heartfelt thanks to his parents, who have always supported him both physically and emotionally, and provided unwavering motivation throughout his studies.

Furthermore, the author wishes to convey sincere appreciation to the lecturers of the discrete mathematics course, especially Mr. Dr. Ir. Rinaldi Munir, M.T., for his invaluable guidance and teaching throughout one semester of the discrete mathematics class.

Lastly, the author would like to thank his friends, who have been true companions in his academic journey as a student of Informatics Engineering, ITB 2023.

REFERENCES

[1] TicTacToeFree Team (2023) History and origins of tic-tac-toe, TicTacToe. Available at: https://tictactoefree.com/tips/tic-tac-toe-history-and-origins [Accessed: 05 January 2025].

[2] Tictactoefree (2024) Tic-Tac-Toe Rules. Learn How to Play the Game, YouTube. Available at: https://www.youtube.com/watch?v=1jee4qFYdP8&t=3s [Accessed: 05 January 2025].

[3] R. Munir, "Aljabar Boolean (Bagian 1)," Informatika.stei.itb.ac.id, 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/12-Aljabar-Boolean-(2024)-bagian1.pdf [Accessed: Jan. 5, 2025].

[4] R. Munir, "Aljabar Boolean (Bagian 2), " Informatika.stei.itb.ac.id, 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/13-Aljabar-Boolean-(2024)-bagian2.pdf [Accessed: Jan. 5, 2025].

[5] J. -F. Weng, S. -S. Tseng and T. -J. Lee, "Teaching Boolean Logic through Game Rule Tuning," in *IEEE Transactions on Learning Technologies*, vol. 3, no. 4, pp. 319-328, Oct.-Dec. 2010, doi: 10.1109/TLT.2010.33.

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, 7 January 2025

Nama dan NIM
Muhammad Hazim Ramadhan Prajoda / 13523009