A Dynamic Programming Approach to Chess Endgame Tablebase Construction

Nathaniel Christian - 13524122
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: nathaniel.christian@gmail.com, 13524122@std.stei.itb.ac.id

***Abstract*—Chess endgames have been extensively studied by thousands of chess experts over many years since the modern rules of chess were established. One of the most important developments for modern competitive chess scene, especially in end game analysis, is the creation of endgame tablebases. These tablebases are commonly constructed using retrograde analysis. This paper brings the construction of chess endgame tablebases from a dynamic programming perspective. A simplified endgame model is developed as a finite state-space graph, and dynamic programming is used to propagate values from terminal positions to all reachable legal positions. The study focuses on a reduced endgame configuration to illustrate how optimal outcomes can be computed without repeated evaluation of the same states.**

Keywords—Dynamic Programming, Chess Endgame, Tablebase, Retrograde Analysis, Game Theory

I. Introduction

Chess is a globally popular deterministic two-player board game. The 64-square board game has a highly strict set of rules, which, if respected, leads the game into a set of legal positions and a given material configuration that has been recorded, analyzed, tracked, and optimized to produce a set of results, which are win, lose, or draw. These sets of material configurations and legal positions, with a limited number of pieces, combined with the minimum number of moves required to achieve the final outcome, are called chess endgame tablebases. Unlike heuristic evaluation functions, which some chess engine uses, these tablebases provide perfect accuracy in determining the optimal move given each position.

The construction of chess endgame tablebases was pioneered by Ken Thompson in 1986 [1]. The method used was retrograde analysis, which means the algorithm starts from terminal positions (e.g. checkmates and stalemates positions) and propagates value backward to predecessor positions. This is, at its core, Bellman's dynamic programming idea from 1957 [3], since the value of a state depends on the values of its successor states. Each position is a state, and the dynamic programming algorithm's recurrence assigns a value (win in k moves) based on successors. In effect, White is minimizing the distance-to-mate (DTM) if winning, and Black is maximizing (delaying mate).

Classic endgame research established many tablebases: e.g. Thomas Ströhlein (1970) [5] solved some 4-piece endings; Ken Thompson (1986) [1] built 4–5-piece databases using depth-to-conversion (DTC) as a metric. By 2005, all 6-piece endings were solved (Nalimov, Haworth, Heinz) [4], and by 2012, all 7-piece endgames were solved by the Lomonosov team at Moscow State University. Syzygy released a very compact 6–7 piece set in 2013-2018 [6]. In 2021, Marc Bourzutschky announced results from the first exploration into 8-piece positions, and by 2026, some subset of practical 8-piece positions is available publicly [7, 8]. These modern tablebases use DTM, DTC or depth-to-zero (DTZ) metrics, and heavily exploit symmetry and compression to fit within currently available computing power.

That being said, the objective of this paper is to explain how dynamic programming can be applied to construct chess endgame tablebases. A simplified endgame model is analyzed to demonstrate how state values can be computed. The King and Rook versus King (KRK) endgame is used as a case study because it is simple enough to analyze while still illustrating the essential ideas behind the method.

II. Chess & Endgame Tablebases

A. State-space Complexity

To understand the necessity of endgame tablebases, we must first examine the computational limits of chess as a mathematical construct. In 1950, Claude Shannon calculated the game-tree complexity of chess to be approximately 10^{120} [10], while later research established a state-space complexity ranging between 10^{43} and 10^{47} legal positions.

He first lays out his baseline assumptions to estimate the game-tree complexity:

1. A typical chess position has about 30 legal moves.
2. A single full "move" (one ply/half move for White, one ply for Black) gives about $30 \times 30 \approx 103$ possibilities.
3. A typical game lasts about 40 full moves (80 plies).

To quote Shannon: "A machine operating at the rate of one variation per micro-second would require over 1090 years to calculate the first move!"

Because this state-space is much larger than what can be exhaustively searched by any classical computer, chess engines (e.g. Stockfish, Leela, etc.) must rely on heuristic evaluation functions and forward-search algorithms, such as Minimax with

Alpha-Beta pruning. While these algorithms are highly optimized, they only approximate the optimal moves by searching up to a finite depth k. They evaluate positions based on material and positional factors, but do not know the exact sequence of moves required to reach a definitive, deterministic outcome.

B. The Endgame Shift

The endgame is the final stage of a chess game, occurring after a significant number of pieces have been exchanged. Compared with the opening and middlegame, the endgame has fewer tactical possibilities, but its strategic precision becomes more important. Small differences in king placement, mobility, and opposition can decide the result of the game. Thus, the branching factor and total state space diminish.

This reduction creates a paradigm shift: the problem transitions from heuristic approximation to deterministic graph solving. In this phase, the game allows for the construction of endgame databases, as the size of the state space decreases as the game approaches its end [10].

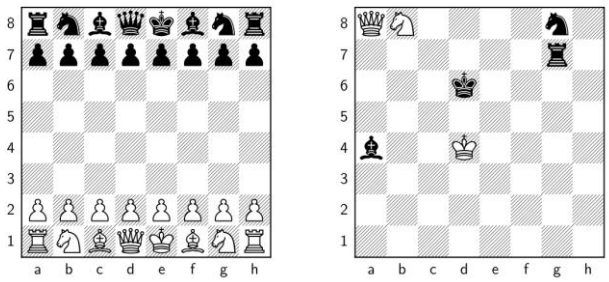

Fig. 1. Normal chess starting position (left) and a chess endgame position with a forced mate in 545 moves (right). *Note: Right image taken from [2], adapted and modified.*

The board can be modeled as a finite, directed state-space graph where every vertex represents a unique legal arrangement of the remaining pieces. Because the number of pieces is highly constrained, the entire subset of the game can be kept within computational memory limits, allowing for the calculation of perfect game-theoretic play.

C. Metrics in Tablebase

An endgame tablebase is a complete representation of a game's state space that assigns each position its corresponding game-theoretic outcome. It stores precomputed, non-heuristic evaluations for all legal positions involving a fixed number of pieces [11].

To compute optimal play, tablebases rely on several distance metrics:

1. Distance-to-mate (DTM)

   This metric measures the number of moves required to force a checkmate under perfect play from both sides. For example, a position with $DTM = 0$ is already checkmate, while $DTM = 1$ can be converted to mate in one move under optimal play. This is the strictest metric of absolute victory and was used in Nalimov tablebases.

2. Distance-to-conversion (DTC)

   This defines the minimum number of moves needed to force a material conversion into a simpler endgame. A conversion usually happens through a capture, delivering a checkmate, or a pawn promotion.

3. Distance-to-zero (DTZ)

   This measures the minimum number of moves until the next irreversible action, such as capture or any pawn move. This metric is primarily used to respect for the 50-move rule and is used by modern tablebases like Syzygy.

D. Retrograde Analysis as Dynamic Programming

Retrograde analysis is the technique of solving a game from the end backward. In chess endgames, the terminal positions are typically known: checkmate and stalemate. This excludes other rules, such as the 50-move rule and the threefold repetition rule. Starting from these positions, the algorithm traces back through the legal predecessor moves (states).

Since the state-space of the game of chess is large, this approach help reducing the amount of possible states, thus the result could be computed by the algorithm.

Retrograde analysis is the practical implementation of dynamic programming for tablebases. Each state is assigned a value once its outcome is logically determined by the values of its successors. The winning side acts to minimize the transition cost (e.g. number of moves to checkmate), while the defending side acts to maximize it. This overlapping subproblem structure allows the algorithm to calculate the perfect evaluation of every legal state without any redundant recalculations.

E. Hardware Constraints

In theory, the dynamic programming framework of retrograde analysis can scale to any number of pieces. However, in reality, it is heavily limited by computer hardware. The main problem is combinatorial explosion: when we add more pieces to the board, the number of possible states grows exponentially. This fast growth creates two major engineering challenges. First, it requires a massive amount of RAM (memory) during the generation phase. Second, it requires huge storage space for the final deployment.

As mentioned before, the first computer endgame tables were calculated by Thomas Ströhlein in 1970. Since then, the need for storage has grown very fast. Traditional tablebase designs focus on the exact DTM metric. This means they must store an explicit integer move-count for every single position. For example, 5-piece tables in the popular Nalimov format take up 7.1 gigabytes of uncompressed space. By 2012, the Lomonosov team generated 7-piece tablebases that needed about 140 terabytes of storage. This massive size meant they had to use a supercomputer.

Because of this, modern endgame tablebases must use advanced lossless compression methods. This helps the data fit

into normal computers while keeping the search queries fast. The popular Syzygy format makes the file size much smaller by splitting the data into Win-Draw-Loss (WDL) tables and DTZ tables. It drops the exact DTM mapping to achieve extreme bit-compression. Table 1 shows this huge jump in computational complexity. It also compares the resource trade-offs between uncompressed DTM formats and compressed formats for different piece counts.

TABLE I. REQUIREMENTS BY PIECE COUNT AND FORMAT

Piece Count	*Format*	*Legal Positions*	*Size*	*RAM*
3–5	Nalimov	26 billion	7.1 GB	16–32 GB
	Syzygy		938 MB	
6	Nalimov	3.7 trillion	1.2 TB	16–32 GB
6	Syzygy		150 GB	
7	Lomonosov	423 trillion	140 TB	~1 TB
7	Syzygy		18.4 TB	
8	Bourzutschky (partial)	~38 quadrillion	68.8 TB (partial)	~64 TB (estimate)

Fig. 2. Computational resource and storage requirements for endgame tablebase generation by piece count and format. *Note: Taken from [8, 12, 13], adapted and modified. The 8-piece tablebase has not yet been fully generated in a complete format due to extreme hardware constraints, requiring an estimated 2 Petabytes of total storage and 64 Terabytes of RAM.*

III. METHODOLOGY

To apply the theoretical principles of retrograde analysis, this section details how to construct the endgame tablebase. By defining the board as a mathematical graph, we use dynamic programming to calculate the optimal outcome for the King and Rook versus King (KRK) endgame without redundant state evaluations.

A. Problem Formulation

Formally, the endgame is written as a directed state-space graph:

$$G = (V, E), \tag{1}$$

where every vertex $v \in \mathrm{V}$ represents a unique, legal board position, and each directed edge $(u, v) \in E$ represents a legal move from position u to position v.

Each state is represented as

$$v = (\mathrm{w_k}, \mathrm{w_r}, \mathrm{b_k}, t), \tag{2}$$

where $\mathrm{w_k}, \mathrm{w_r}, \mathrm{b_k}$ denote the board coordinates of the White King, White Rook, and Black King, respectively, while

$$t \in \{White, Black\} \tag{3}$$

indicates which side to move.

Only legal chess positions are included in V. Illegal configurations, such as overlapping pieces, adjacent kings, or positions that violate the rules of chess, are excluded during state generation.

For every state v, let

$$Succ(v) = \{u \mid (v, u) \in E\} \tag{4}$$

denote the set of legal successor states, and

$$Pred(v) = \{u \mid (u, v) \in E\} \tag{5}$$

denote the set of predecessor states.

The objective of this study is to compute a value function

$$F: V \to \{Win, Draw, Loss\} \times N, \tag{6}$$

which assigns every legal position its game-theoretic outcome and its corresponding DTM, if applicable.

B. Dynamic Programming Formulation

The value assigned to a state depends on the values of its successor states. Thus, the problem satisfies Bellman's Principle of Optimality, making it suitable to be solved using dynamic programming.

Let $T \subseteq V$ denote the set of terminal states. These consist of positions in which the game has already reached its final outcome. For every terminal state,

$$F(v) = 0, \qquad v \in T \tag{7}$$

For all non-terminal positions, the recurrence depends on the player to move.

If it is White's turn and the position is winning, White chooses the move that minimizes the remaining distance to checkmate,

$$F(v) = 1 + min(u \in Succ(v))\, F(u) \tag{8}$$

whereas, if it is Black's turn, Black attempts to prolong the game as much as possible. Therefore,

$$F(v) = 1 + \max(u \in Succ(v))\, F(u) \tag{9}$$

These recurrence relations define the optimal value of every position. However, directly evaluating Equations (8) and (9) through recursion is impractical because the state graph contains cycles. Instead, the values are computed using retrograde analysis, which evaluates states in reverse order beginning from the known terminal positions.

C. Retrograde Analysis Formulation

Initially, all terminal checkmate positions are assigned a DTM value of zero. Their predecessor states are then examined according to two propagation rules.

A state is classified as winning if at least one legal move leads directly to a losing state for the opponent. Formally,

$$\exists u \in Succ(v): u \text{ is losing} \tag{10}$$

Otherwise, a state is classified as losing only when every legal move leads to a winning state for the opponent,

$$\forall u \in Succ(v): u \text{ is winning} \tag{11}$$

These two rules are repeatedly applied until no additional states can be assigned a value. Any remaining unlabeled states

are classified as draws because neither player can force a decisive result under optimal play.

The propagation process therefore computes both the game-theoretic outcome and the corresponding DTM value for every reachable legal position.

D. Algorithm

The overall construction procedure is summarized in Algorithm 1.

Algorithm 1 Retrograde Dynamic Programming

Require: Legal state graph $G = (V, E)$
Ensure: Tablebase F
Initialize $F(v) \leftarrow Unknown$ for all $v \in V$
Initialize an empty queue Q
for all terminal checkmate states v **do**
    $F(v) \leftarrow 0$
    Push v into Q
end for
while Q is not empty **do**
    $v \leftarrow$ Pop(Q)
    for all $p \in Pred(v)$ **do**
        if $F(p)$ is Unknown **and** $\exists u \in Succ(p)$ such that u is losing **then**
            Mark p as Winning
            Assign the corresponding DTM value
            Push p into Q
        else if $F(p)$ is Unknown **and** $\forall u \in Succ(p)$, u is Winning **then**
            Mark p as Losing
            Assign the corresponding DTM value
            Push p into Q
        end if
    end for
end while
for all remaining states $v \in V$ **do**
    if $F(v)$ is Unknown **then**
        Mark v as Draw
    end if
end for
return F

Fig. 3. Backward value propagation in the state-space graph using retrograde dynamic programming.

Algorithm 1 uses a bottom-up dynamic programming approach known as tabulation. Rather than using top-down recursion or memoization, tabulation builds the complete solution outward from the known base cases. By employing a queue data structure Q, the algorithm guarantees that the assigned DTM values represent the absolute shortest path to victory, or the longest path to defeat.

The process initializes by identifying terminal checkmate states, assigning them a DTM of 0, and pushing them into the queue. During each iteration, a solved state v is popped, and its immediate predecessors $p \in Pred(v)$ are evaluated.

IV. Implementation and Results

The theoretical framework was implemented through code using the Python programming language and the source code is available in the project repository [14].

To evaluate the King and Rook versus King (KRK) endgame, the algorithm must first construct the entire state-space. Iteratively placing the three pieces across the board yields $64 \times 64 \times 64$ possible physical combinations. Because each configuration can occur with either White or Black to move ($t \in \{White, Black\}$), this total is multiplied by two, resulting in a theoretical state-space of 524,288 combinations.

After filtering out invalid configurations (such as overlapping pieces or illegal checks), the final graph is reduced to exactly 399,112 valid legal positions. Within this state-space, the algorithm successfully identified 216 terminal checkmate nodes.

```
Generating state space graph (V, E)...
Graph built: 399112 valid states, 216 checkmate nodes.

Running retrograde dynamic programming wave...
Tablebase complete.
```

To verify the correctness of the generated tablebase, the algorithm was tested against specific KRK configurations. The primary objective is to confirm that the retrograde dynamic programming accurately minimizes the DTM for the winning side (White) while maximizing the survival plies for the defending side (Black). The positions are represented using standard Forsyth-Edwards Notation (FEN).

a. Verification of shortest path (Mate in 2):

The first test evaluates a position where White possesses a material advantage but must maneuver precisely to trap the Black king on the edge of the board without allowing an escape.

Initial FEN: `7k/R7/5K2/8/8/8/8/8 w - - 0 1`

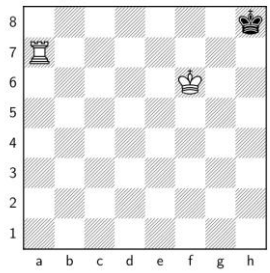

Fig. 4. Initial board configuration for Test Case 1. White to move and force mate.

```
Output
Evaluating FEN: 7k/R7/5K2/8/8/8/8/8 w - - 0 1
DP Result:  Win in 3 plies (half-moves)
------------------------------------------------
[DTM 03] 7k/R7/5K2/8/8/8/8/8 w - - 0 1
         --> White plays Kf6-g6
[DTM 02] 7k/R7/6K1/8/8/8/8/8 b - - 0 1
         --> Black plays kh8-g8
[DTM 01] 6k1/R7/6K1/8/8/8/8/8 w - - 0 1
         --> White plays Ra7-a8
[DTM 00] R5k1/8/6K1/8/8/8/8/8 b - - 0 1 (CHECKMATE)
```

This execution trace confirms that the Bellman optimality condition correctly guides the sequence. The engine prioritizes restricting the opponent's mobility (Kf6-g6) over an ineffective immediate check, arriving at the absolute minimum DTM.

b. Verification of mating net execution (Mate in 3):

A more complex scenario was tested to observe the dynamic programming algorithm's ability to construct a multi-move mating net from a randomized winning position.

Initial FEN: `7k/8/8/4K3/8/8/8/R7 w - - 0 1`

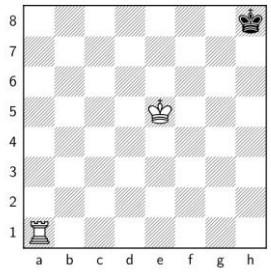

Fig. 5. Initial board configuration for Test Case 2. White to move and execute a mating net.

```
Output
Evaluating FEN: 7k/8/8/4K3/8/8/8/R7 w - - 0 1
DP Result:  Win in 5 plies (half-moves)
--------------------------------------------------
[DTM 05] 7k/8/8/4K3/8/8/8/R7 w - - 0 1
         --> White plays Ke5-f6
[DTM 04] 7k/8/5K2/8/8/8/8/R7 b - - 0 1
         --> Black plays Kh8-g8
[DTM 03] 6k1/8/5K2/8/8/8/8/R7 w - - 0 1
         --> White plays Ra1-h1
[DTM 02] 6k1/8/5K2/8/8/8/8/7R b - - 0 1
         --> Black plays Kg8-f8
[DTM 01] 5k2/8/5K2/8/8/8/8/7R w - - 0 1
         --> White plays Rh1-h8
[DTM 00] 5k1R/8/5K2/8/8/8/8/8 b - - 0 1 (CHECKMATE)
```

This output demonstrates that the engine correctly forces the Black king into opposition before delivering the fatal check, aligning perfectly with established game-theoretic play for KRK endgames.

c. Verification of depth (Longest force mate):

To ensure the dynamic programming queue can scale to the deepest possible branches of the state-space graph without cyclical errors, the algorithm was tested on the longest known forced mate in the KRK endgame. The pieces are placed as far apart as possible, requiring White to systematically force the Black king from the center to the edge.

Initial FEN: `1R6/8/8/8/4k3/8/8/K7 w - - 0 1`

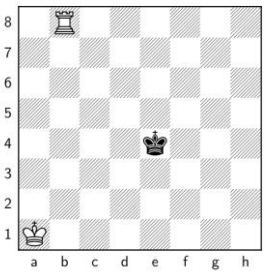

Fig. 6. Initial board configuration for Test Case 3. The pieces are maximally dispersed, representing the worst-case scenario for White.

```
Output
Evaluating FEN: 1R6/8/8/8/4k3/8/8/K7 w - - 0 1
DP Result:  Win in 31 plies (half-moves)
--------------------------------------------------
[DTM 31] 1R6/8/8/8/4k3/8/8/K7 w - - 0 1
         --> White plays Ka1-a2
[DTM 30] 1R6/8/8/8/4k3/8/K7/8 b - - 0 1
         --> Black plays Ke4-d4
...  (Output continues) ...
[DTM 03] 8/8/8/8/8/1R3K2/8/7k w - - 0 1
         --> White plays Kf3-g3
[DTM 02] 8/8/8/8/8/1R4K1/8/7k b - - 0 1
         --> Black plays Kh1-g1
[DTM 01] 8/8/8/8/8/1R4K1/8/6k1 w - - 0 1
         --> White plays Rb3-b1
[DTM 00] 8/8/8/8/8/6K1/8/1R4k1 b - - 0 1 (CHECKMATE)
```

This verifies that the traversal mechanism processes the overlapping subproblems correctly across the entire combinatorial span of the board, matching the established mathematical limit of the KRK endgame.

d. Verification of draw by insufficient material:

The algorithm must also correctly identify positions where optimal play cannot force a win, ensuring that invalid "Win" states are not hallucinated. This test case places the Black king directly adjacent to an undefended White rook.

Initial FEN: `8/8/8/4k3/4R3/8/8/K7 b - - 0 1`

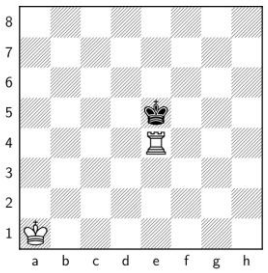

Fig. 7. Initial board configuration for Test Case 4. Black to move and capture the rook.

```
Output
Evaluating FEN: 8/8/8/4k3/4R3/8/8/K7 b - - 0 1
DP Result:  Forced Draw
--------------------------------------------------
```

By capturing the rook (Kxe4), Black transitions the board into a King versus King state. Because the graph generator evaluated that no successors from that state can ever reach a terminal checkmate node, the state remained "Unknown" throughout the entire while loop. The algorithm correctly executed its final sweep, permanently marking the position and its predecessors as a forced Draw.

e. Verification of stalemate avoidance:

A critical flaw in heuristic search engines is the horizon effect, which can sometimes lead to accidental stalemates if the engine miscalculates the depth of the tree. The dynamic programming tablebase circumvents this entirely. This test case presents a scenario where a careless king moves by White (e.g., Kf7) would result in a stalemate.

Initial FEN: `8/8/8/4k3/4R3/8/8/K7 b - - 0 1`

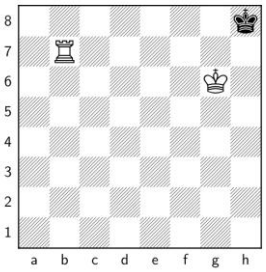

Fig. 8. Initial board configuration for Test Case 5. White to move, with immediate checkmate or stalemate possibilities.

```
Output
Evaluating FEN: 7k/1R6/6K1/8/8/8/8/8 w - - 0 1
DP Result:  Win in 1 plies (half-moves)
--------------------------------------------------
[DTM 01] 7k/1R6/6K1/8/8/8/8/8 w - - 0 1
         --> White plays Rb7-b8
[DTM 00] 1R5k/8/6K1/8/8/8/8/8 b - - 0 1 (CHECKMATE)
```

The output proves that the algorithm mathematically guarantees victory. By storing the DTM in the dictionary F, the engine recognizes Rb8# as a $DTM = 1$ state, whereas any move leading to a stalemate is classified as a "Draw". The algorithm selects the minimal winning distance, bypassing the stalemate risk.

V. Conclusion

This paper presented a dynamic programming approach to the construction of chess endgame tablebases through retrograde analysis. By modeling the King and Rook versus King (KRK) endgame as a finite directed state-space graph, the study demonstrated how game-theoretic outcomes and their corresponding DTM values can be propagated efficiently from terminal positions to all reachable legal states without redundant computation.

The implementation successfully generated a complete KRK tablebase consisting of 399,112 legal positions and correctly identified winning, losing, and drawing states. Experimental verification using multiple test cases showed that the algorithm consistently produced optimal play, including shortest forced mates, correct draw detection, and avoidance of stalemate positions. These results confirm that retrograde analysis naturally satisfies Bellman's Principle of Optimality and serves as an effective bottom-up dynamic programming technique for solving finite deterministic games.

Although the KRK endgame was selected because of its manageable state-space, the same dynamic programming framework can be extended to more complex endgames. However, the rapid growth of the state-space introduces significant computational and storage challenges, making efficient compression techniques and high-performance computing essential for larger tablebases.

Video Link at Youtube

https://youtu.be/YDKlKrTQwDQ

Acknowledgment

The author would like to thank Dr. Ir. Rinaldi Munir, Ir. Rila Mandala, and Dr. Muhammad Zuhri Catur Candra for their guidance and support throughout the IF2211 Strategy Algorithm course. Appreciation is also extended to fellow students for their discussions and feedback during the preparation of this work.

References

[1] K. Thompson, "Retrograde Analysis of Certain Endgames," ICGA Journal, vol. 9, no. 3, pp. 131–139, Sep. 1986, doi: 10.3233/ICG-1986-9302.

[2] ChessOK, "Top-8 DTM Tablebase." [Online]. Available: https://tb7.chessok.com/articles/Top8DTM_eng

[3] R. Bellman, Dynamic Programming. Princeton, NJ, USA: Princeton University Press, 1957.

[4] [4] G. Haworth, "6-Man Chess Solved," ICGA Journal, vol. 28, no. 3, pp. 153–153, Sep. 2005, doi: 10.3233/ICG-2005-28304.

[5] Chessprogramming Wiki, "Thomas Ströhlein." [Online]. Available: https://www.chessprogramming.org/Thomas_Str%C3%B6hlein

[6] Chessprogramming Wiki, "Syzygy Bases." [Online]. Available: https://www.chessprogramming.org/Syzygy_Bases?9ePZY=IsfnjuQgyK

[7] ARVES, "8-Men Tablebase: First Explorations." [Online]. Available: https://www.arves.org/arves/index.php/en/endgamestudies/theory/1509-8-men-tablebase-first-explorations

[8] Lichess, "OP1: Partial 8-piece Tablebase Available," Lichess Blog, 2024. [Online]. Available: https://lichess.org/@/Lichess/blog/op1-partial-8-piece-tablebase-available/1ptPBDpC

[9] C. E. Shannon, "Programming a Computer for Playing Chess," Philosophical Magazine, vol. 41, no. 314, pp. 256–275, 1950. [Online]. Available: https://vision.unipv.it/IA1/ProgrammingaComputerforPlayingChess.pdf

[10] M. A. J. H. Donkers, H. J. van den Herik, and J. W. H. M. Uiterwijk, "Perfect Knowledge Revisited," Artificial Intelligence, vol. 134, no. 1–2, pp. 71–92, 2002, doi: 10.1016/S0004-3702(01)00152-7.

[11] D. Gomboc, C. R. Shelton, A. S. Miner, and G. Ciardo, "Comparing Lossless Compression Methods for Chess Endgame Data," in Proc. ECAI 2024, Frontiers in Artificial Intelligence and Applications, vol. 392, pp. 4116–4123, 2024, doi: 10.3233/FAIA240982.

[12] K. W. Regan, "Rating Computer Science via Chess," in Computing and Software Science, Lecture Notes in Computer Science, vol. 10000, Cham, Switzerland: Springer, 2019, pp. 200–216, doi: 10.1007/978-3-030-04203-5_12.

[13] Chessprogramming Wiki, "Endgame Tablebases." [Online]. Available: https://www.chessprogramming.org/Endgame_Tablebases

[14] https://github.com/Vearance/itb/tree/main/sem-4/makalah-stima

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, 19 Juni 2026

Nathaniel Christian – 13524122