MEDWIN PUBLISHERS
Committed to Create Value for Researchers

Open Access Journal of Data Science & Artificial Intelligence
ISSN: 2996-671X

An Optimization Method for Quiescence in Chess-Playing Automata

Wheeler SF*

Department of Information Science, University of North Texas, USA

***Corresponding author:** Stephen F Wheeler, Department of Information Science, University of North Texas, USA, Tel: 1 (214) 642-7868; Email: Stephen.Wheeler@unt.edu

Research Article

Volume 2 Issue 1

Received Date: August 01, 2024

Published Date: August 13, 2024

DOI: 10.23880/oajda-16000138

Abstract

The purpose of this paper is to report the results of a study of a method for improving the performance of the quiescence phase of Alpha-Beta (α-β) search. The α-β enhancement to the Minimax algorithm optimizes the performance of depth-first search when the solutions being searched are arranged in near best-first order reducing the computational effort from $O(b^d)$ to $O(\sqrt{b^d})$, where b is the branching factor of the game-tree and d is the depth of the search, Knuth and Moore. To postpone the asymptotic behaviour of the combinatorial explosion, a full breath search is only carried out to 5 levels of depth in this research. A narrow width search expanding only solutions involving the exchange of material, a pawn promotion or king-in-check situation is then expanded until the position reaches quiescence where no material exchanges or promotions are present. When quiescence is reached, the evaluation function is called to score the leaf node of the game-tree. For chess-playing programs, it has long been held that material exchanges should be explored first before other solutions are expanded to ensure optimum performance of α-β pruning, Gillogly. The research reported in this paper has established statistically that α-β pruning is improved if a solution not involving an exchange of material or promotion is tried first rather than a material exchange solution during the quiescence phase of α-β search.

Keywords: Alpha-Beta; Minimax; Negamax; Optimization; Quiescence

Abbreviations

MCTS: Monty Carlo Tree Search; HAL: Heuristic Associative Linear-Algorithm; PVB: Provisional Backed-up Value; NLP: Natural Language Processing; OS: Operating System.

Introduction

In 1944, mathematician, physicist, and computer scientist, John von Neumann in collaboration with an economist, Oskar Morgenstern, authored their seminal work, "Theory of Games and Economic Behaviour" [1]. Together they conceived a revolutionary mathematical theory of economic and social organization based on a theory of games of strategy. Not only would this theorem revolutionize economics, but it would give rise to an entirely new field of scientific inquiry known as Game Theory and the Minimax Theorem.

Claude E Shannon [2], an MIT mathematician and Bell Labs researcher is regarded as the founder of information theory and the father of computer chess, applied this principle in a paper he published in 1950 titled "Programming a Computer for Playing Chess" [2]. By applying game theory to the chess problem he identified the principles that form the essence of today's chess playing programs. Since the publication of Shannon's paper, programming a computer for playing chess has moved from a conceptual framework to one of the early

cornerstones of artificial intelligence research. Chess was then and is now the Drosophila Melanogaster (fruit fly) of artificial intelligence research.

Minimax Search, the Essence of Applied Game Theory

The Minimax Theorem is the algorithmic realization of Game Theory in two-player, zero-sum games of perfect information, such as chess. In the minimax theorem, the two players are Min and Max. The algorithmic realization of Minimax requires a Minimizer and a Maximizer. The Minimizer returns the minimum of the values produced by the Maximizer and passes that value one level up the game-tree toward the root of the game-tree, if that value is less than the value previously passed up one level of the game-tree. The Maximizer, on the other hand, returns the maximum of the values produced by the Minimizer and passes that value one level up the game-tree if that value is greater than the value previously passed up. This process continues until the best move, from the machine's perspective, has been backed up the game-tree to the root node of the game-tree. Minimax search discovers the Principle Continuation that is the best path through the game-tree from the root node to the leaf node of the game-tree with the best minimax value [3]. A four ply illustration of a minimax game-tree is shown in Figure 1 with its backed-up values shown in parentheses.

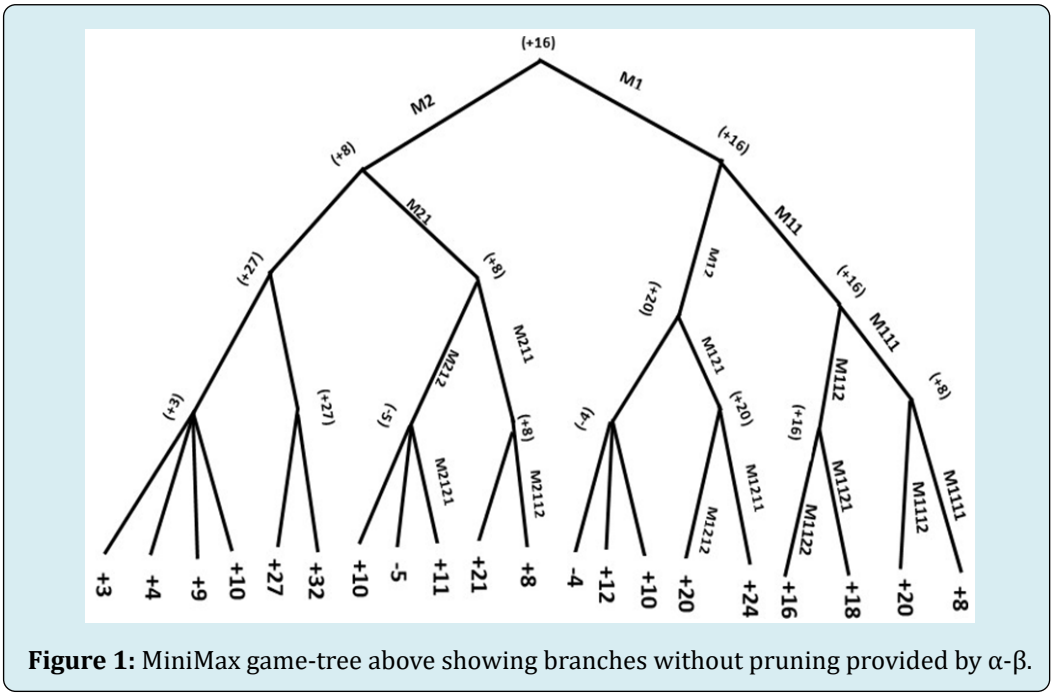

Figure 1: MiniMax game-tree above showing branches without pruning provided by α-β.

Alpha-Beta (α-β) Enhancement to Minimax

The discovery of the α-β extension of minimax search is not clear. Arguably, the most thorough and detailed description of the α-β extension was described by Donald E Knuth [4] and Ronald W Moore [5] of Stanford University who described the algorithm in a paper they published in 1975 Knuth and Moore [4]; Marsland [5]. Although, Knuth and Moore were among the first to describe the algorithm, other researchers included some of the principles of the α-β extension of minimax in their experimental models, including Arthur Samuel of IBM in his checkers playing program, [6]. Without α-β pruning, the minimax game-tree grows exponentially, where b is the branching factor of the game-tree and d is the depth of search, where the effort coefficient is described as $O(b^d)$. With α-β enhancement of minimax search, when the game-tree is arranged in near best-first order, the game-tree growth rate is reduced to $O(\sqrt{b^d})$, Knuth and Moore [4]. The odd levels of the α-β game-tree including the root node are Alpha nodes. Even levels of the α-β game-tree consist of Beta nodes. Alpha nodes of the game-tree maximize and Beta nodes minimize. During α-β search of the game-tree a node that has a returned value that is less than Alpha or that is greater than Beta will be pruned from the game-tree search as an Alpha or Beta cutoff. Alpha and Beta are dynamic values that when taken together form what is known as the Alpha-Beta window. The α-β window narrows as α-β search progresses and more nodes and branches of the game-tree are pruned whose values fall outside of the narrowing α-β window [4,7].

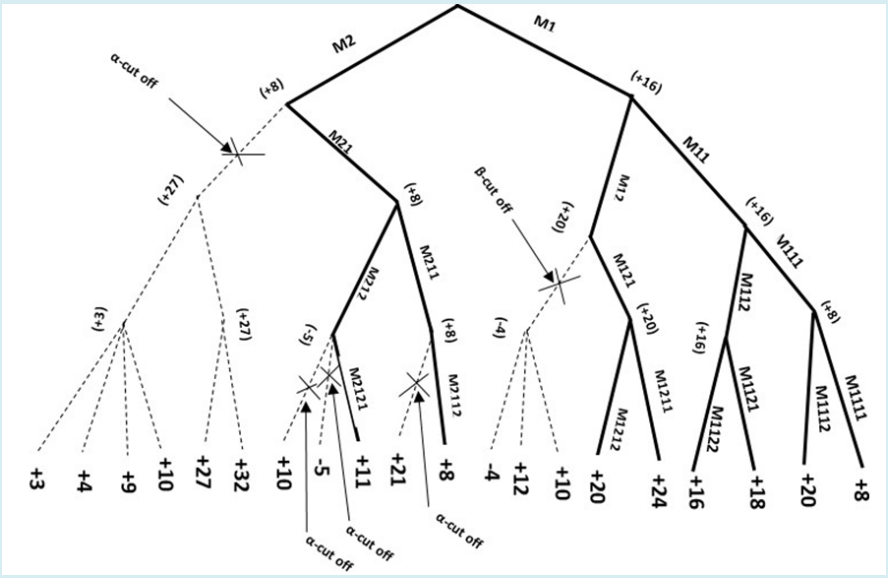

Figure 2: Alpha-Beta game-tree above showing branches pruned as α and β cut offs.

Monty Carlo Tree Search (MCTS) forms the search component of Deep Mind's Alpha Zero reinforcement machine learning algorithm that was trained to play chess, beat Stock fish 8 in a computer chess match. Independent research, however, has shown that in certain applications α-β search might out-perform MCTS [8]. The Komodo-mc chess program, for example, has effectively replaced α-β search with MCTS. Komodo-mc plays at Grandmaster level, but not quite as strong as Komodo with α-β search presently. Komodo's FIDE rating is 3200. Komodo is the world's second highest rated linear symbolic chess-playing computer program. Stock fish 11 is currently the highest rated chess program in terms of player strength.

HAL (Heuristic Associative Linear-algorithm), this author's chess-playing program has never been officially rated in terms of its player strength. However, in a chess game played between HAL and Komodo on chess.com, Komodo required 50 moves to defeat HAL, and HAL was not playing at its highest level of play [9].

The forward pruning of α-β search, cuts-off or prunes branches of a strongly ordered game-tree whose backed-up values fall outside the established α-β search window as illustrated in Figure 2.

The Negamax Implementation of Alpha-Beta Search

Negamax is a hybridization of the Alpha-Beta algorithm. As mentioned earlier, the Alpha-Beta al- gorithm contains both a maximizer and a minimizer, while the Negamax algorithm contains only a Maximizer [5]. Negamax returns the maximum score for the side whose turn it is to play. The maximum returned score for the opponent player is then negated making the result behave as the value returned by the minimizer in standard Alpha-Beta. Hence the name Negamax, the negative of the maximum returned score. Negamax pruning is identical to Alpha-Beta pruning, but is more easily implemented as a recursive function [10]. The Pascal source code logic for the Negamax function of this author's chess program, HAL, follows below.

```
(* ALPHA–BETA *)
(* NegaMax Fail–Soft Alpha–Beta (NFAB) *)

FUNCTION NFAB (P: POINTER; Alpha, Beta, Depth: INTEGER): INTEGER;
  VAR
        I, Width: INTEGER;
        Score, Value: INTEGER;
        NewPly: POINTER;
  BEGIN
        IF (Depth <= 0 ) THEN
         BEGIN
(* If depth = 0 then Call the EVAL Function and score the terminal node *)
          NFAB: = Eval (P)
      END
    ELSE
      BEGIN
          IF (P ^ . LX > 1) THEN
            (* I f Level > 1 and not a t e r m i n a l node *)
             BEGIN
                Width: = Generate (P);
                    (* Gen new branch of Game–Tree *)
```

```
                IF (Width > 0) THEN
                  BEGIN
                        Locate-Refutations (P);
        (* Locate Refutations and Killer moves then
*)
                  Order (P)
        (* Sort this branch (level) of the Game–Tree
*)
                  END
             ELSE
                BEGIN
                  NFAB: = Eval (P)
                        (* Otherwise call Evaluator *)
                END
           END;
 Score: = –INF;    (* set maximizer to low–score *)
 I: = 0;           (* initialize move index *)
 NewPly: = Create (P);
             (*Create a new level of the Game–Tree
*)
  REPEAT
        I: = I + 1;          (* Increment move index *)
        Make (P);
             (* make move at PI then call Alpha–
        Beta *)
        Value: = –NFAB (NewPly, –Beta, –Max
(Alpha, Score),
                  Decrement (P, Depth) );
        Undo (P);
             (* After Alpha–Beta s cored move,
        unmake move *)
          (* If Move i s an Improvement, Save Better
        Score *)
          IF (Value > Score) THEN
            BEGIN
                  Score: = Value;      (*  save
        better score *)
                  Save _ Refutation (P)
                        (* save better move in
refutationtable *)
                  END;
        (* Build Refutation Table from Triangular Work
        Array *)
          IF (P ^. LX = 1) THEN
                (* IF first level of Game–Tree *)
            BEGIN
              Build _ Refutation _ Table (P)
            END;
                  (* Test for end of  Iteration
              Sequence at Level P ^. LX *)
          UNTIL ( (Score >= Beta) OR (I = Width) );
             (* Put KILLERS into  Killer List *)
          IF ( (Score >= Beta ) AND (P ^. LX > 1) )
        THEN
            BEGIN
              Save _ Killer (P);
          END;
        (* Back Score up to previous level of the Game–Tree
*)
          Drop (NewPly);
          (* drop newply of the Game–Tree *)
          NFAB: = Score (* Return best score *)
        END
 END;                    (* End of NFAB Alpha–Beta
Function *)
```

Code 1: Pascal code for a Negamax version of the depth-limited Alpha-Beta function

Quiescence Extension of Alpha-Beta

Quiescence search is a narrow extended search after the full-width fixed-depth α-β search reaches its fixed-depth limit. This is known as the search horizon (Figure 3).

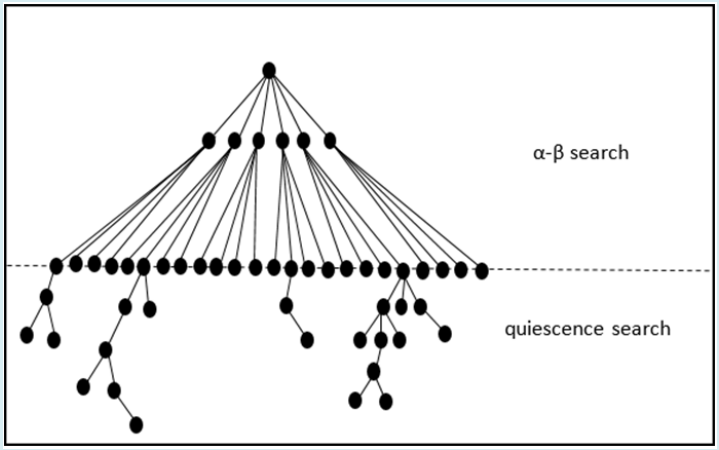

Figure 3: Quiescence Extension to Alpha-Beta Search

By extending α-β search with a narrow but deep quiescence search, the horizon effect can be addressed and eliminated. The horizon effect refers to a condition occurring when a fixed-depth, exhaustive α-β search ends on a terminal-node that contains an action that might be incomplete without knowing the consequence of that action if the next level of the game-tree were known, Berliner [11]. For example, if the fixed-depth search ends at a terminal-node on the search horizon that is a capture of an opponent's piece, the returned score of the exchange of material might appear to be a good action for the program, but had the search been extended another ply of depth we might discover that the opponent would capture the capturing piece. The net value of the exchange might be less than favourable for the machine. By extending the search with a narrow search until the position becomes quiet where no exchanges occur, the program can avoid the consequences of the horizon effect. This narrow extended search is known as quiescence search, Marsland and Schaeffer [12]. In addition to single-ply extensions for capture moves, many implementations of quiescence search also extend for king-in-check situations for one or two additional ply of depth. Some implementations of the quiescence phase of α-β search also extend additional plies of depth when a pawn promotion is present at a terminal-node [13]. This author's chess program, HAL, will also extend an additional ply of depth for a capture, pawn promotion or a king-in-check situation.

In this author's chess-playing program, HAL, the quiescence phase of α-β search is managed by a Decrement function that is called as a parameter within the recursive call to the α-β search function, NFAB. In a recursive call to α-β search (NFAB) we must include a reference to a variable that contains a value initially indicating to what depth (number of plies) we wish the full-width fixed-depth α-β search to be performed. If, for example, we wish the program to generate a fixed-depth game-tree of five levels of depth, we would initialize the value of that variable to 5 and decrement it by 1 each time another recursive call to the α-β search function is performed. When the Depth variable is decremented to 0, five levels (or ply) of the game-tree have been generated. At that point the program has reached the terminal-node (leaf node) of that path through the game-tree. An Evaluation function is then called to score the value ascribed to that path through the game-tree.

In this author's chess-playing program, HAL, the depth-limit for recursive calls to the α-β search function is located in an integer variable named Depth. This variable is passed to the Decrement function as an imbedded parameter within the recursive call to the α-β search function NFAB thusly:

Value: = –NFAB (NewPly, –Beta, –Max (Alpha, Score), Decrement (P, Depth));

It should be noted that the initial call to the Alpha-Beta algorithm NFAB is as follows:

Value: = NFAB (P, Alpha, Beta and Depth));

The Decrement function will then decide whether to decrement the Depth variable or not. If the Decrement function does not decrement the Depth variable, then it has found that the move at the terminal-node is a capture, a pawn promotion, or a King-in-check situation, and an additional level of the game-tree will be generated until Quiescence has been reached. When Quiescence is reached during the Quiescence phase of α-β search the terminal position is said to be a Dead position to which the Evaluation function (Eval) is invoked to compute the value of that path through the game-tree that ends on the terminal-node in question [14]. The value returned by the Evaluation function will be backed-up the game-tree toward the root-node of the game-tree by the rules of Minimax. The value returned by the Evaluation function is known as the Provisional Backed-up Value (PVB) [3]. The Pascal source code for the Decrement function of the HAL chess program follows below.

```
(* QUIESCENCE *)
(* Decrementor for Depth Value in Quiescent NFAB Search Function *)
FUNCTION Decrement (P: POINTER; N: INTEGER): INTEGER;
  VAR
    D, I, L, S, X: INTEGER;
  BEGIN
    S: = P ^. SX;
    (* S = Side whose turn it is to move, White = 1, Black = 2 *)
    I: = P ^. IX;
    (* I = Index of move i n Move–List that might be a capture or Promotion *)
    L: = P ^. LX;
    (* L = Level of depth of the Game–Tree being expanded *)
    D: = N – 1;
    (* D = Recursive alpha –beta value of full –width level syet to be expanded *)
    X: = S + ( ( S – 1 ) * –2 ) + 1;
    (* X = Opponent; If S = 1 t hen X = 2, I f S = 2 t hen X = 1 *)
    (* Test for Singular Extension for Quiescence Move Generation *)
  IF (N = 1) THEN
      (* A Leaf –Node of the Game–Tree has been reached *)
  BEGIN
    IF ( (L = Horizon) OR (L = Quiescence) OR
      (P ^. Move_List [P ^. MLI [I] ] . Mode_Cell = 'M') )
```

```
        THEN
                (* Mated or Search Horizon Reached *)
          BEGIN
            D: = 0
          END
      ELSE
          BEGIN
             IF (P ^ . Move_List [P ^ . MLI [I] ] . Mode_Cell IN
[ 'C ', P'] )
              THEN
                (* Capture or Pawn Promotion *)
              BEGIN
                D: = N
                (* IF a Capture or Pawn Promotion, Don't
          decrement Depth *)
               END
                (* Allow one more level to be generated *)
          ELSE
              BEGIN
                D: = N-1
                        (* Else, IF not a Capture or
                  Promotion, decrement Depth *)
                            END
                       END
                  END
             ELSE
            BEGIN
                D: = N – 1          (* Else, IF None of above,
          decrement Depth *)
          END;
      Decrement: = D     (* Return Depth value to Alpha–Beta
Recursive Call *)
END;
```

Code 2: Pascal code for the Quiescence search function.

Optimization Method for Quiescence in Alpha-Beta Search

Quiescence search might extend the search many additional levels before reaching quiescence. If the program can cause a cut off to occur before extending search to quiescence, a large amount of time and computational effort can be saved to achieve the same result. The way to achieve this is to try a move that is not a capture, a pawn promotion or a king-in-check situation before expanding a capture or promotion move or a king-in-check situation. During the quiescence phase of α-β search the move- list is sorted bringing captures and pawn promotions to the top of the move-list. If after sorting there is a capture or promotion move at the top of the move-list, the algorithm will index up through the move-list until a move is found that is not a capture or pawn promotion. The move index for that move is then moved to a hold variable and all moves above the non-capture, non-promotion move are move down in the move-list leaving a place at the top of the move-list for the move saved in the hold variable. The move in the hold variable is then moved to the top of the move-list to be expanded first. If the move inserted at the top of the move-list does not because a cut off to occur, the next move in the move-list is expanded that will cause quiescence search to occur.

```
(* Quiescence Search Optimization Algorithm Called from
NFAB α-β *)

PROCEDURE Order (P: POINTER);
        VAR      H, I: INTEGER;
           (* Sort Move–List at P for width WX *)
BEGIN
        Sort (P, P ^ .WX);
        (* Leaf Node when Depth =1, LX i s Level Index *)
        (* Test for Capture or Promotion *)
        (* at the top of Move–List *)
        IF ( (P ^ . LX > 1) AND (Depth = 1) AND
              (P ^ . Move_List [P ^ . MLI [1] ] . Mode_Cell IN
['C ', 'P'] ) )
        THEN
        (* IF True Then Initialize Index I *)
        BEGIN
           I: = 1;
             WHILE ( ( P ^ . Move_List [ P ^ . MLI [ I ] ] .
        Mode_Cell
              IN ['C', 'P'] ) AND (I < P ^ .WX) ) DO
        (* Loop down Move–List incrementing I *)
        (* until a move is found that is not a Capture *)
        (* or a Pawn promotion *)
              BEGIN
                 I: = I + 1
              END;
              IF (NOT (P ^. Move_List [P ^. MLI [I] ] .
        Mode_Cell
                    IN [ 'C ','P'] ) AND ( I <= P ^ .WX ) )
           THEN
        (* IF the move a t index location in Move–List *)
        (* is not a Capture nor Promotion then *)
        (* move the index of that moves location *)
        (* to the Move–List index hold variable H *)
              BEGIN
                 H: = P ^. MLI [I];
                 FOR I: = (I – 1) DOWNTO 1 DO
        (* and then Move all moves in the Move–List *)
        (* down by one until the Move–List index = 1 *)
              BEGIN
                   P^. MLI [I + 1]: = P ^. MLI [I]
              END;
        (* then   insert H in the top of the Move–List *)
              P^. MLI [I]: = H
        END
     END
```

END;

Code 3: Pascal Code for Optimization of Quiescence Phase of α-β Search.

The Experiment

When the chess program, HAL, enters the quiescence phase of α-β search of the game-tree, the search is extended one additional ply (level) of depth if the last move considered was a capture move or a pawn promotion or king-in-check situation. This process continues until there are no captures or promotions produced and the search is said to have reached quiescence. At which point the static evaluator is called and the terminal position is scored and the returned value is backed up the game- tree toward the root node of the tree.

This research expands on the author's earlier published research involving the behaviour of α-β search on game-trees [15], and an optimization technique first introduced in theory by Dr. Monroe N [16] in his chess playing algorithm: Ostridge Newborn. Dr. Newborn [16] incorporated a one-move-generator to generate and try only the first move of a move-list before generating a complete list of all legal moves for the present board position. If the first move causes an Alpha or Beta cut off, generating a complete move-list is unnecessary. If a cut off does not occur on the first move, the full- move-list-generator is called and a list of all legal moves is generated for the current board position.

The Experimental Model

The experimental model that rendered the metrics reported in this research paper was created by its principal author Wheeler S [9]. The source code is written in Turbo Pascal 6.0, consisting of 8,000+ lines of source code. The program has been setup to report the metrics that support this research. The Turbo Pascal 6.0 compiler produces only a 32-Bit DOS executable (.exe). To execute HAL on a 64-bit Windows Operating System, an emulator, such as DOSBOX must be used. HAL can be executed on a Linux or UNIX operating system with DOSEMU.

The chess program, HAL, utilizes the Iterative-Deepening, Fail-Soft Negamax α-β search algorithm (NFAB) with Quiescence search singular extensions, the Killer Heuristic that maintains four Killer moves in its Killer-Table and a larger Refutation-Table of previously discovered moves that caused pruning to occur [17]. These methods collectively help ensures that moves with the highest probability of achieving maximum α-β pruning will be arranged at the top of the move-list after reordering of the move-list occurs. These processes are applied to each level of the game-tree as each new ply of legal chess moves is generated [5,9]. The chess program, HAL, has full communications capability through natural language processing (NLP) such that it can be configured through a series of English directives given to the chess automaton, HAL, at start-up. The hardware on which HAL was executed consisted of a Hewlett Packard, Elitebook 8530w laptop computer with 8 Gigabytes of DRAM memory and an Intel T9900 3.06 Gigahertz Core2 Duo CPU installed. The operating system (OS) was 32- Bit Windows Professional.

The Test Runs

To gather the metrics supporting the research reported in this paper, the chess program, HAL, was run twice. Both runs of the program were given the identical 24 chess moves. HAL (the chess program) played the black pieces in both instances. One execution of the chess program was setup to include the author's quiescence search optimization enhancement. The other execution of the chess program was setup to exclude the quiescence search optimization enhancement.

The setup for both runs of the program, HAL, only required setting a Boolean variable to true (quiescence optimization) or False (without quiescence optimization) through natural English typed commands. The two executions of the chess program were configured by commands given to HAL's initial NLP (Natural Language Processing) interface dialogue. To set the chess program up to run with quiescence optimization the following request was entered: "HAL, attempt quiescence optimization." To set the chess program up to run without quiescence optimization the following request was entered: "HAL, do not attempt quiescence optimization." HAL must also be requested to report the metrics for both program setups by entering the following request when the chess program prompts for user input, "HAL, report game statistics." HAL was also requested to play with skill level 5, which generates a 5-ply full-width game-tree before extending with a narrow-width quiescence search.

The Metrics Produced by the Experimental Model, HAL

The metrics produced by the two separate runs of the chess program, HAL, are included in the Table 1 above. One run was without quiescence optimization and the other was run with quiescence optimization. The following table shows the computation time and number of nodes processed for each of the twenty-four chess moves for both runs of the chess program, HAL.

Human White Moves	HAL Black Moves	Time with Opti- mization Min:Sec.Hnd	Time Without Opti- mization Min:Sec.Hnd	Nodes Expanded with Opti- mization	Nodes Expanded Without Optimization	Branches Pruned: Alpha-Beta Cutoffs without Optimization	Branches Pruned: Alpha-Beta Cutoffs with Optimization
e2-e4	e7-e5	0.000694	0.00.50	792	860	108	107
g1-f3	g8-f6	0.000891	0.01.43	1,18,987	1,19,031	15,200	14,066
f1-b5	f8-d6	0.000706	0.01.25	98,123	1,03,685	13,665	13,809
d2-d3	a7-a6	0.00.43	0.001146	56,116	85,608	11,257	9,740
b5-c4	b7-b5	0.00.31	0.000752	39,374	59,978	7,565	6,085
c4-b3	0-0	0.00.32	0.02.14	46,380	1,49,232	25,161	6,993
0-0	b8-c6	0.000949	0.07.58	1,42,661	7,73,163	67,940	17,437
a2-a3	c6-d4	0.04.51	0.14.50	7,78,288	11,50,234	1,35,374	78,988
b3-a2	c8-b7	0.04.17	0.16.15	78,55,657	16,06,844	1,46,695	75,755
h2-h3	d4xf3	0.012616	0.43.18	3,39,779	4,85,855	3,53,683	2,45,852
d1-f3	d6-c5	0.12.41	0.016181	11,29,705	22,18,876	2,01,930	1,19,815
b1-c3	d7-d6	0.011921	0.028229	15,41,251	38,13,630	3,69,283	1,58,496
g3-h2	d8-h4	0.007755	0.49.31	9,50,458	47,18,602	4,31,837	96,570
b2-b4	c5-d4	0.51.51	0.014757	44,47,001	17,82,210	1,78,761	4,38,498
c1-b2	h5-f4	0.51.25	1.08.00	49,84,948	62,73,266	6,10,099	4,83,596
a1-e1	a6-a5	0.078727	0.167083	1,08,07,657	2,22,14,772	22,17,027	10,69,545
b4xa5	a8xa5	0.03485	0.088646	46,67,748	1,16,46,847	11,66,199	4,56,105
h2-h1	d4xc3	0.030012	0.055104	39,64,597	71,51,202	7,46,377	4,23,911
b2xc3	a5-a3	0.05.48	0.005162	4,77,992	5,93,873	74,412	61,489
a2-b3	f4xd3	0.07.25	0.10.44	6,65,984	9,97,150	1,15,033	82,843
e1-e3	d3-f4	0.006586	0.010625	8,56,418	14,38,775	1,45,665	89,376
c3-b2	a3-a5	0.09.56	0.21.25	8,91,364	19,88,336	2,02,731	93,377
b2-c3	a5-a6	0.013275	0.26.35	18,39,570	24,35,221	2,45,140	1,76,240

Table 1: Table produced from the two runs of the chess program, HAL, both with and without optimization.

A line graph illustrating a comparison of the performance metrics and variance produced by quiescence optimization as compared to the performance metrics without quiescence optimization are offered in Figures 4 & 5 below.

In both line graphs below, a solid line is with optimization and a dashed line is without optimization.

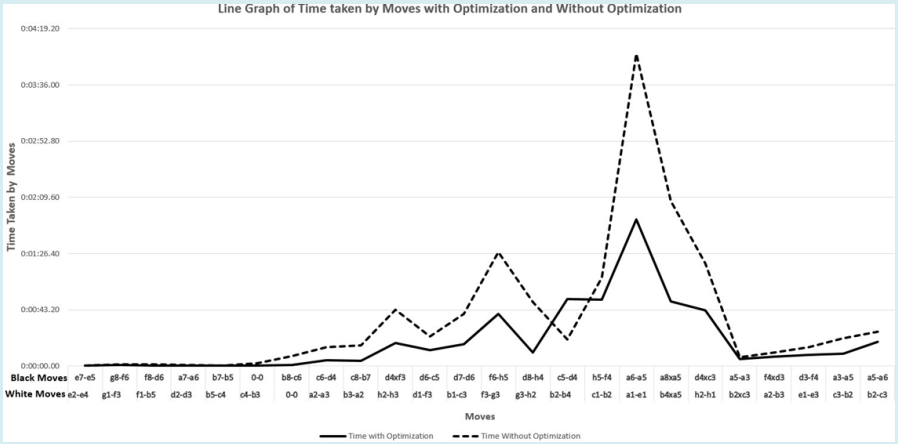

Figure 4: Line Graph of the Elapsed Move Times with Optimization and without Optimization.

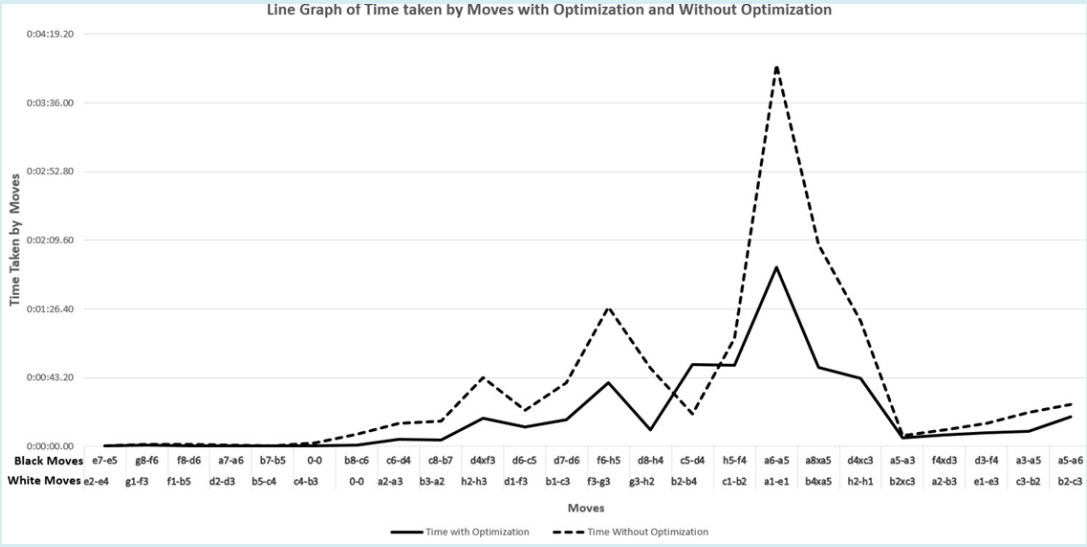

Figure 5: Line Graph for the Computational Effort of moves with Optimization and without Optimization.

The Hypothesis

The Null Hypothesis

H0: Trying a move that will not cause a singular extension of the game-tree before a move that will cause a singular extension will not demonstrate a statistically significant improvement in performance of the quiescence phase of α-β search.

The Alternative Hypothesis

Ha: Trying a move that will not cause a singular extension of the game-tree before a move that will cause a singular extension will demonstrate a statistically significant improvement in performance of the quiescence phase of α-β search.

The Hypothesis Test

The hypothesis test was performed with a standard T-Test as applied to the output metrics produced by the chess program for both the quiescence alpha-beta search with optimization and quiescence alpha-beta search without optimization.

T-Test Results

With a confidence interval of 95% and an α (alpha) of 0.05, for both a two-tailed and a one-tailed t-test of the standard distribution of the data points, the P value of the t-test was less than α rejecting the aforementioned Null hypothesis in favour of the alternative hypothesis. Thus, this establishes the statistical significance of quiescence alpha-beta search with optimization over quiescence alpha-beta search without optimization. The t-test results follow below.

```
import numpy as np
import pandas as pd
from bio in fokit . analys import stat

df = pd . read _ csv ("chess _ data 1.csv ")
dframe = df [ ['Moves Without Optimization' , 'Moves with Optimization'] ]

res = stat ( )
res. ttest (df = dframe,
res = ['Moves Without Optimization', 'Moves with Optimization'] , test_ type = 3)
print (res . summary)
```

Code 4: Python program for T-Test Results.

The P value is less than 0.05 in both one-tail and two-tail t-test, which means we can reject the null hypothesis in favour of the alternative hypothesis (Table 2).

Paired	T-Test
Sample Size	24
Difference Mean	1.52E+06
t	2.63178
Df	23
P-value (one-tail)	0.007454
P-value (two-tail)	0.014908
Lower 95.0	326237
Upper 95.0	2.72E+06

Table 2: Hypothesis Test Results.

Conclusion

The findings of this research have been demonstrated by experimentation and empiricism. The modification to the quiescence extension of iterative-deepening alpha-beta aspiration search demonstrates a statistically significant performance improvement of deep game-tree search as established by this research for game-trees of the type and dimension as described in this research paper.

For researchers who wish to verify the results of the research reported in this paper, and for peer review of these findings, the author offers the Windows executable code (.exe) and the Pascal source code (.pas), with instructions for setting up the experiment to anyone requesting a copy. To request a copy of the experimental model, HAL, used in this research, send a request by email to swheeler@ai-scientist.com. The experimental model, HAL, will be returned to the requester, free of charge by Zipped file attachment in a reply email.

References

1. Neumann J, Morgenstern O (2007) Theory of games and economic behaviour. In Theory of games and economic behaviour. Princeton university press.

2. Shannon CE (1950) Xxii. Programming a computer for playing chess. The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science 41(314): 256-275.

3. Nilsson NJ (1971) Problem-solving methods in artificial intelligence. Stanford Research institute pp: 1-73.

4. Knuth DE, Moore RW (1975) An analysis of alpha-beta pruning. Artificial intelligence 6(4): 293-326.

5. Marsland TA (1987) Computer chess methods. Encyclopedia of Artificial Intelligence 1: 159-171.

6. Samuel AL (1959) Some studies in machine learning using the game of checkers. IBM Journal of research and development 44(1-2): 220-229.

7. Russell SJ (2010) Artificial intelligence a modern approach. Pearson Education.

8. Baier H (2016) A rollout-based search algorithm unifying mcts and alpha-beta. In Computer Games pp: 57-70.

9. Wheeler SF (2010) The HAL (Heuristic Associative Linear-algorithm) chess program. CPW.

10. Marsland TA, Campbell M (1982) Parallel search of strongly ordered game trees. ACM Computing Surveys 14(4): 533-551.

11. Berliner HJ (1974) Chess as problem solving: The development of a tactics analyser.

12. Marsland TA, Schaeffer J (1990) Computers, chess and cognition. CPW.

13. Frey PW (1984) Chess Skill in Man and Machine. CPW.

14. Levy DNL (1988) Computer Chess Compendium. CPW.

15. Wheeler SF (1987) A performance benchmarck of the alpha-beta procedure on randomly ordered non-uniform depth-first game-trees generated by a chess program. ICCA Journal 10(1): 1-43.

16. Newborn M (1975) Computer chess (A.C.M. monograph series). Academic Press.

17. Gillogly JJ (1972) The technology chess program. Artificial Intelligence 3: 145-163.