Maseeh College of Engineering and Computer Science

PORTLAND STATE UNIVERSITY

Artificial Intelligence

CS 441/541

Pawn Chess Optimization:
Empirical Comparison of Minimax Algorithms

Authors and Contributions:

Cordet Gula

Alpha-Beta Pruning
Background Research
Introduction
Proposal

Ryan Filgas

Minimax Algorithm
Initial Graph Code
Efficiency Improvements
Final Testing

Nicole Kurtz

Heuristics
Test Improvements
Testing

Amila Ferron

Program Architecture
Code Reviews
Debugging
Unit Tests

December 8, 2022

Contents

1 Introduction

Combinatorial games in computer science have been historically used to improve the efficiency of algorithms. The push to improve AI algorithms occurred after the first AI Winter in 1966 when research funding was halted [1], [3]. In 1973, the Lighthill Report mentions the 'combinatorial explosion' as the root cause of the AI Winter. The failure to achieve expectations was because current methods were insufficient to solve problems with significant time and space complexity [1]-[2]. In an effort to solve intractable problems, new and more efficient algorithms needed to be developed.

Many problems to date have been solved due to the development of improved algorithms. Despite the success of today's algorithms, some combinatorial games remain practically intractable because of the state space required to solve the problem. Although it is mathematically solvable, chess is considered an intractable combinatorial game despite the deterministic environment because the state space is roughly 10^{43} with approximately 10^{120} variations of moves from the initial state and cannot be solved by a polynomial-time algorithm, also known as an NP-Hard problem [4]-[8].

At best, a generalized n x n chess game can be completed in exponential time. NP-Hard problems such as these, are often approached with approximation algorithms [9]. To study the efficiency of different algorithms, sub-games of chess, such as n-queens, n-rooks, and n-pawns are often used [10] in addition to the Minimax algorithm because it limits the game tree to a certain depth and utilizes optimization properties [11]. Minimax used on NP-Hard problems can reduce the complexity of some games to polynomial time [12], but cannot reduce the complexity of some games such as chess which requires exponential-time [13]-[14] and can only reach PSPACE-completeness or PSPACE-hardness given a generalized chess board that contains at least one king, queen, and a polynomial number of pawns relative to the board size [15]. Even a solo chess game – only a single color player – is NP-Complete only if there is one type of piece on the board and there exists multiple restrictions such as x-type piece can only capture a maximum of two other pieces [16].

Designed for zero-sum games, Minimax is used as a popular algorithm for playing chess. Due to the exponential search space, variations of Minimax have been developed to increase the efficiency of large state space games. Some examples include best-first Minimax, evaluation functions, heuristics, and Alpha-Beta Pruning [17]-[20]. Heuristics are beneficial because they allow for better problem-solving by defining a value of a certain state [21]. Alpha-Beta Pruning on the other hand, adds another beneficial component to the Minimax algorithm by reducing the game-tree complexity and speeding up computation time without losing valuable information.

In Alpha-Beta Pruning, there is a search cutoff that reduces the search to a branching factor of up to $b^{d/2}$ as opposed to b^d for Minimax [21], [22]. Deep Blue, a famous chess machine, was state of the art because it not only defeated the best chess player in the world, it could search up to 40 billion positions for each move by using Minimax with Alpha-Beta Pruning along with evaluation functions [23]-[24]. Interested in studying the efficiency of different algorithms in chess, we chose to implement pawn chess with Minimax, heuristics, and Alpha-Beta Pruning with both a random baseline opponent and a deterministic opponent. The purpose of our study is to test the effect heuristics has on proficiency and compare computation speed between generalized Minimax and Minimax with Alpha-Beta Pruning.

2 Methods

In order to limit complexity we chose to create an AI model to play the game of pawn chess, in which only the pawns of a regular chess game are placed in their usual starting positions on a chess board and the goal is to be the first to reach the other side, when promotion would occur during a standard game. Legal moves are the same as in a standard chess game, with captures made one diagonal square ahead. AI agents implement improvements for speed and performance and were pitted against one another for evaluation. A game class keeps track of the state of the game and the current board is passed to each player along with a request for their next move when it is their turn. Decision time per player is measured by the processing time (not the wall clock time) that player takes between the start of a call to select a move and the return from that call. During testing, pairs of opponents played against each other for several games, alternating which player started each game, and the results were recorded and plotted. A common baseline player opposed sets of similar players to test their configurations. Opponents ranged from simple to complex, starting with a random algorithm, replacing that with the Minimax algorithm, adding heuristics, and finally including Alpha Beta pruning.[1]

2.1 Minimax Algorithm

The complexity of Minimax can be roughly estimated using the branching factor(b) to the number of plies(d). There were an average of 31 moves per game given two random players over 10,000 runs in testing with a rough branching factor of 8 moves. The complexity then is in the neighborhood of 8^{31} states to explore given most pawn moves will be forward and 1 in 8 can move at a time. Compared to traditional chess around 10^{123} [28], this number is small.

Our implementation of the Minimax algorithm is broken up into four major functions in addition to a heuristics evaluation function discussed later and an alpha beta pruning function:

- A wrapper function initializes the root node, sets the depth and initiates a recursive call to the pre-order function.
- A pre-order searching Minimax function recursively traverses a state tree where each node is a different possible move followed by the possible moves after that etc. until a terminal state is reached. When a terminal state is reached the heuristic score as determined by an evaluation function is returned through a node object. If it's the maximizers turn it returns the greatest of its children, and the minimizer returns the minimum. Lastly when the root of the tree is reached the move that led to the max if the tree is returned, and this process will start over for the next player.
- An unpacking function gets the possible moves for each player and converts each to a node returning a list of nodes for the pre-order search to call itself with as an argument to keep searching the tree.

[1] Code is available at https://github.com/aferron/AIChessPlayer.

- Lastly a function checks whether a terminal state has been reached: if black wins, white wins, or if the set depth has been reached. Either player winning returns a set value and a depth limit evaluates a heuristic function to return a score for the move, otherwise nothing is returned and the recursive calls continue.

2.2 Alpha-Beta Pruning Algorithm

In an effort to optimize the Minimax algorithm, the Alpha-Beta Pruning algorithm was implemented for pawn chess. Given that Alpha-Beta Pruning was activated, once Minimax reached a maximum depth *d*, Minimax would begin backtracking. For each child in the unpacked array of children, Alpha-Beta Pruning would take in the current node, the maximizer as a chess color, and the child node. Alpha-Beta Pruning would begin evaluating from the leaf node and evaluate the maximizer or minimizer nodes during the backtracking process. During gameplay, each node created within the search tree had an alpha and beta value initialized to *alpha = -inf* and *beta = inf*.

Since a termination state returns the earned reward from the leaf node, Alpha-Beta Pruning passes in the parent node and the child node. From the perspective of the child node, if the parent state is at a maximizer level, then the ancestor is the minimizer and vice versa. If the parent node were the maximizer and there is an ancestor, then the parent node's alpha value would be compared to the ancestor node's beta value. If the alpha value was greater than or equal to the beta value, then beta pruning would occur. Additionally, if the parent were the minimizer and has its own parent, then alpha pruning would occur if the parent's beta value is less than or equal to the ancestor's alpha value. When pruning occurs upon finding a boundary violation, the Alpha-Beta Pruning algorithm returns true and the search in the current subtree is terminated.

If pruning does not occur and the parent is a maximizer, the parent's alpha is set to the maximum value of either the current alpha value or the child's reward value. If the parent is the minimizer, then the parent's beta value is set to the minimum of either the current beta value or the child's reward value. If no pruning occurs, then Alpha-Beta Pruning returns false and the Minimax search continues. The process repeats until the backtracking hits a termination state. The goal for implementing Alpha-Beta Pruning in addition to the Minimax algorithm is to find the same path as Minimax but with a fraction of the computation time.

2.3 Heuristics

A heuristic function is a means of informing the search function to select board states that will bring the player closer to the goal state. After researching pawn chess strategy, six means of evaluating a board state were determined. The heuristics are as follows: Maximize number of pieces on the board, keep pawns diagonally supported, place pawns side-by-side, move towards the opposite side of the board, avoid stacking pawns, avoid moving piece into locations that could be immediately captured.

Heuristic 0, maximize number of pieces on the board, was developed based on the idea that having more pieces on the board means a player is able to control more squares and thus poses a greater threat to the opponent. This is especially true if a player is able to

outnumber their opponents pieces. This heuristic measures the number of pawns of the player minus the number of pawns of the opponent.

Heuristic 1, keep pawns diagonally supported, protects pawns from opponents. Pawns capture opponents diagonally, thus if a pawn that is diagonally supported is captured, the opponent will be captured, as well. It important that before a player captures an opponent, that it has more attackers then there are defenders. By diagonally supporting pawns, it prevents this from being true for the opponent. This is calculated by counting the number of diagonal supports per pawn.

Heuristic 2, place pawns side-by-side. This heuristic allows for pawns to control a wider range of the board. Two pawns placed side-by-side control the four squares in front of the pawns. More control of the board prevents the opponents pieces from progressing. This is calculated by counting the number of side-by-side pawn pairs.

Heuristic 3, move towards opposite side of the board, is based on the ultimate task of pawn chess which is to convert your pawn into a queen by moving your pawn to your opponents back row. This is calculated by counting how far a player is from its starting location.

Heuristic 4, avoid stacking pawns, prevents a player from inhibiting a pawn movement by having two pawns in the same column. Each stacked pawn results in subtracting one to determine the heuristic value.

Finally, Heuristic 5, avoid moving pieces into locations that could result in an immediate capture, checks to make sure that a move wouldn't enter a square that is being attacked by an opponent. This helps ensure that a player is not unnecessarily losing pieces. If a board state would result in a player's piece being in jeopardy, 20 points are deducted from its heuristic value. This large value is due to the fact that losing a pawn in pawn chess can cost you the entire game.

3 Results

3.1 Random Agent Vs. Random Agent

As a control, two random agents are played against each other, giving expected results showing that they evenly split the number of wins as seen in Figure 1. The time to select the next move is trivial for a random agent.

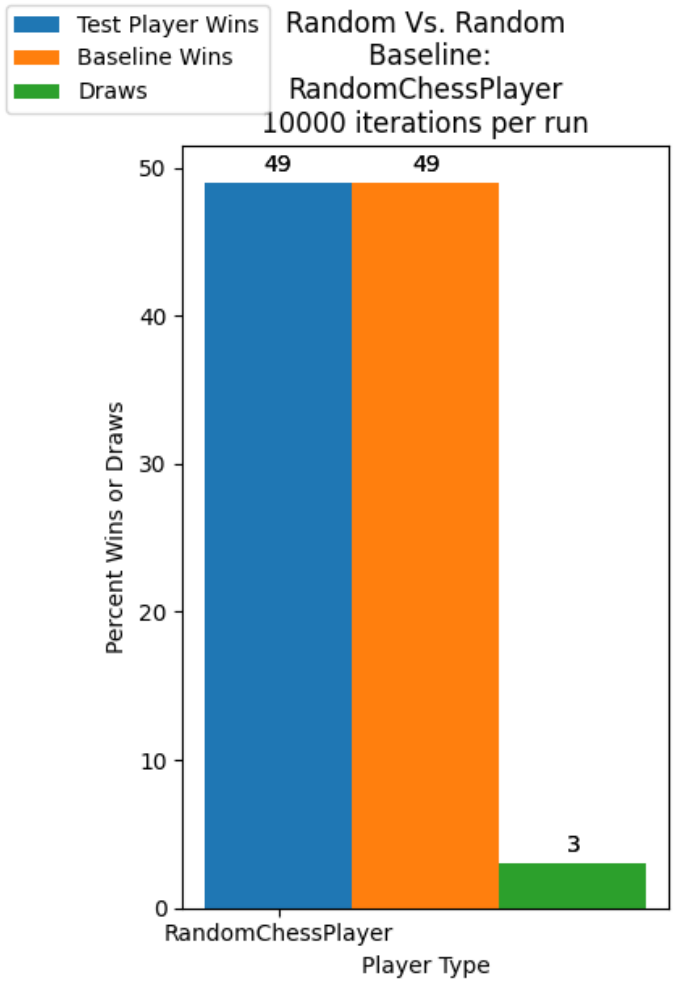


Figure 3.1.1: Results for a random agent versus random agent matchup.

3.2 Random Agent vs. Minimax Agent

Two Minimax agents searching to 1 and 2 moves ahead were each played against a random algorithm baseline agent. Their results in Figure 2 show significant improvement over the random player even at a depth of 1 and increasing improvement with a depth of 2. As would be expected the time to select the next move increases dramatically from a depth of 1 to a depth of 2. Empirically, we can say that the maximum practical depth that we are able to process on our machines is about 4. Average decision time per Minimax player is shown in Figure 3.

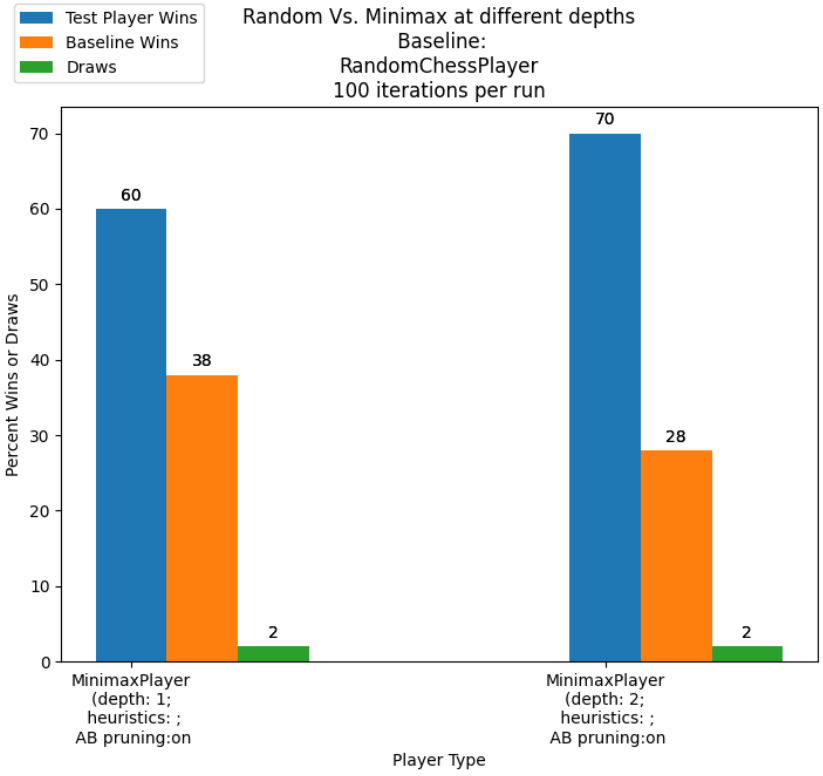


Figure 3.2.1: Minimax agent win results at depths 1 (left) and 2 (right).

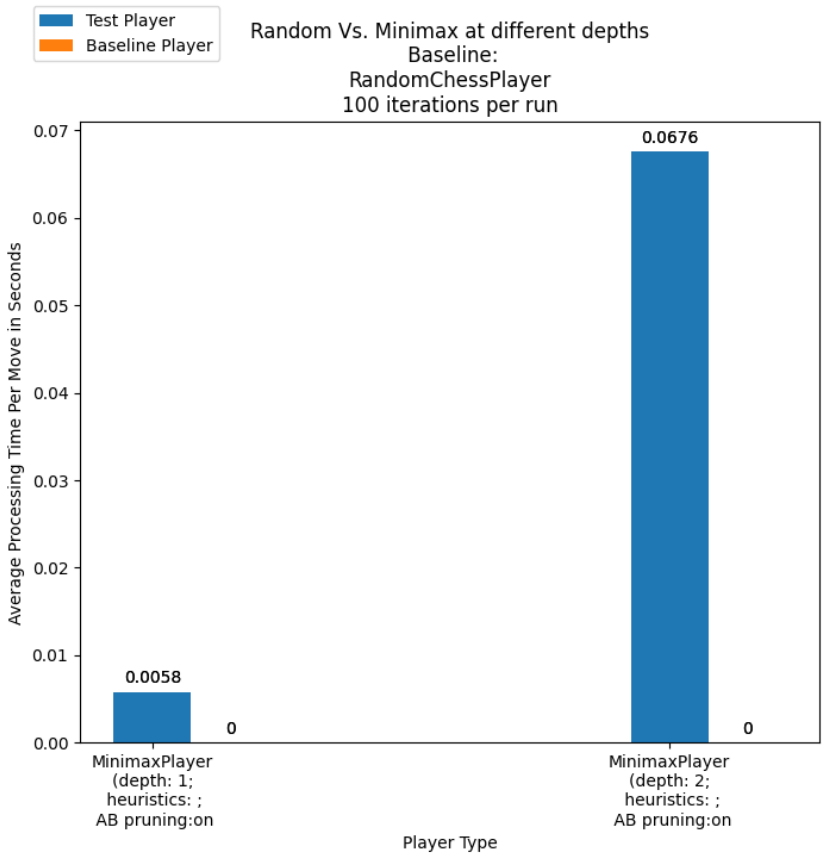


Figure 3.2.2: Minimax agent time results at depths 1 (left) and 2 (right).

3.3 Multi Agent Speed Test

Four timed tests against a random baseline were run to compare speed differences between the Minimax algorithm with and without Alpha Beta Pruning and heuristics. Alpha Beta Pruning ran approximately 13.1 times faster compared to the classic Minimax algorithm in no-heuristic tests. Adding all heuristics made the processing time around 2.5 times slower for both variations of Minimax. When adding heuristics to both Minimax and Minimax with AB pruning the difference increased with AB Pruning being about 15x faster.

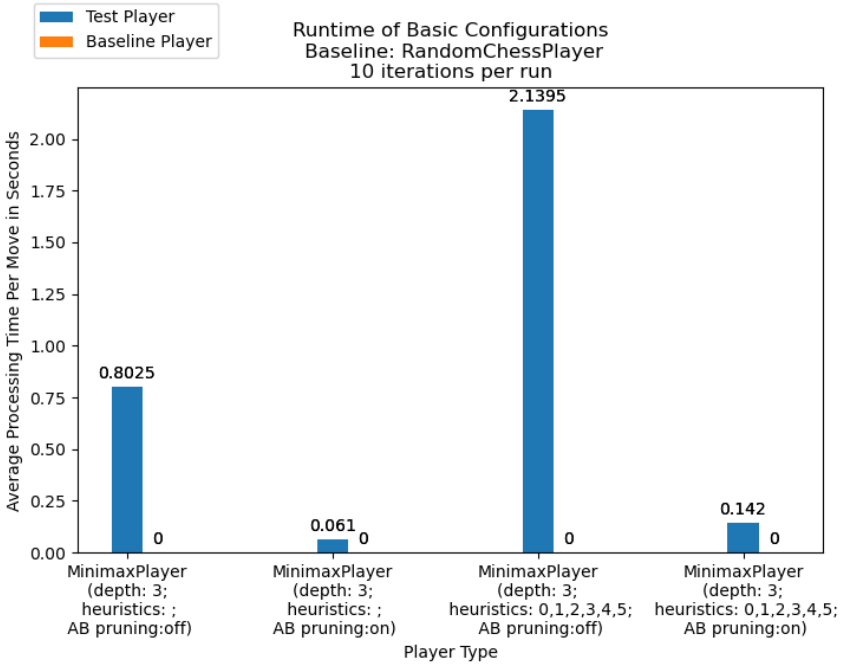


Figure 3.3.1: Random Baseline vs Minimax, Minimax w/ AB Pruning, Minimax w/heuristics and Minimax w/heuristics + AB pruning

3.4 Minimax Agent and Heuristics Ablation Study

Several tests to investigate the removal of heuristics to measure the impact on game and time performance of a chess player.

Figure 3.4.1, demonstrates that use of all heuristics makes for a significantly higher win rate then any single heuristic. Curiously, Heuristic 0, which maximizes number of chess pieces on the board, is only slightly worse then using all heuristics. Through this and the Heuristic vs Heuristic study above (section 3.3), Heuristic 0 has proven to be the most effective. The removal of any one of Heuristics 1 - 5 from an opponent with all heuristics, demonstrated only a small decrease in opponent win success.

When examining how heuristics impact average processing time per move, using all heuristics resulted in nearly a double increase in processing time compared to use of a single heuristic. A single heuristic took about .3 seconds while all heuristics took roughly .48 seconds. The removal of any heuristic resulted in a reduction of processing time by about 12% (Figure 3.4.3).

In Figure 3.4.4, the impact of adding heuristics on processing time is apparent with a 46% increase when using five heuristics to calculate heuristic value. Although the increased use of heuristics results in improved win rate, it has the downside of significant application slow down.

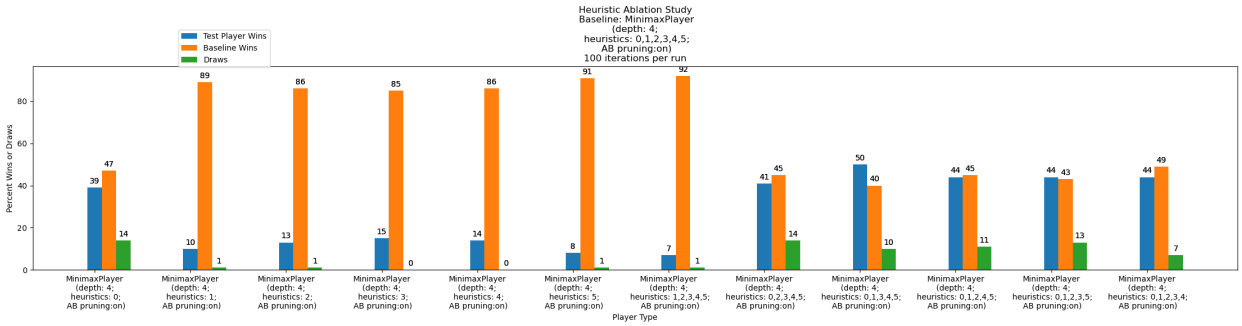


Figure 3.4.1: Ablation study showing the success of heuristic 0

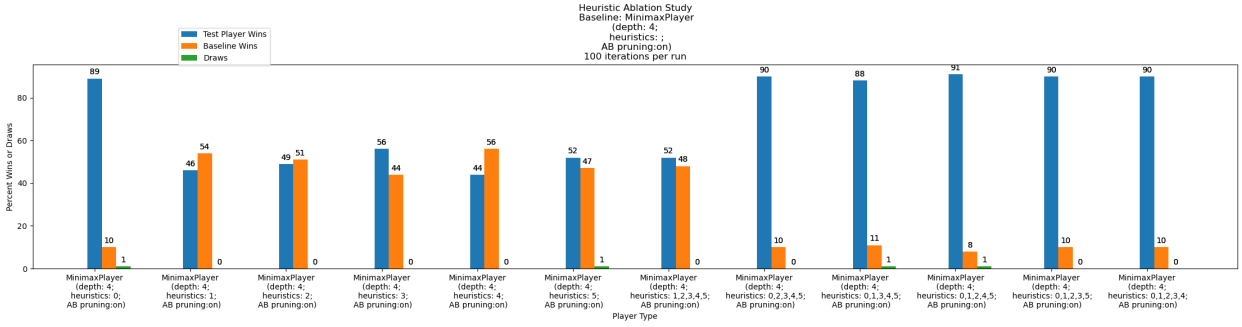


Figure 3.4.2: Minimax without heuristics against Minimax opponent with heuristics

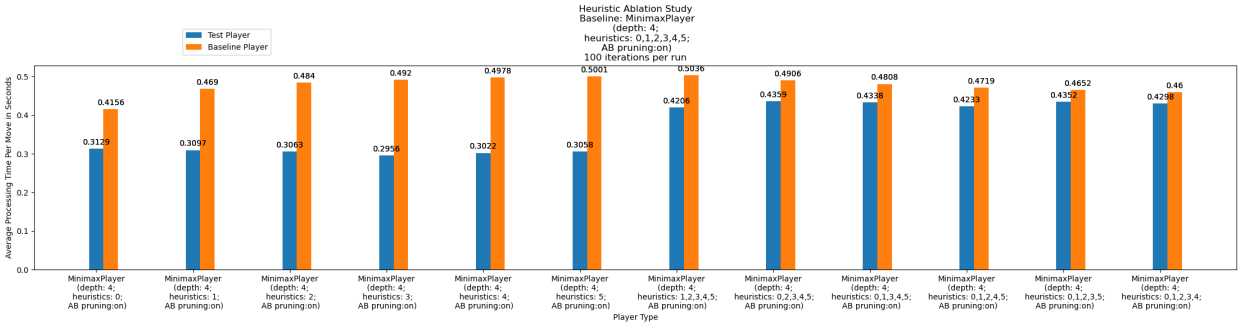


Figure 3.4.3: Speed test comparison of Minimax player with all heuristics against opponents with some heuristics

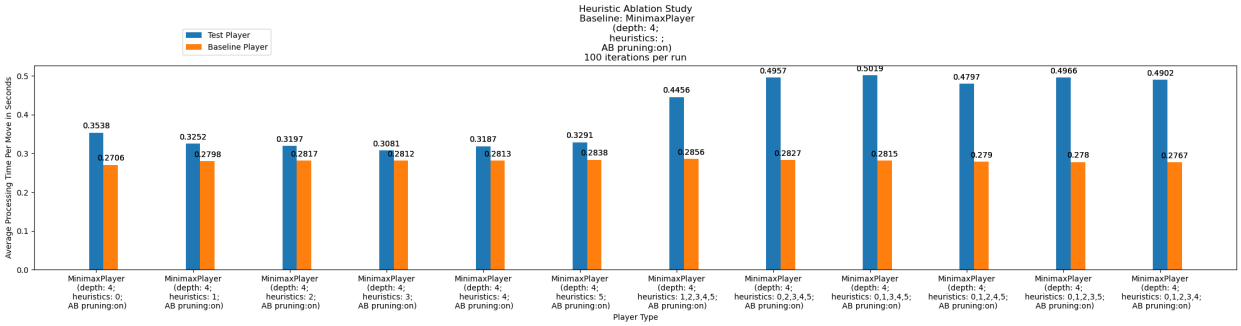


Figure 3.4.4: Speed test comparison of Minimax player with no heuristics against opponents with some heuristics

3.5 Heuristic vs. Heuristic

To obtain a better idea of how effective each heuristic is at evaluating a board state, a Minimax player using only one of the heuristics played against Minimax opponents with each of the other heuristics. Overall, the heuristics performed similarly to one another (see Appendix 4.4 for all charts), each winning roughly 50% of the time, with the exception of heuristic 0, maximize number of pieces.

Maximize number of pieces won more than 80% of games against all other heuristics, as shown in Figure 3.5.1. This heuristic is likely very powerful because if a player loses a piece in pawn chess, they are allowing an open column for an opponent to win the game.

Additionally, the average processing time per move in seconds was plotted for each heuristic (Figure 3.5.2). All heuristics performed similarly, taking approximately 0.29 seconds, except heuristic 0 which only took 0.24 seconds. All other heuristic required looping through a matrix to determine the value, while heuristic 0 was simply hitting the chess library API to obtain the number of pawns per player. Heuristic 0 was not only effective, it was fast.

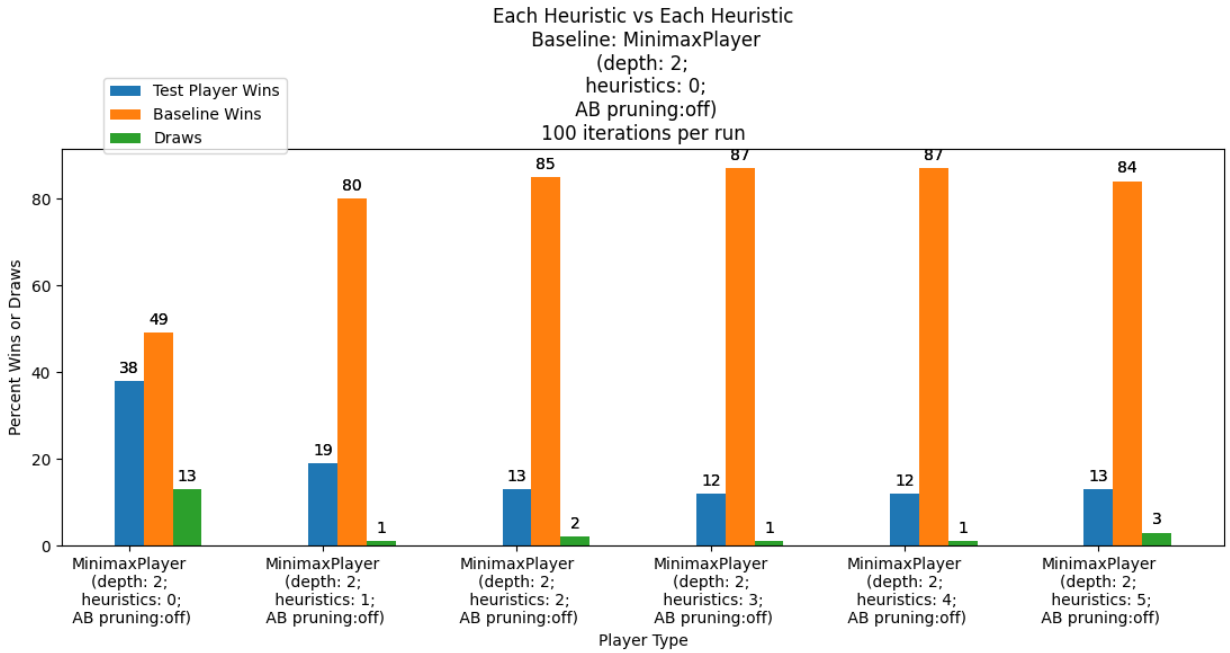


Figure 3.5.1: Minimax with Heuristic 0 against Minimax opponents with a single Heuristic

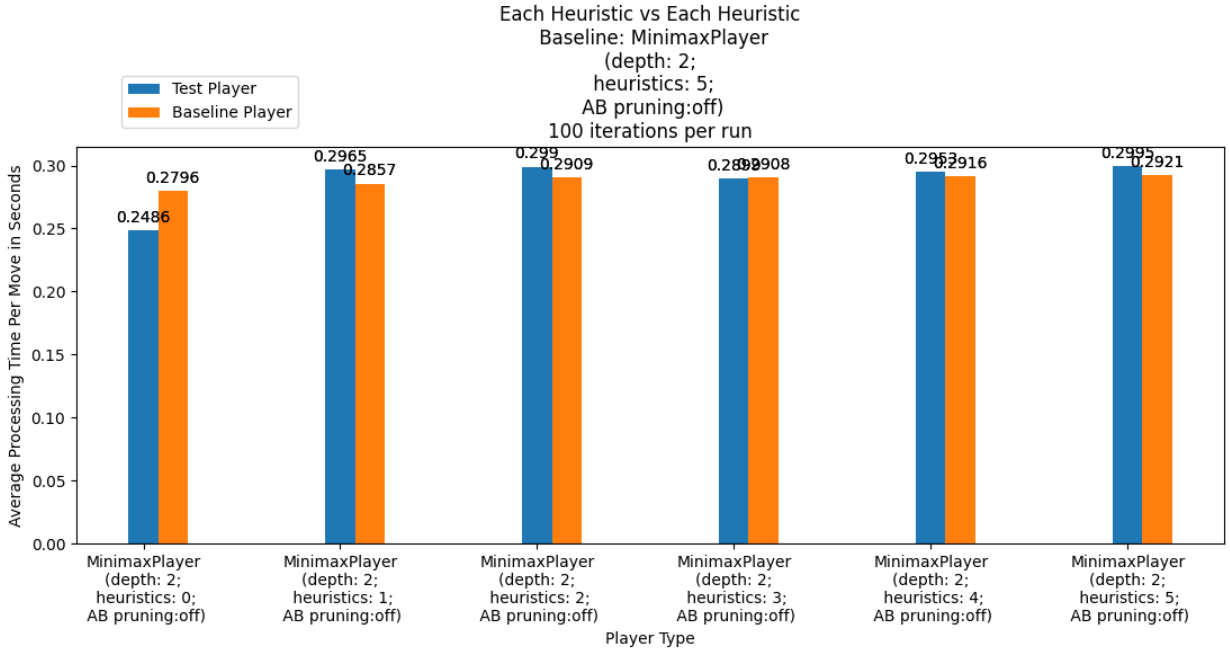


Figure 3.5.2: Average processing time of each heuristic

3.6 Deterministic Agent vs. Alpha-Beta Pruning Agent

To test the consistency of Alpha-Beta Pruning, a deterministic agent was implemented and used as a baseline. The deterministic agent was defined as an agent that picks the first move in a list of moves every time. Figure 3.6.1 shows the deterministic agent played against another deterministic baseline opponent had approximately 50% chance of winning. Without heuristics, Minimax and Minimax with Alpha-Beta Pruning wins performed about the same with a difference in wins equal to ±1 where Alpha-Beta Pruning performed in a fraction of the computation time as Minimax (see Figure 3.3.1). Minimax and Minimax with Alpha-Beta Pruning with heuristics showed approximately the same increase in the number of wins equal to ±8. Results for agents with and without Alpha-Beta Pruning confirm the pruning is effective and maintains performance as expected.

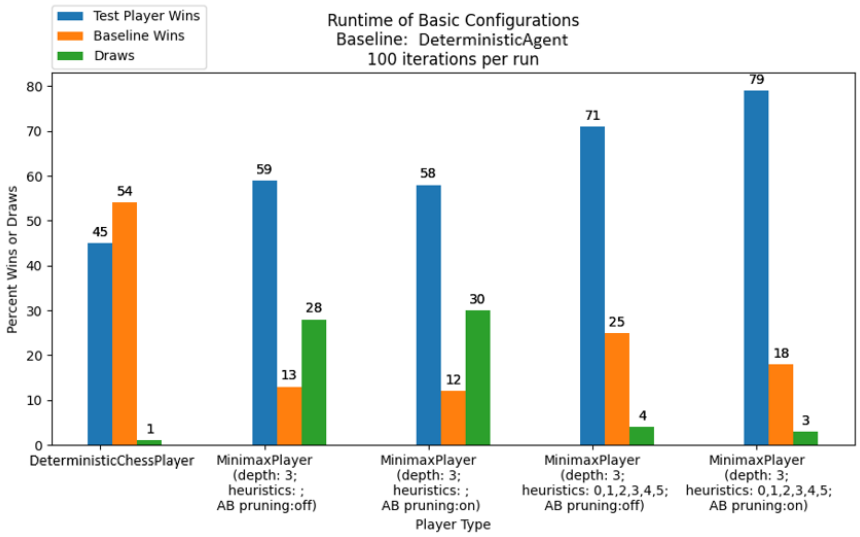


Figure 3.6.1: Results with a deterministic baseline opponent against Minimax, Minimax with Alpha-Beta Pruning, and both with and without heuristics for depth three.

3.7 Random Agent vs. Alpha-Beta Pruning Agent

A series of tests compared results against a random opponent for Minimax with and without Alpha-Beta Pruning. When the random baseline opponent played against a random agent, the win chance was approximately 49%. When Minimax was run without heuristics, Minimax and Alpha-Beta Pruning showed a difference in wins equal to ±4. Running Minimax and Alpha-Beta Pruning with heuristics against random baseline agent showed a difference in wins equal to ±1. The results show consistent win records with reduced computation time and confirms that pruning maintains performance while improving speed (see Figure 3.3.1). Compared to the deterministic baseline agent, playing pawn chess against the random baseline agent showed more variability with each test that was run.

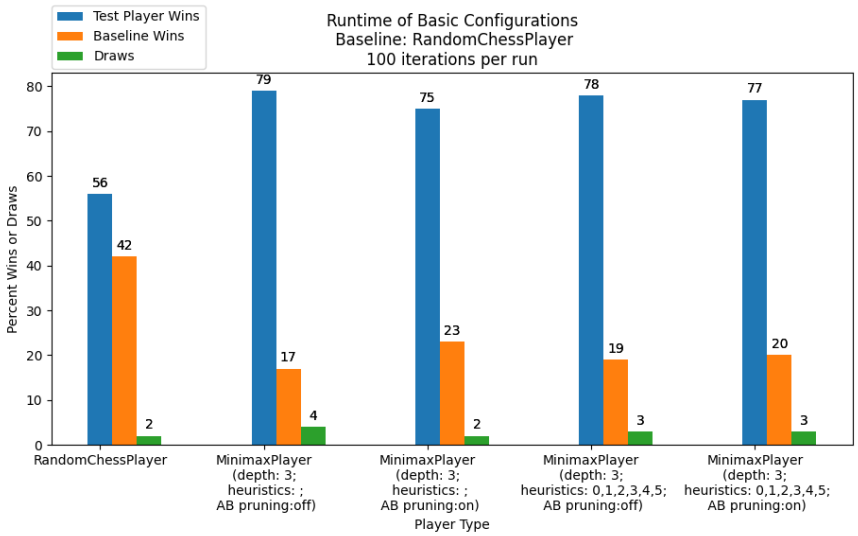


Figure 3.7.1: Results with a random baseline opponent against Minimax, Minimax with Alpha-Beta Pruning, and both with and without heuristics for depth three.

3.8 Minimax Agent with Heuristics and Alpha-Beta Pruning at Various Depths

A series of tests evaluated the performance of a Minimax agent using all heuristics against a set of Minimax agents with depths from 1 to 4, inclusive. The results show the player with greater depth consistently winning more games against a player with a shallower depth. When two players with the same depths face off the results are closer to even. Results for a baseline with depth 4 are in Figure 4. Results for baselines with depths 1 - 3 are in the Appendix. Time to compute increases from 0.005 at a depth of 1 to 0.44 seconds at a depth of 4 will all heuristics and Alpha-Beta Pruning as seen in Figure 5.

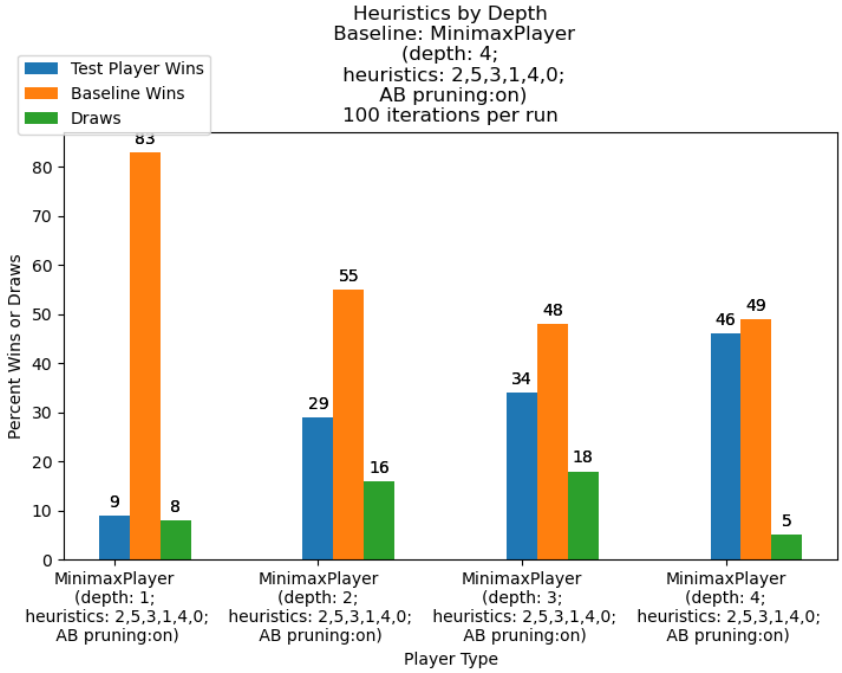


Figure 3.8.1: Win/Loss/Draw percentages for Minimax agent with all heuristics at depth 4 Vs. Minimax agents with all heuristics at depths 1 - 4.

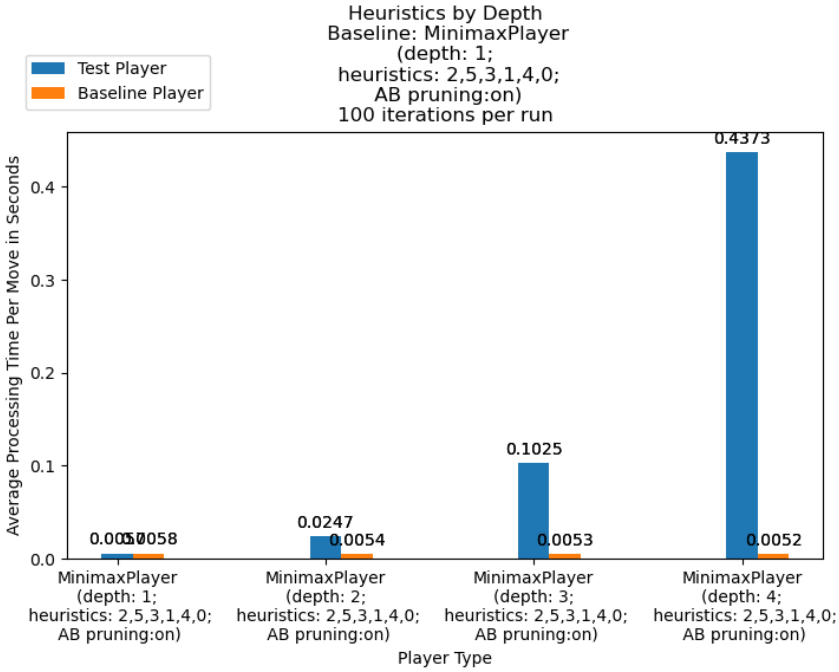


Figure 3.8.2: Time to compute for Minimax agent with all heuristics at depth 1 Vs. Minimax agents with all heuristics at depths 1 - 4.

4 Discussion

Overall, the Minimax algorithm with Alpha-Beta Pruning performed as expected. The Minimax algorithm markedly outperformed the random baseline and added heuristics outperformed the Minimax baseline. We found that maximizing the number of pieces on the board was the highest indicator of success when compared to the other heuristics tested. In addition to this, Alpha-Beta Pruning allowed the Minimax algorithm to reach higher depth settings in a shorter time frame which enabled further testing. Given the nature of combinatorial games such as pawn chess, we found that using a random agent showed results with more variability than a deterministic agent. Despite controlling for many variables, there was still a margin of error that can occur between each test. Although we have tested a variety of variables, there are still a myriad of different algorithms and tests that can be run with our implementation.

Future improvements to this project could include implementing Monte Carlo Tree Search and making more speed optimizations that leverage other parts of the chess library for game facilitation and heuristic evaluation. Additionally, the weighting of heuristics based on performance could result in more effective heuristic calculation when combined.

References

[1] S. Schuchmann, "History of the first AI Winter," towardsdatascience.com, May. 12, 2019. [online]. Available: https://towardsdatascience.com/.

[2] J. Lighthill, "A general survey by Sir James Lighthill FRS Lucasian Professor of Applied Mathematics Cambridge University," Cambridge University. July, 1972. Available: https://www.aiai.ed.ac.uk/events/lighthill1973/lighthill.pdf.

[3] A. Rhodes. CS 541. Class Lecture, Topic: "AI Winter." Maseeh College of Engineering and Computer Science, Portland State University, Portland, OR, Oct. 2022.

[4] R. V. Yampolskiy and A. EL-Barkouky, "Wisdom of artificial crowds algorithm for solving NP-hard problems," International Journal of Bio-Inspired Computation, vol. 3, no. 6, 14, November 2011. [online serial]. Available: http://cecs.louisville.edu/ry/ArtificialCrowds.pdf

[5] J. Erickson. Algorithms. Class Lecture, Topic: "NP-Hard Problems." Department of Computer Science, University of Illinois, Champaign, IL, 2010. [online document]. Available: https://courses.engr.illinois.edu/cs573/fa2010/notes/30-nphard.pdf

[6] R. M. Karp, "Reducibility Among Combinatorial Problems," 1972. [Online serial]. Available: https://cgi.di.uoa.gr/ sgk/teaching/grad/handouts/karp.pdf

[7] C. E. Shannon, "Programming a Computer for Playing Chess." Philosophical Magazine, vol. 41, no. 314, Mar., 1950. [online serial]. Available: http://archive.computerhistory.org.

[8] E. D. Demaine and Hearn, R. A. "Playing Games with Algorithms: Algorithmic Combinatorial Game Theory," Lecture Notes in Computer Science, vol. 2136. September 2001. [Online serial]. Available: https://arxiv.org/pdf/cs/0106019.pdf

[9] D. S. Johnson, "Approximation algorithms for combinatorial problems," In Proc. Association for Computing Machinery Symposium on Theory of Computing. 1973, pp 38-49. Available: https://dl.acm.org/doi/pdf/10.1145/800125.804034.

[10] N. D. Elkies, "On numbers and endgames: Combinatorial game theory in chess endgames," Games of No Chance, vol. 29. 1996. [Online serial]. Available: https://arxiv.org/pdf/math/9905198.pdf.

[11] E. Duchene, "Combinatorial games: from theoretical solving to AI algorithms," in HAL. [online document], 2018. Available: HAL Archives, https://hal.archives-ouvertes.fr/hal-01883569/document.

[12] D. Koller and N. Megiddo, "The complexity of two-person zero-sum games in extensive forms' experience," Games and Economic Behavior, vol. 4, no. 4, Oct, 1992. [Online serial]. Available: http://theory.stanford.edu/ megiddo/pdf/recall.pdf.

[13] Y. Shitov, "The Diameter of Chess Grows Exponentially," The American Mathematical Monthly, vol. 123, no. 1, January, 2016. [Online serial]. Available: https://www-jstororg.proxy.lib.pdx.edu

[14] A. S. Fraenkel and D. Lichtenstein, "Computing a perfect strategy for n×n chess requires time exponential in n," in Automata, Languages and Programming. vol. 115, May, 2005. [Online document]. Available: https://link.springer.com/content/pdf/10.1007/3-540-10843-2-23.pdf.

[15] J.A. Storer, "On the Complexity of Chess." Journal of Computer and System Sciences. vol. 27, pp. 77-100. 1982. [online document]. Available: https://reader.elsevier.com

[16] N. R. Aravind, N, Misra, and H. Mittal, "Chess is Hard even for a Single Player." Department if Computer Science and Engineering, Indian Institute of Technology, Gandhinagar, India, Mar. 30, 2022. [online document]. Available: https://arxiv.org/pdf/2203.14864.pdf.

[17] A. Rhodes. CS 541. Class Lecture, Topic: "Minimax Algorithm." Maseeh College of Engineering and Computer Science, Portland State University, Portland, OR, Nov. 2022.

[18] D. Nau, P. Purdom, and C. Tzeng, " Experiments on alternatives to Minimax." International Journal of Parallel Programming, vol. 15, pp. 163-183, 1983. [online serial]. Available: https://link.springer.com/article/10.1007/BF01414444.

[19] R. E. Korf and D. M. Chickering, "Best-first Minimax search," Artificial Intelligence, vol. 84, pp. 299-337, 1996. [online serial]. Available: https://core.ac.uk/download/pdf/82284884.pdf.

[20] B. W. Ballard, "The *-Minimax Search Procedure for Trees Containing Chance Nodes," Artificial Intelligence, vol. 21, pp. 327-350, 1983. [online serial]. Available: https://www.cs.uleth.ca

[21] S. Russell and N. Peter. Artificial Intelligence: A Modern Approach, 4th ed. Pearson Education: Hoboken, NJ, 2021. [E-book] Available: Kindle Edition.

[22] D. E. Knuth and R.W. Moore, "An Analysis of Alpha-Beta Pruning," Artificial Intelligence, vol. 6, pp. 293-326, 1975. [online serial]. Available: http://www-public.telecom-sudparis.eu

[23] M. Campbell, J. A. Hoane Jr., and F. Hsu, "Deep Blue," Artificial Intelligence, vol. 134, pp. 57-83, 2002. [online serial]. Available: https://core.ac.uk/download/pdf/82416379.pdf.

[24] F. Hsu, "IBM's Deep Blue Chess Grandmaster Chips," Micro IEEE, 1999. [online document]. Available: http://www.cse.msu.edu/ cse841/papers/DeepBlue.pdf.

[25] T. Preston-Werner, S. Chacon, C. Wanstrath, and P.J. Hyett, Github. [online software]. San Francisco, CA: Microsoft Corporation, 2008.

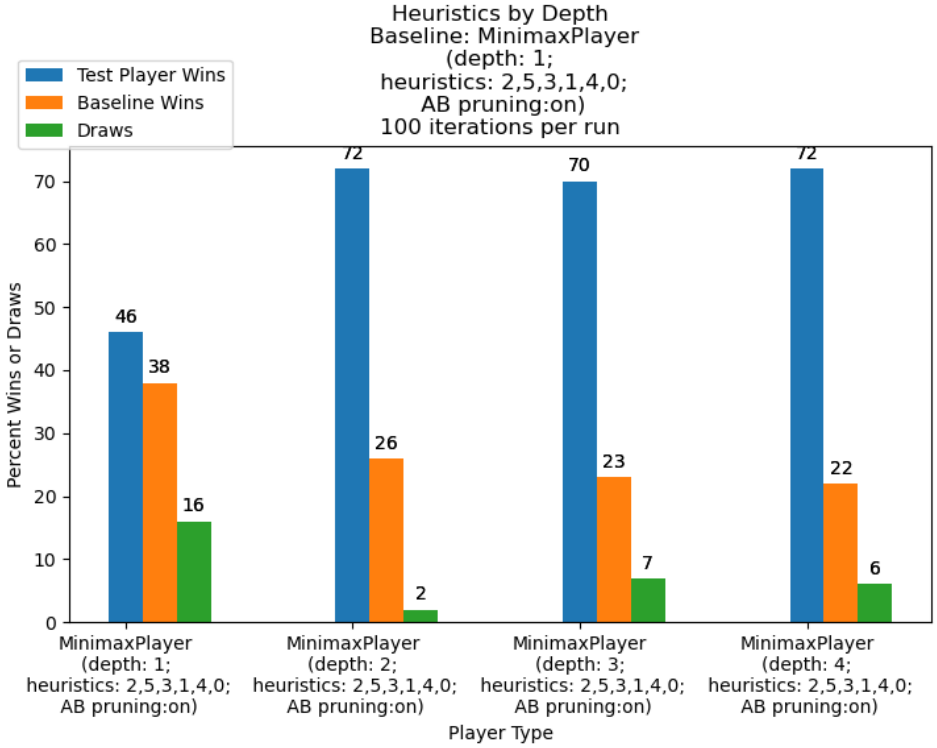
[26] G. V. Rossum, Python, [programming language]. Wilmington, DE: Python Software Foundation, 1991.

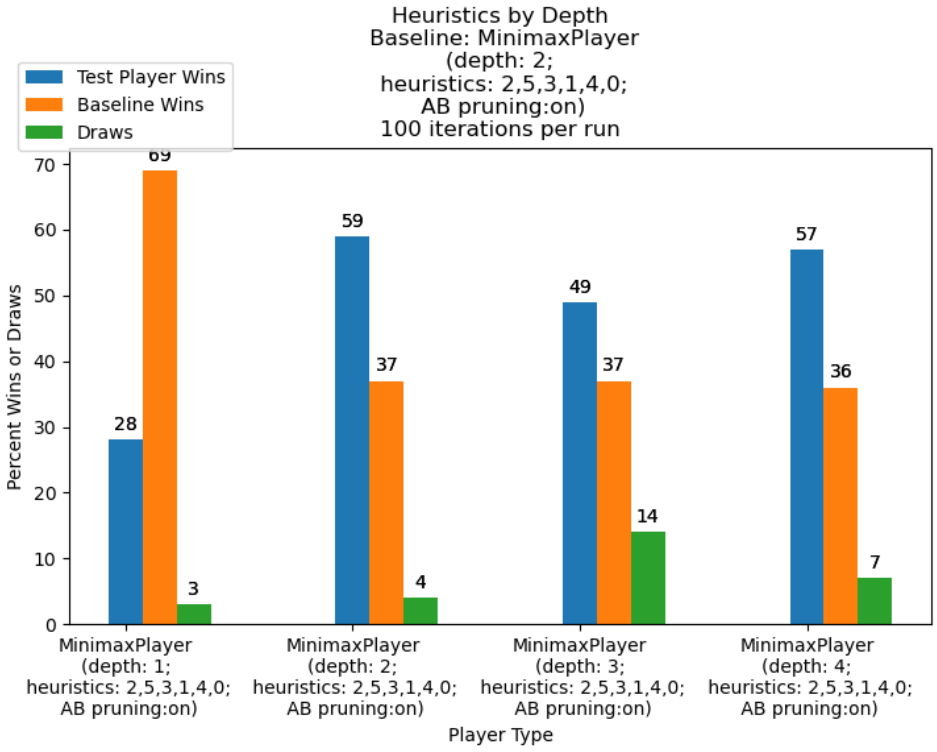
[27] Digital Science UK Limited, Overleaf, [online software]. Farringdon, London: Digital Science, 2014.

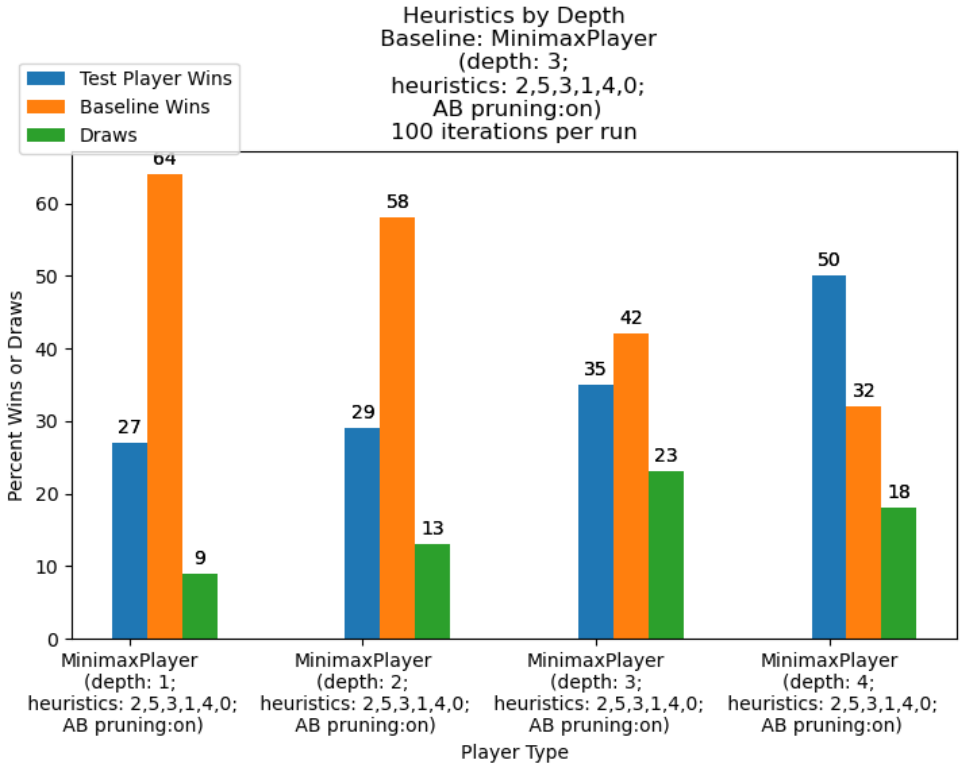
[28] Arshia Atashpendar, Tanja Schilling, & Thomas Voigtmann (2016). Sequencing chess. EPL (Europhysics Letters), 116(1), 10009.

Appendix

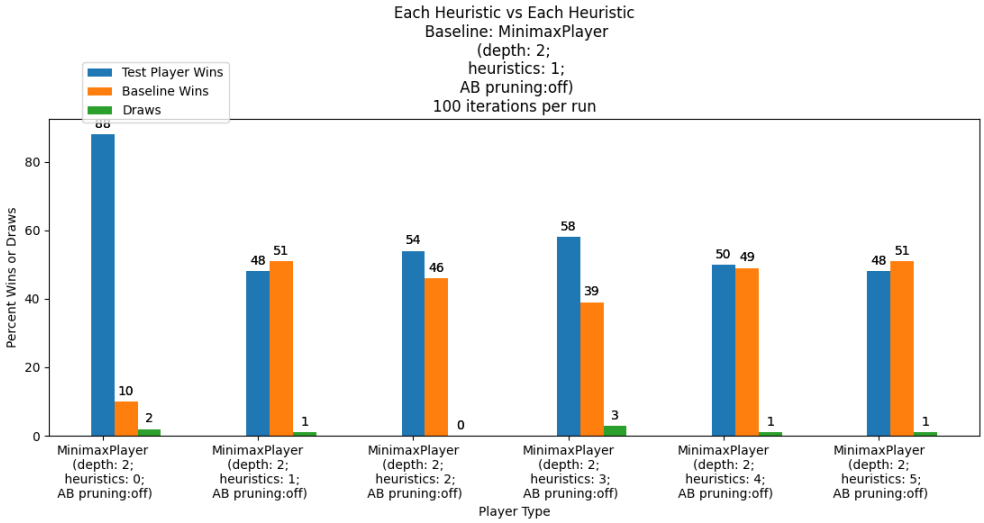
4.1 Results for Minimax Agents at Depths 1 - 4

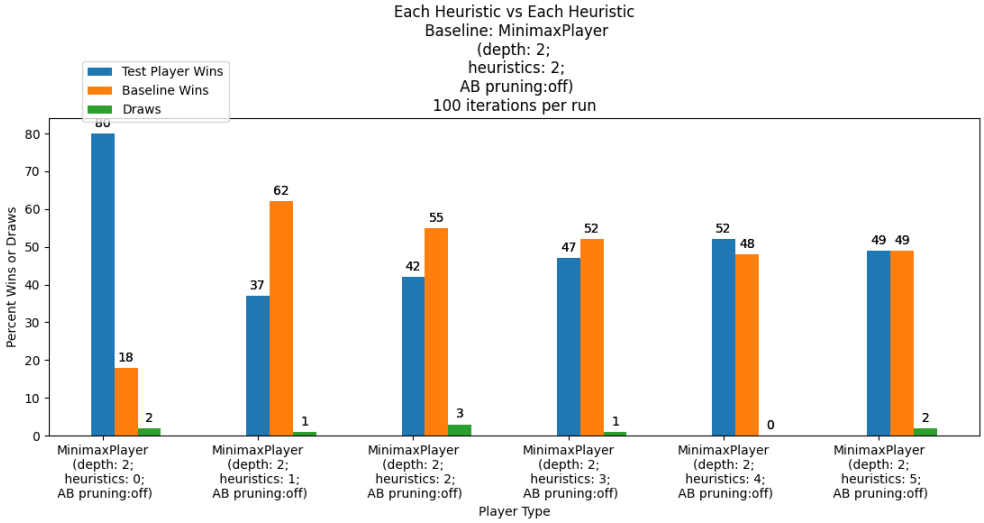


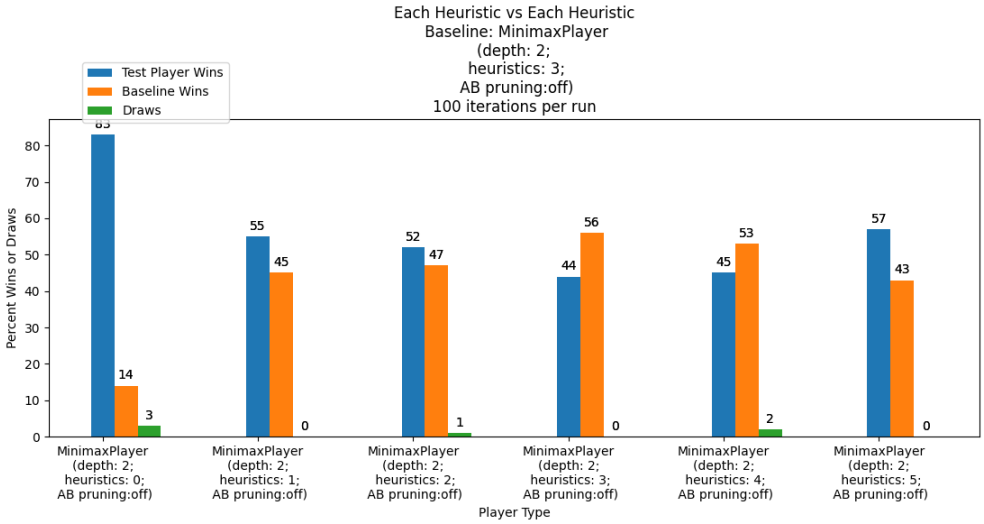


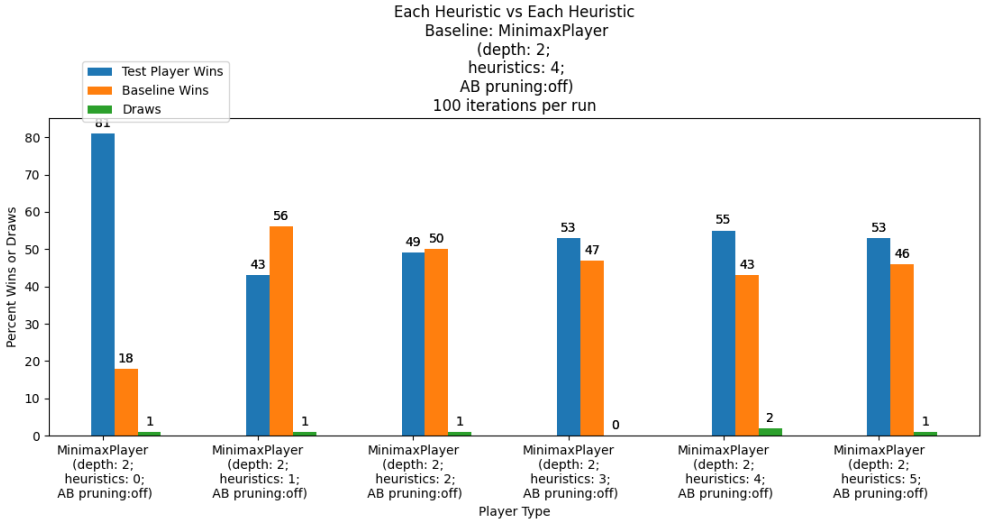

Heuristics by Depth
Baseline: MinimaxPlayer
(depth: 3;
heuristics: 2,5,3,1,4,0;
AB pruning:on)
100 iterations per run
Test Player Wins
Baseline Wins
Draws
Percent Wins or Draws
0
10
20
30
40
50
60
27
64
9
29
58
13
35
42
23
50
32
18
MinimaxPlayer (depth: 1; heuristics: 2,5,3,1,4,0; AB pruning:on)
MinimaxPlayer (depth: 2; heuristics: 2,5,3,1,4,0; AB pruning:on)
MinimaxPlayer (depth: 3; heuristics: 2,5,3,1,4,0; AB pruning:on)
MinimaxPlayer (depth: 4; heuristics: 2,5,3,1,4,0; AB pruning:on)
Player Type

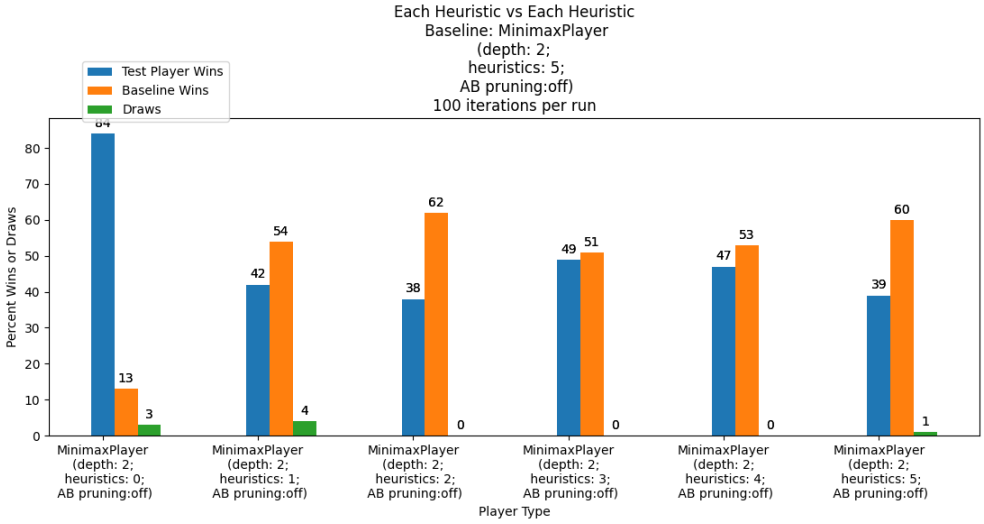
4.2 Results for Heuristic vs Heuristic








Each Heuristic vs Each Heuristic
Baseline: MinimaxPlayer
(depth: 2;
heuristics: 4;
AB pruning:off)
100 iterations per run
Test Player Wins
Baseline Wins
Draws
Percent Wins or Draws
81
18
1
43
56
1
49
50
1
53
47
0
55
43
2
53
46
1
MinimaxPlayer (depth: 2; heuristics: 0; AB pruning:off)
MinimaxPlayer (depth: 2; heuristics: 1; AB pruning:off)
MinimaxPlayer (depth: 2; heuristics: 2; AB pruning:off)
MinimaxPlayer (depth: 2; heuristics: 3; AB pruning:off)
MinimaxPlayer (depth: 2; heuristics: 4; AB pruning:off)
MinimaxPlayer (depth: 2; heuristics: 5; AB pruning:off)
Player Type


Each Heuristic vs Each Heuristic
Baseline: MinimaxPlayer
(depth: 2;
heuristics: 5;
AB pruning:off)
100 iterations per run
Test Player Wins
Baseline Wins
Draws
Percent Wins or Draws
84
13
3
42
54
4
38
62
0
49
51
0
47
53
0
39
60
1
MinimaxPlayer (depth: 2; heuristics: 0; AB pruning:off)
MinimaxPlayer (depth: 2; heuristics: 1; AB pruning:off)
MinimaxPlayer (depth: 2; heuristics: 2; AB pruning:off)
MinimaxPlayer (depth: 2; heuristics: 3; AB pruning:off)
MinimaxPlayer (depth: 2; heuristics: 4; AB pruning:off)
MinimaxPlayer (depth: 2; heuristics: 5; AB pruning:off)
Player Type