Characterizing the Nature of Probability-Based Proof Number Search A Case Study in the Othello and Connect Four GamesCharacterizing the Nature of Probability-Based Proof Number Search A Case Study in

by Anggina Primanita

Submission date: 05-Apr-2023 12:06PM (UTC+0700)
Submission ID: 2056346716
File name: Characterizing_the_Nature_of_Probability-Based_Proof_Number.pdf (632.51K)
Word count: 7658
Character count: 37818

information

MDPI

Article

Characterizing the Nature of Probability-Based Proof Number Search: A Case Study in the Othello and Connect Four Games

Anggina Primanita [1,†], Mohd Nor Akmal Khalid [2,3,*,†] and Hiroyuki Iida [2,3,†]

[1] Department of Informatics, Universitas Sriwijaya, Jl. Palembang-Prabumulih Raya KM 32, Inderalaya 30862, South Sumatera, Indonesia; s1820431@jaist.ac.jp
[2] School of Information Science, Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan; iida@jaist.ac.jp
[3] Research Center for Entertainment Science, Japan Advanced Institute of Science and Technology, 1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan
* Correspondence: akmal@jaist.ac.jp
† These authors contributed equally to this work.

Received: 13 March 2020; Accepted: 7 May 2020; Published: 13 May 2020

Abstract: Variants of best-first search algorithms and their expansions have continuously been introduced to solve challenging problems. The probability-based proof number search (PPNS) is a best-first search algorithm that can be used to solve positions in AND/OR game tree structures. It combines information from explored (based on winning status) and unexplored (through Monte Carlo simulation) nodes from a game tree using an indicator called the probability-based proof number (PPN). In this study, PPNS is employed to solve randomly generated positions in Connect Four and Othello, in which the results are compared with the two well-known best-first search algorithms (proof number search (PNS) and Monte Carlo proof number search). Adopting a simple improvement parameter in PPNS reduces the number of nodes that need to be explored by up to 57%. Moreover, further observation showed the varying importance of information from explored and unexplored nodes in which PPNS relies critically on the combination of such information in earlier stages of the Othello game. Discussion and insights from these findings are provided where the potential future works are briefly described.

Keywords: best-first search; probability-based proof number search; Connect Four; Othello

1. Introduction

A best-first search algorithm was initially introduced as a method to find the game-theoretic value using a specific technique to progress towards a game tree framework. One of the prominent best-first search algorithms in such a context is the proof number search (PNS) [1]. PNS utilizes two variables, proof and disproof numbers (*pn* and *dn* for short, respectively), as the search indicators to find the best possible options. The proof number of a node represents the smallest number of leaf nodes that have to be proven in order to prove that it is a win, while the disproof number represents the smallest number of leaf nodes that have to be disproved in order to prove that it is a loss, thus representing the difficulty in proving a node [2]. A node is then chosen to provide the best possible choice (called the most proving node (MPN)) and subsequently expanded for further search [3–5].

Finding a game-theoretic value has shown to be fruitful when employing PNS [6]. However, several drawbacks observed from its implementation include the usage of a large amount of memory space, overly long solutions, and the see-saw effect (The see-saw effect is observed when the search goes back and forth between several sub-trees, preventing it from progressing to a deeper branch) [3,4].

An improvement of PNS that aims to reduce the usage of computation resources is the depth-first proof number (df-pn) search [7], which turns PNS from a best-first search algorithm into a depth-first search by introducing iterative deepening. The df-pn expands less on the interior node and reduces the number of proof numbers and disproof numbers that have to be recomputed.

The Monte Carlo tree search (MCTS) is another prevalent best-first search algorithm that has been employed to solve games. First, proposed by [8], the MCTS framework consists of four steps: (1) selection, (2) expansion, (3) simulation, and (4) backpropagation. The algorithm starts with selecting the next action based on a stored value (selection). When it encounters a state that cannot be found in the tree, it expands the node (expansion). The node expansion is based on multiple, randomly simulated games (simulation or playout). The value is then stored and backpropagated to the root of the tree, where the algorithm continues to repeat the steps until the desired outcome is reached (backpropagation). MCTS utilizes simulation to gain information from unexplored nodes, and this was expanded by the introduction of upper confidence bound applied to trees (UCT) by [9]. A new best-first search algorithm derived from MCTS and UCT was then introduced by [10]. The algorithm's goal was to overcome the difficulty found in building heuristic knowledge for a non-terminal game state by employing stochastic simulations. Determining a game-playing strategy is effective when using multiple simulations. Its usage has become well known, especially in the field of Go, and in part led to the AlphaGo's wins against top grandmasters [11].

Although it has been proven that MCTS converges to minimax when evaluating the available moves [12], MCTS converges only in so-called "Monte Carlo Perfect" games [13]. By leveraging both strengths of PNS and MCTS, an improved game solver's quality is expected. An early example of such a combination is the Monte Carlo proof number search (MCPNS) [14]. The work proposed the best-first search algorithm by employing its MIN/SUM rules to backpropagate information coming from the simulation, while gaining access to node information based on its Monte Carlo infused proof number, called the Monte Carlo proof number (*pmc*). The MCPNS provides a better degree of flexibility while retaining its reliability. However, the drawback is that its MIN/SUM rules compute results from integer numbers, while the statistical results from the MCTS counterparts produce a real number, thus leading to a loss of information from the simulation (Such information loss is related to the minimization of accuracy in the floating-point arithmetic, where it deals with complicated formulae that suffer from substantial errors due to round-off (called numerical stability). This problem can be addressed by adopting extended precision or logarithm transform, where such information becomes numerically stable and can compute to full double precision. Hence, we would direct interested readers to [15] for further reading.

To conform to the requirement of processing real numbers derived from statistical probability, the probability-based proof number search (PPNS) [16] is used, which solves positions in a simulated balanced and unbalanced tree [16]. It applies the idea of "searching with probabilities" first suggested by [17] to draw better results from real numbers produced by Monte Carlo simulation in the playout step.

PPNS is proposed to address the need for processing real numbers that are derived from statistical information that of the product of Monte Carlo simulation. PPNS is independently developed but utilizes similar principles as product propagation (PP) [18]. The main difference between PPNS and PNS is that the PPNS indicator is derived from both explored and unexplored areas of the tree to obtain information on the most proving node (MPN) (Figure 1), whereas PNS utilizes only the information from the explored area of the tree [19].

Nevertheless, the quality of PPNS as a solver in an unbalanced and balanced game tree, especially in a real game environment, is limited. Connect Four and Othello are two-player perfect information games, which have distinct features for application of the PPNS. Connect Four can be won between 13-ply and 42-ply, as well as by sudden death moves, thus making its game tree highly varied, and its structure unbalanced. Meanwhile, Othello is a well-balanced game that requires a sequential decision to end the game and offers fairness to both sides of the players [20,21], thus making it a

good candidate for a real game with a balanced game tree. Connect Four is characterized as *P*-Space, while Othello is *P*-space complete [1]. All of these traits make these two games suitable for use as testbeds to characterize the nature of PPNS as well as its distinction from other game solvers. Therefore, the contribution of this study is twofold. Firstly, this study intends to determine the characteristics that make the performance of PPNS optimal in solving game positions. Secondly, such performance of the PPNS is demonstrated by adopting it in real two-player, zero-sum, perfect information games such as Othello and Connect Four.

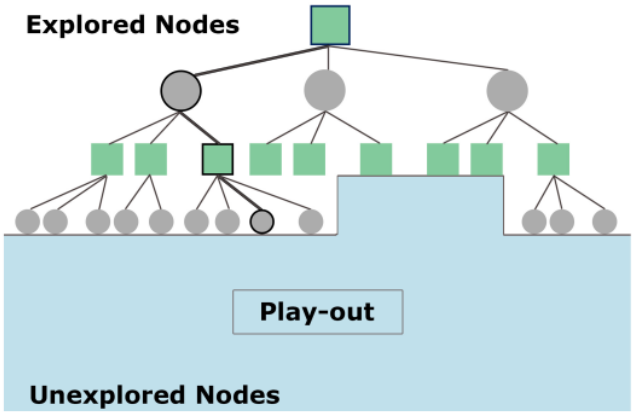

Figure 1. Explored and unexplored nodes in a MIN/SUM game tree framework. Green squares signify MIN nodes and gray circles signify SUM nodes. Nodes with black outline signify the most proving node.

2. Test Bed Games: Connect Four and Othello

2.1. Connect Four

Connect Four is a two-player perfect information board game and belongs to the broader class of *k*-in-a-row games such as tic-tac-toe or Gomoku [22]. The two players (red and yellow) take turns placing their pieces on the board, and the main goal is to line up (at least) four chips in an either horizontal, vertical, or diagonal manner on a standard 7×6 board (Figure 2).

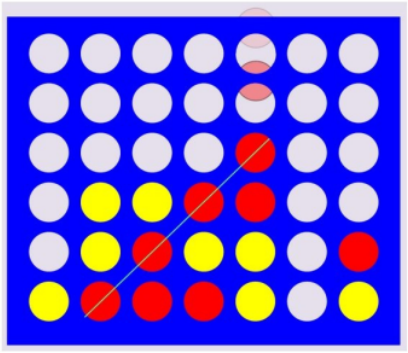

Figure 2. Illustration of Connect Four with a 7×6 board (adopted from [23]).

In Connect Four, gravity is essential so that the pieces fall as far to the bottom as possible. Thus, in each state, at most w moves are possible (a piece placed in one of the columns, if it is not yet filled). The default board size of Connect Four has been solved by Allen (reported in rec.games.programmer on 1 October 1988), and was then independently solved 15 days later in [24]. Its reachable states were roughly estimated in 7×10^{13} states.

The game can be won between 13-ply to 42-ply (the board is filled), making the depth of the search tree highly varied. There are also possibilities for sudden death moves to occur during the game, which made its game-tree structure unbalanced. In terms of complexity, Connect Four has a reasonably good game-tree due to its move limitation and board size.

2.2. Othello

Othello is a board game that requires two players to play, of which one player is playing black disks while the other plays white disks. Othello game has several characteristics:

1. It does not have random elements in its rule.
2. Every gained advantage of a player is equal to the gained disadvantage of their opponent.
3. All of the players are perfectly informed of every previous event in the game, including the initialization stage. All of these characteristics define Othello as a deterministic, zero-sum, perfect information game.

Othello is generally played on an 8×8 board. At the start of the game, two disks of each color are placed diagonally in the center of the board (Figure 3a). The first to play is the player who plays a black disk, followed by a white player. Each player takes turns and makes a legal move on the board until there is no empty square in the board or the opponent is no longer able to make a legal move. A legal move in Othello is defined as placing a disk adjacent to an opponent disk so that the opponent's disk or a row of the opponent's disks is in between the new disk and another disk of the player's color. All of the opponent's disks between these two disks are considered as "captured" and turned over to match the player's color.

The state-space size of Othello is approximately 10^{28} [25], with maximum moves of 60 (30 for each player). It is a balanced game of which the player does not encounter "sudden death" while playing the game. Most of the time, the player is required to play the game until the board is full. Othello can be divided up into three phases: the opening, middle game, and end game. The opening game is defined as the first moves up until the 20th to 26th move. The end game is defined as the last 10–16, and the middle game is all states between the opening and the end game [20].

In playing Othello, some terms are used to explain the movements of the player. These terms are essential to Othello, as they affect how the game progresses [26]. The player's degree of mobility, or "mobility" in short, is used to describe the number of options that are available to a player in a particular stage of the game. In Figure 3b, the available options for the black player are marked with a triangle. In this case, the degree of mobility of the black player is 6. Another relevant term in Othello is "stability," which defines a state where there are disks that cannot be captured by the opponent, no matter what moves they have made. Figure 3c is an illustration of an 8×8 Othello board after 22 moves, with the black player to move. In this stage, white disks marked with an inner circle are the stable disks of the white player. It is a definite disk that cannot be turned into black no matter what sequence is taken by the black player in the subsequent game.

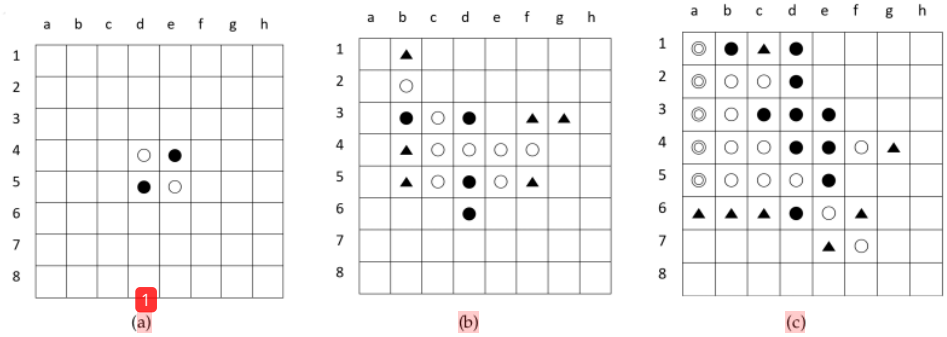

Figure 3. Illustration of Othello with an 8×8 board. (**a**) initial position; (**b**) After eight moves, black is to play. The triangle marked the black player's mobility of six (six move options); (**c**) After 22 moves, black is to play. The triangle marked the black player's mobility of seven (seven position options), and the white disks with marked inner circles are the stable disks (cannot be captured by the black player).

3. Probability-Based Proof Number Search (PPNS)

Probability-based proof number search (PPNS) is a best-first search algorithm that is set to be used in an AND/OR game tree structure. It uses an indicator called probability-based proof number (PPN) to indicate the probability of a node leading to the expected position.

3.1. The Probability-Based Proof Number Indicator

A probability-based proof number (PPN) is a number that specifies the probability of a node to be proven in an AND/OR tree [16]. The AND/OR tree consists of three types of nodes viz. terminal nodes, leaf nodes, and internal nodes. The PPN that is assigned to an OR and AND internal node is the product of its child values. As such, the PPN of a node contains two different pieces of information derived from the current game-tree. All of these nodes have their PPN ($n.ppn$) value that is calculated based on the following formula:

1. If a node n is a terminal leaf node,

   (a) if (n) is a winning node,

   $$n.ppn = 1 \tag{1}$$

   (b) and if (n) is not a winning node,

   $$n.ppn = 0 \tag{2}$$

2. If a node n is a leaf node, then let R be the winning rate as a result of game playout, and θ is a small positive number close to 0.

   (a) If $R \in (0, 1)$,

   $$n.ppn = R \tag{3}$$

   (b) if $R = 1$,

   $$n.ppn = R - \theta \tag{4}$$

   (c) and if $R = 0$,

   $$n.ppn = R + \theta \tag{5}$$

3. If a node n is an internal node,

(a) if n is an OR node,

$$n.ppn = 1 - \prod_{n_c \in \text{children of } n} (1 - n_c.ppn) \quad (6)$$

(b) and if n is an AND node,

$$n.ppn = \prod_{n_c \in \text{children of } n} n_c.ppn \quad (7)$$

3.2. The PPNS Algorithm

The core algorithm of the PPNS consists of four steps: selection, expansion, simulation (or playout), and backpropagation.

1. **Selection**: For all nodes, select the child with the maximum PPN at OR nodes, and the child with the minimum PPN at AND nodes. Regard these nodes as the most proving node (MPN) for expansion. The process of the selection step is given in Algorithm 1.
2. **Expansion**: Expand the most proving node (Algorithm 2). In this phase, all available next moves from the MPN position are regarded as a child node.
3. **Simulation/Playout**: For positions not already in the tree, the move is simulated in a random self-play mode until the game end (Algorithm 3). After several playouts, the PPNs of expanded nodes are derived from Monte Carlo evaluations. In this step, if the result of simulations (R) is equal to either 1 or 0, which is either reduced or added with a small value (θ) to differentiate it from the leaf node.
4. **Backpropagation**: Update the PPNs from extended nodes back to the root, while following the AND/OR probability rules from Section 3.1 (Algorithm 4).

All of the steps are repeated until PPN of the root node reaches 1 or 0. If the PPN is equal to 1, the game is defined as solved. Otherwise, it is defined as unsolved. In this study, PPNS is implemented based on Algorithm 5.

Algorithm 1 Selection algorithm

```
1:  procedure SELECTION(node)
2:      CreateNode(MPN)
3:      if node is OR node then
4:          MPN.ppn ← −∞
5:          for all child of node do
6:              if child.ppn ≥ MPN.ppn then
7:                  MPN ← child
8:              end if
9:          end for
10:     else                                          ▷ node is AND node
11:         MPN.ppn ← ∞
12:         for all child of node do
13:             if child.ppn ≤ MPN.ppn then
14:                 MPN ← child
15:             end if
16:         end for
17:     end if
18:     return MPN
19: end procedure
```

Algorithm 2 Expansion algorithm

```
1: procedure EXPANSION(node)
2:     if node is an Internal node then
3:         Expand node
4:         PlayOut(node,player)                          ▷ Determine node's PPN
5:     else if node is a Leaf node then
6:         if Win then
7:             root.ppn ← 1
8:         else
9:             root.ppn ← 0
10:        end if
11:    end if
12: end procedure
```

Algorithm 3 Playout algorithm

```
1: procedure PLAYOUT(node, player)
2:     win ← 0
3:     for count ← 1 to simulation_num do
4:         Simulate game randomly until End
5:         if player wins then
6:             win++
7:         end if
8:     end for
9:     R = win/simulation_num
10:
11:    if R is 1 then
12:        R ← R − θ
13:    else if R is 0 then
14:        R ← R + θ
15:    end if
16:    node.ppn ← R
17: end procedure
```

Algorithm 4 Backpropagation algorithm

```
1: procedure BACKPROPAGATION(node)
2:     if node is an AND node then
3:         ppn ← 1
4:         for each child of node do
5:             ppn ← ppn ∗ child.ppn
6:         end for
7:     else if node is an OR node then
8:         ppn ← 0
9:         for each child of node do
10:            ppn ← ppn ∗ (1 − child.ppn)
11:        end for
12:        ppn ← 1 − ppn
13:    end if
14:    node.ppn ← ppn
15: end procedure
```

Algorithm 5 Probability-based proof number search

```
1: function PPNSEARCH(root)
2:     Create root node
3:     while root.ppn ≠ 0 and root.ppn ≠ 1 do
4:         MPN ← Selection(root)
5:         Expansion(MPN)
6:         BackPropagation(root)
7:     end while
8:     if root.ppn is 1 then
9:         return true                                  ▷ Root is solved
10:    else if root.ppn is 0 then
11:        return false                                 ▷ Root is unsolved
12:    end if
13: end function
```

The PPN of a node contains two pieces of information derived from the game tree: information acquired from the current known tree structure (Equations (1) and (2)) and information derived from the unknown tree structure. From the current position, Equations (3)–(5) are employed to calculate the probability of winning or losing, thus providing more information from part of the tree that is yet to be expanded. While obtaining the PPN of a node, every statistical value produced by a simulated game is accountable, which differs compared to the MIN/SUM rules (i.e., used by MCPNS, where the usage of MIN/SUM rules may disregard some of these values due to its rule).

4. Experimental Results and Analysis

4.1. Experiment Setups

The experiments were performed by a computer with an Intel i5-8400 processor (2.81 GHz) using 8 GB of RAM, running Windows 10, on a 64-bit machine. The program for both Connect Four and Othello are built in C++ and can be publicly accessed at https://github.com/90sradiosong/ProbabilityPNS. The experiments were performed sequentially to ensure correctness and independent measurements. This means that only one position is solved by one algorithm at a time.

Three different algorithms, PNS, MCPNS, and PPNS, were given the task to solve each position independently. The parameters for each algorithm are given as follows: For MCPNS and PNS, the number of simulated moves is 60, while the $\theta = 0.01$ was set for the PPNS.

The experimental procedures were twofold (due to the nature of gameplay between the two games, Connect Four and Othello). For the first part of the experiment, 200 Connect Four positions were generated, where each position contains 12-ply of randomly generated moves. All algorithms are terminated when it expands into 35,000,000 nodes or when the time elapsed reaches 420 s, whichever comes first. The number of iterations performed, the number of nodes and visited nodes, and the time used to solve the position were measured.

For the second part of the experiment, 200 Othello positions were generated for three different stages. Each position contains either 18, 26, or 32 randomly generated moves (a total of 600 positions). In Othello's case, one move is defined as a position taken by one player or a half-ply. These positions are selected, which represent the opening, mid-opening, and middle positioning of the Othello game stage. All algorithms are stopped once it either expands into 8,000,000 nodes, or the time elapsed reaches 600 s. The number of completions for each position was measured.

4.2. Experiment Results on Connect Four

The result of the experiment is shown in Figure 4. All of the positions produced the same conclusion unless it reached either set limit. In the limited configuration, PNS performs the best, with 122 positions (solved and unsolved), followed by PPNS with 102 positions, and MCPNS with 82 positions (out of 200 positions in total). Note that PNS is bounded by the amount of memory (it stopped its search due to an excess number of nodes visited), while PPNS and MCPNS are bounded by time (it stopped its search due to time limit set).

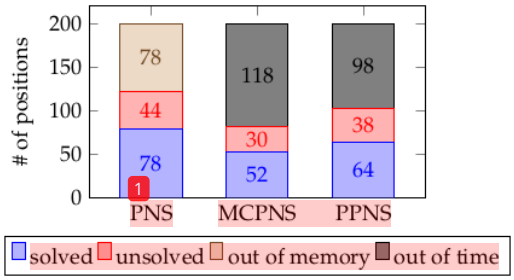

Figure 4. Number of Connect Four positions solved, unsolved, and out of bounds by proof number search (PNS), Monte Carlo proof number search (MCPNS), and probability-based proof number search (PPNS).

Since PPNS combines two kinds of information (the information from the explored part of the tree and information from the unexplored part of the tree), it should be able to utilize such information to conclude. The PPNS visited a total of 2,295,856 nodes, while PNS and MCPNS visited 1,921,730 and 1,114,283 nodes, respectively, which corresponds to the position with the highest time difference between PNS and PPNS (322.345 s). This observation implies that PPNS was not able to thoroughly combine the information from both parts of the tree (sub-optimal performance), which can be demonstrated from the PPN value of the root for such a position (Figure 5a).

For the first 10,000 iterations, $PPN = 0.005794$ at the root, which is also the peak value for the entire solving process. The PPN decreases and stabilizes (between the 70,000th iteration and the 170,000th iteration), while a sudden increase appears (180,000th iteration) with $PPN = 0.001699$ at the root, which subsequently stayed under 0.001 for the rest of the process. While $PPN \approx 0.000000$ by the 340,000th iteration, the algorithm keeps iterating until it reaches the 677,765th iteration (twice the expected iterations). This also affects the number of nodes needed to solve the position. By the 340,000th iteration, the total number of nodes explored is 1,205,010, while at the 677,765th iteration, the total number of nodes explored is also almost doubled (2,274,949 nodes).

Another solved position with a time difference of 77.802 s between PNS and PPNS was also observed. For this position, PPNS visited a total of 661,783 nodes, while PNS and MCPNS visited 11,055 and 1567 nodes, respectively. In this case, there is a large difference between total nodes visited by PNS and MCPNS. Observation of the PPN value of the root of this position (Figure 5b) shows a similar trend. The $PPN = 0.99995$ at the root for the first 10,000 iterations, which then fluctuates until the 100,000th iteration, where the $PPN = 1.00000$. Nevertheless, the algorithm makes 214,110 iterations.

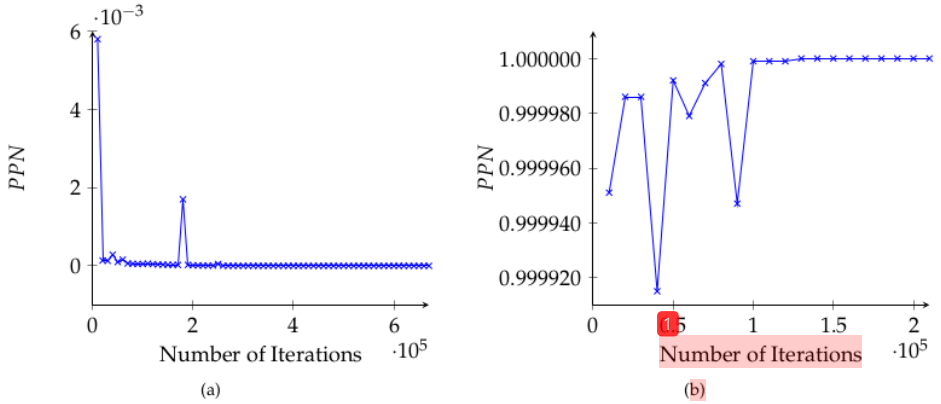

Figure 5. Probability-based proof number (PPN) value of the root of two different positions for PPNS. (**a**) an example position with a time difference of 322.345 s; (**b**) an example position with a time difference of 77.802 s.

From these two observations, it can be concluded that one of the problems that occurs in PPNS is the prolonged search problem. A prolonged search may occur due to the precision of the floating-point number, which contains a trailing value. In other words, a PPN of 0.000000 still contains a minimal trailing number, which is not recognized as an absolute value. The state of a conclusion is defined as an integer (1 for solved and 0 for unsolved). A trailing number in the float number case which is caused by how the computer stores floating points leads to an incessant loop since it is hard to reach such a precise integer number. As such, a precision rate value ($pr = 0.00001$) was added to the core PPNS algorithm to ensure that the search can be timely optimized while not affecting the other parameters (Algorithm 6).

Algorithm 6 Probability-based Proof Number Search (with precision rate parameter)

```
1:  function PPNSEARCH(root)
2:      Create root node
3:      while |root.ppn − 0| ≤ pr and |root.ppn − 1| ≤ pr do
4:          MPN ← Selection(root)
5:          Expansion(MPN)
6:          BackPropagation(root)
7:      end while
8:      if root.ppn is 1 then
9:          return true                                  ▷ Root is solved
10:     else if root.ppn is 0 then
11:         return false                                 ▷ Root is unsolved
12:     end if
13: end function
```

A further experiment was conducted with two different values of θ ($\theta = 0.01, 0.001$) and three different pr values ($pr = 0.001, 0.0001, 0.00001$) were used, resulting in six different configurations (Figure 6a,b). These configurations were applied to 200 generated Connect Four positions similar to those of the original experiment in Section 4.1.

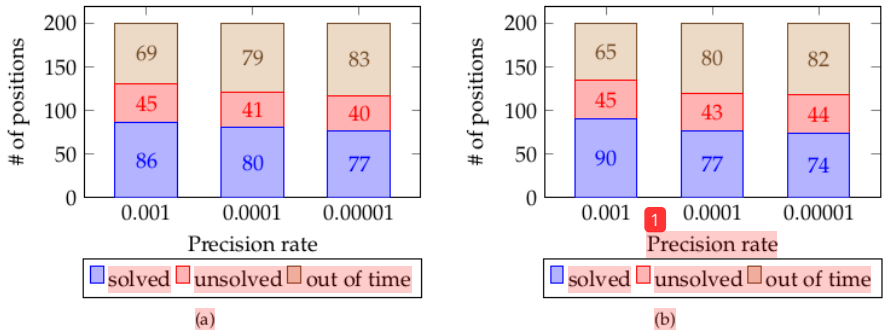

Figure 6. Number of positions solved, unsolved, and out of bounds by PPNS. (**a**) $\theta = 0.01$, (**b**) $\theta = 0.001$.

The application of the pr parameter in PPNS increases the total positions that are concluded (solved or unsolved). The optimal configuration is $\theta = 0.001$ and $pr = 0.001$ with a total of 135 positions. This configuration is higher than that of PNS (122 positions) and MCPNS (82 positions) from the previous experiment. Elapsed time and nodes explored are also affected by the application of the pr value. The average elapsed time and nodes explored are given in Table 1. The change of θ value does not significantly affect the average time and node, but an increase in the pr value yields a longer elapsed time, and a higher number of nodes visited. A small number of pr value leads to an increase in the total number of positions. However, lowering the pr value to a rate where it is less precise than that of θ would lead to false conclusions. In this case, the best pr value would be of the same with θ, or one rate above it (smaller pr).

Table 1. Average time and node for each PPNS configurations.

θ	pr	Average Time(s) *	Average Node **
0.01	no precision	251.180925	2,004,052.895
0.01	0.001	183.256315	1,323,718.975
0.01	0.0001	214.63364	1,582,187.985
0.01	0.00001	226.707205	1,667,750.875
0.001	0.001	174.90736	1,269,587.96
0.001	0.0001	217.987025	1,614,710.08
0.001	0.00001	229.07937	1,638,641.2

* less is better; ** more is better, relative to *.

4.3. Experiment Results on Othello

The experiment results are given in Table 2. The experiment was conducted with consideration of the precision rate ($pr = 0.00001$) parameter, identified from the Connect Four experiment. All of the positions produced the same conclusion unless it reached the termination condition (positions without any conclusion are designated as "out"). Based on the results, PPNS performs the best by solving most positions in every stage of the game (72% positions on average), followed by PNS (21.09% positions on average) and MCPNS (12.14% position on average).

The PPNS performance was as expected, which further strengthens the findings of the previous study employing PPNS on the *P*-game tree [16]. This performance affirms that the PPNS optimal solver for games with a balanced tree structure. The performance difference of each algorithm in different stages of the game is depicted in Figure 7. It is interesting to see the differences in growth patterns given by each algorithm. Small differences were observed for both PNS and MCPNS in opening and mid-opening stages but sharply increased during the middle game stage.

Table 2. Experimental result of different algorithms applied to Othello positions.

Moves	Algorithm	Solved	Unsolved	Out	Completion * (%)
18	PPNS	52	56	92	54.00
	PNS	12	1	187	6.50
	MCPNS	5	4	191	4.50
26	PPNS	63	88	49	75.50
	PNS	19	16	165	17.50
	MCPNS	10	10	180	10.00
32	PPNS	82	91	27	86.50
	PNS	39	39	122	39.00
	MCPNS	6	37	157	21.50

* Completion = (Solved + Unsolved)/200; even if the result concludes that it is unsolved, the algorithm is still able to conclude.

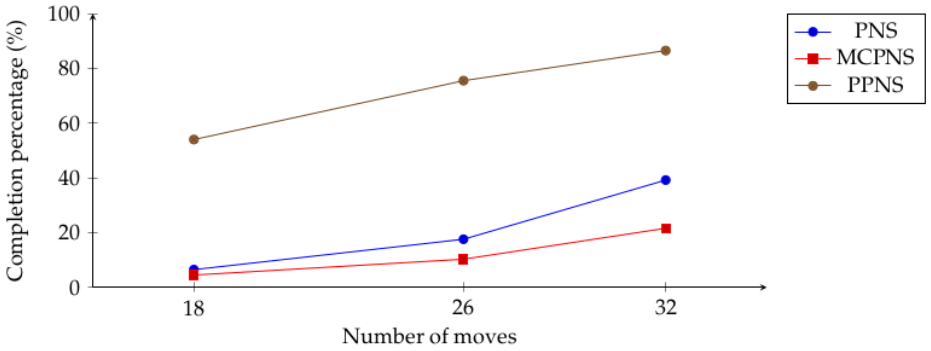

Figure 7. The completion percentage of PNS, MCPNS, and PPNS for different numbers of moves.

In contrast, PPNS completion growth behaves differently where sudden increases were observed in the opening and mid-opening stages while stabilizing when reaching the middle game stage. While the increase in completion percentage in PNS and MCPNS is expected (more information revealed throughout the gameplay), the growth pattern of PPNS is a curious situation. Hence, a further experiment is conducted by adding more positions to solve.

Similar to the original experiment setup (Section 4.1), while retaining similar experiment variables to ensure that the result is valid, another 200 Othello positions were generated for three additional stages (a total of six stages). The new stages considered 22 (opening), 36 (middle), and 40 (end) randomly generated positions—thus illustrating the performance of PPNS throughout all stages of Othello (see results in Table 3). The reduction of the number of positions that are marked as "Out" shows the increase in the completion percentage.

Table 3. Additional experiment result of applying PPNS to different stages of Othello positions.

Moves	Solved	Unsolved	Out	Completion * (%)
18	52	56	92	54.00
22	60	57	83	58.50
26	63	88	49	75.50
32	82	91	27	86.50
36	97	94	8	95.50
40	83	117	0	100.00

* Completion = (Solved + Unsolved)/200; even if the result concludes that it is unsolved, the algorithm is still able to conclude.

In the opening stage of the game (move 18 and 22), since there are still many spaces on the board, the outcome of the game highly varies, which creates a large search space (Figure 8). However, the degree of mobility of each player can be small, as there are not as many disks on the board, which makes the PPNS completion increase slowly. Subsequently, the mid-opening of the game (between opening and middle stages) represented by the move 26, around one-third of the board has been filled with players' disks. The focus has likely shifted from the center position, where a strong Othello player increases their mobility by making a high number of varying moves. Thus, PPNS completion experienced sharp increases.

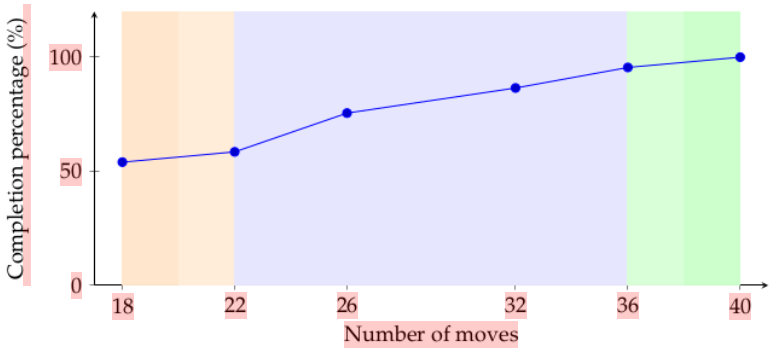

Figure 8. The completion percentage of PPNS in different stages of Othello. The left-most area colored in orange represents the opening stage of Othello, the middle area (blue) represents the middle game of Othello, and the right-most green area represents the end game of Othello.

The middle stage (move 32 and 36) follows the player that competed to gain more stable disks that cannot be captured by their opponent. As more parts of the board are filled with Othello disks, the mobility is again reduced, which makes the completion of PPNS increase slowly. Finally, coming to the border of the middle and end stages (move 40), the number of positions available is minimal, and the sequence of moves taken becomes crucial. The PPNS degree of completion is increased from the previous stage and reaches 100% completion. At this stage, PPNS is said to solve most positions.

5. Discussion

This study conducted experiments on two distinct games, Connect Four and Othello, which adopted to examine the quality of PPNS. These games represent the real case of unbalanced and balanced game tree structures. In addition, two related algorithms, PNS and MCPNS, were similarly applied to these games for comparison purposes. Based on such a setting, the experiments were conducted in twofold.

In the first experiment, among the three algorithms applied to Connect Four, PPNS showed that there are positions where it performs suboptimally. It visits more nodes than PNS before solving the positions (against the idea of PPNS combining information of the visited node and the probability of unexplored nodes). Upon further analysis, the prolonged search problem was identified, which caused the PPNS to underperform. The compared algorithms (PNS and MCPNS) uses the integer-based backpropagation technique, while PPNS use real numbers—thus making the precision of floating-point affect PPNS performance. Thus, the PPN of a node went through product operations, where errors on precision arose.

A precision rate (pr) parameter is introduced to the PPNS algorithm to negate the precision problem. Experiments with various configurations show that the addition of the pr value increases the performance of PPNS without affecting its accuracy result. It was found that the closer the pr value is to θ, the better the PPNS performance. However, a pr value lower than θ will worsen the PPNS

performance. The results from the experiments demonstrate that PPNS with a pr value reduces the amount of explored nodes needed to solve a position by up to 57%. This situation implies that even a small amount of explored nodes allowed information from an unexplored area to be exploited and combined to reach the desired goal.

Another experiment was also performed while considering the different stages of the Othello game (opening, mid-opening, and middle stages). It was observed that an increasing amount of information allowed for more positions to be solved. PPNS utilizes information from both explored and unexplored nodes, resulting in a drastically better performance compared to the other two algorithms, and highlighting the importance of information from the explored nodes.

The experiment was expanded to observe the behavior of PPNS into more specific stages of Othello. The result from additional stages of Othello's positions serves as an indication of the PPNS behavior. In the opening and mid-opening stages, the player mobility (and branching factor of the search tree) is limited, and the possible search space is enormous. As such, statistical information from the unexplored nodes during these stages was crucial for selecting the most proving node (MPN), which either leads to better mobility or just thinning out the number of choices (a sharp increase in the PPNS completion percentage).

Compared to PNS and MCPNS, such an increase appears in the later stage (middle) when more information becomes available. This situation showed that PPNS was able to obtain such information earlier. For the middle to end game stages, the PPNS degree of completion slowly increases to 100% completion when approaching the end game. In this stage, players' mobility is again limited because the board is getting filled with player disks, and the search space becomes small. In this stage, information from the explored nodes becomes more prominent since its information is more readily available. As such, information from unexplored nodes becomes prominent and highly critical for PPNS and lies between the opening and middle stages of the Othello game.

Current insight suggested from the two experiments leads to a hypothesis that PPNS is suitable for a game that requires a long look-ahead strategy. PPNS has been introduced to solve two different tree structures (an unbalanced tree in Connect Four and a balanced tree in Othello), where PPNS demonstrates better performance than the other related algorithms. PPNS application to Connect Four showed that the quality of the available information is critical compared to the quantity of the information, where a small amount of explored nodes, combined with the appropriate statistical information (or rather the precision of said information) of the unexplored nodes, can vastly improve the performance of the solver. Furthermore, the PPNS application to Othello demonstrates the importance of considering the appropriate "moment" to take advantage of the information from both the explored and unexplored nodes to solve more positions faster and earlier.

6. Conclusions

PPNS is a best-first search algorithm that employs information from both inside and outside of a game tree. In this paper, PPNS, along with the other two best-first search algorithms (PNS and MCPNS), were employed to solve randomly generated game positions of Connect Four and positions in different stages of Othello. The result from the application of PPNS in Connect Four identified an implementation problem related to a prolonged search that arises from the real number-based operation. Alleviating the problem through the precision rate (pr) parameter, PPNS provides performance improvement compared to the compared algorithms by solving a higher number of positions. It was found that, in the event of reduced explored information, PPNS is able to reach the desired conclusion by exploiting information from an unexplored part of a game tree.

Moreover, information coming from unexplored nodes affects the ability of PPNS to solve positions at all of Othello's stages, subsequently increasing its quality by boosting the completion percentage of PPNS in Othello in earlier stages of the game compared to other best-first search algorithms. Further observation towards PPNS showed that the combined information from explored and unexplored nodes affects its behavior in a different stage of Othello. The information of

the unexplored nodes becomes highly critical in the earlier stages of the Othello game, while the information of the explored nodes becomes prominent in later stages of the Othello game.

Further works in this direction include, but is not limited to, (1) an application of PPNS in another real game with a larger tree, (2) expansion into the depth-first version of PPNS (df-PPNS), and (3) a comparison between depth-first proof number search and df-PPNS in (very) hard problem domains.

Author Contributions: Conceptualization, A.P.; methodology, A.P.; software, A.P.; validation, A.P., M.N.A.K. and H.I.; formal analysis, A.P.; investigation, A.P.; resources, M.N.A.K. and H.I.; data curation, M.N.A.K. and H.I.; writing—original draft preparation, A.P.; writing—review and editing, M.N.A.K. and H.I.; visualization, M.N.A.K. and H.I.; supervision, M.N.A.K. and H.I.; project administration, M.N.A.K. and H.I.; funding acquisition, M.N.A.K. and H.I. All authors have read and agreed to the published version of the manuscript.

Funding: This research is funded by a grant from the Japan Society for the Promotion of Science, within the framework of the Grant for fundamental research.

Conflicts of Interest: The authors declare that there is no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Allis, L.V.; van der Meulen, M.; Van Den Herik, H.J. Proof-number search. *Artif. Intell.* **1994**, *66*, 91–124. [CrossRef]
2. Khalid, M.N.A.; Yusof, U.K.; Iida, H.; Ishitobi, T. Critical Position Identification in Games and Its Application to Speculative Play. In Proceedings of the International Conference on Agents and Artificial Intelligence, Lisbon, Portugal, 10–12 January 2015; SCITEPRESS—Science and Technology Publications, Lda: Setubal, Portugal, 2015; Volume 2, pp. 38–45. doi:10.5220/0005179900380045. [CrossRef]
3. Kishimoto, A.; Winands, M.H.; Müller, M.; Saito, J.T. Game-tree search using proof numbers: The first twenty years. *Icga J.* **2012**, *35*, 131–156. [CrossRef]
4. Ishitobi, T.; Plaat, A.; Iida, H.; van den Herik, J. Reducing the Seesaw Effect with Deep Proof-Number Search. In *Advances in Computer Games*; Plaat, A., van den Herik, J., Kosters, W., Eds.; Springer International Publishing: Cham, Switzerland, 2015; pp. 185–197.
5. Primanita, A.; Iida, H. Nature of Probability-based Proof Number Search. In Proceedings of the Sriwijaya International Conference of Information Technology and Its Applications (SICONIAN), Palembang, Indonesia, 16 November 2019.
6. Van Den Herik, H.J.; Winands, M.H. Proof-number search and its variants. In *Oppositional Concepts in Computational Intelligence*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 91–118.
7. Nagai, A. Df-pn Algorithm for Searching AND/OR Trees and Its Applications. Ph.D. Thesis, Department of Information Science, University of Tokyo, Tokyo, Japan, 2002.
8. Coulom, R. Efficient selectivity and backup operators in Monte Carlo tree search. In *International Conference on Computers and Games*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 72–83.
9. Kocsis, L.; Szepesvári, C. Bandit based Monte Carlo Planning. In *ECML-06. Number 4212 in LNCS*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 282–293.
10. Chaslot, G.M.J.B.; Winands, M.H.M.; Herik, H.J.V.D.; Uiterwijk, J.W.H.M.; Bouzy, B. Progressive strategies for Monte Carlo tree search. *New Math. Nat. Comput.* **2008**, *4*, 343–357. [CrossRef]
11. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484. [CrossRef] [PubMed]
12. Bouzy, B.; Tutorial, C. Old-fashioned computer go vs monte-carlo go. In Proceedings of the IEEE Symposium on Computational Intelligence in Games (CIG), Honolulu, HI, USA, 1–5 April 2007.
13. Althöfer, I. On board-filling games with random-turn order and Monte Carlo perfectness. In *Advances in Computer Games*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 258–269.
14. Saito, J.T.; Chaslot, G.; Uiterwijk, J.W.; Van Den Herik, H.J. Monte Carlo proof-number search for computer Go. In *International Conference on Computers and Games*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 50–61.
15. Higham, N.J. *Accuracy and Stability of Numerical Algorithms*; SIAM: Philadelphia, PA, USA, 2002; Volume 80.
16. Song, Z.; Iida, H.; Van Den Herik, H.J. Probability based Proof Number Search. In Proceedings of the 11th International Conference on Agents and Artificial Intelligence, Prague, Czech Republic, 19–21 February 2019.

17. Palay, A.J. *Searching with Probabilities*; Technical Report; Carnegie-Mellon Univ Pittsburgh PA Dept of Computer Science: Pittsburgh, PA, USA, 1983.
18. Saffidine, A.; Cazenave, T. Developments on product propagation. In *International Conference on Computers and Games*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 100–109.
19. Stern, D.; Herbrich, R.; Graepel, T. Learning to solve game trees. In Proceedings of the 24th International Conference on Machine Learning, Corvalis, OR, USA, 20–24 June 2007; pp. 839–846.
20. Van Eck, N.J.; van Wezel, M. Application of reinforcement learning to the game of Othello. *Comput. Oper. Res.* **2008**, *35*, 1999–2017. [CrossRef]
21. Buro, M. The evolution of strong Othello programs. In *Entertainment Computing*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 81–88.
22. Edelkamp, S.; Kissmann, P. On the complexity of BDDs for state space search: A case study in Connect Four. In Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 7–11 August 2011.
23. Maby, E.; Perrin, M.; Bertrand, O.; Sanchez, G.; Mattout, J. BCI could make old two-player games even more fun: a proof of concept with "Connect Four". *Adv. Hum. Comput. Interact.* **2012**, *2012*, 124728. [CrossRef]
24. Allis, V. A Knowledge-Based Approach of Connect-Four-the Game is Solved: White Wins. Master's Thesis, Vrije Universiteit, Amsterdam, The Netherlands, 1988.
25. Van Den Herik, H.J.; Uiterwijk, J.W.; Van Rijswijck, J. Games solved: Now and in the future. *Artif. Intell.* **2002**, *134*, 277–311. [CrossRef]
26. Lazard, I. Othello Strategy Guide. 1993. Available online: http://radagast.se/othello/Help/strategy.html (accessed on 18 October 2019).

Characterizing the Nature of Probability-Based Proof Number Search A Case Study in the Othello and Connect Four GamesCharacterizing the Nature of Probability-Based Proof Number Search A Case Study in

ORIGINALITY REPORT

96%	95%	95%	9%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	mdpi-res.com Internet Source	95%
2	Submitted to Curtin International College Student Paper	1%
3	www.scitepress.org Internet Source	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		