# Deep Proof-Number Search and Aesthetics of Mating Problems

by

Taichi Ishitobi

SUBMITTED TO
JAPAN ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

SUPERVISOR
Prof. Dr. Hiroyuki Iida

Schoolo of Information Science
Japan Advanced Institute of Science and Technology

*Supervised by*
Prof. dr. H. Iida

*Reviewed by*
Prof. dr. H.J. van den Herik (Leiden University, the Netherlands)
Prof. dr. T. Kaneko (University of Tokyo, Japan)
Dr. M. A. B. M. Iqbal (Universiti Tenaga Nasional, Malaysia)
Prof. dr. K. Ikeda
Prof. dr. S. Hasegawa

# Abstract

In this study, we focus on the search indicators used in the search algorithms. The search indicators are the values given in each node in the game tree for guiding the search direction of the search algorithms. The search indicators have been used for developing a strong computer player. However, we hypothesize that there are other potentials. Under this assumption, we tried to consider a new aspect of the search indicators and to confirm its utilization.

**Chapter 2** presents the previous works related to the AND/OR tree search and the conspiracy number search. Additionally, we focus on the concomitant search indicators. We explained each algorithm with focus on the search indicators. And, we provided the studies of the search indicators and tried to find the other viewpoints except the original meaning. Then, recent researches of the conspiracy number search present the good hints to us.

**Chapter 3** describes a new search algorithm based on proof numbers, named DeepPN. DeepPN has three search indicators (pn, dn, *deep*) and a single parameter, *R*, that allows a choice between depth-first and best-first behavior. DeepPN employs two types of values, viz., proof numbers and deep values which register the depth of nodes. For measuring the performance of DeepPN, we tested DeepPN on solving Othello endgame positions and on the game of Hex. We achieved two indicative results in Othello and Hex. The algorithm owes its success to the formula in which best-first and depth-first search are applied in a "balanced" way. The results show that DeepPN works better than PN-search in the games which build up a *suitable* tree.

**Chapter 4** shows the relationship between the search indicators and the evaluation of the mating problems in shogi (Tsume-shogi). We focused on the aesthetics of tsume-shogi such as interesting, beautiful and refreshing. Then, it seems a promising approach to focus on the relationship between aesthetic and complexity indicated by the proof and disproof numbers for the assessment of tsume-shogi. The indicator of the disproof number looks more promising than other factors such as the proof number and the number of visited nodes during search when using the tsume-shogi with relatively short steps in the competition. This means that the difficulty of Magire is more critical component for aesthetic.

**Keywords:** Search indicator, Deep Proof-Number Search, Aesthetics of mating problems, Conspiracy Number Search

# Acknowledgement

Firstly, I would like to express my sincere gratitude to my advisor Prof. Hiroyuki Iida for his continuous support during my Ph.D, for his patience, motivation, and immense knowledge. From the time I entered JAIST until today, he helped me from various aspects. I could grow gradually thanks to his support, such as the opportunities that he gave me of studying three times abroad, joining conferences, etc. I could not have imagined having a better advisor and mentor for my Ph.D.

Besides my adviser, I would like to show my greatest appreciation to Prof. Jaap van den Herik and Prof. Aske Plaat. They supported me during two of my stays abroad for my research. At first, I could learn the basic way of doing research for my Ph.D. Then, I succeeded in proposing the DeepPN algorithm during my last experience of studying abroad. Without their guidance and persistent help, this thesis would not have been possible.

I also would like to thank Prof. Kokolo Ikeda for giving me insightful comments and suggestions at times when I experienced problems. Prof. Ikeda helped me not only in my research, but also in my life. I cannot forget his support.

Additionally, I would like to thank the rest of my thesis committee: Prof. Tomoyuki Kaneko, Dr. Azlan Iqbal and Prof. Shinobu Hasegawa. They gave me constructive comments and warm encouragement. And also, their hard questions made my thesis better in the end.

Thirdly, I thank supporters: Asakura and Joke. They supported my life, so I could concentrate on my study. I also want to thank my fellow lab-mates, for having had with me stimulating discussions and for all the fun we have had together in the last five years.

Last but not least, I am deeply grateful to my family, for helping me to achieve the challenge of my Ph.D and in assisting my life. I made them worry so much, but now, I could graduate from JAIST thanks to their devoted assistance.

# Contents


**1 Introduction** **1**
- 1.1 Background . . . 1
- 1.2 Problem Statement and Research Questions . . . 3
- 1.3 Research Methodology . . . 4
- 1.4 Structure of the Thesis . . . 4

**2 Solving Game Positions and Search Indicators** **5**
- 2.1 Solving Game Positions . . . 5
  - 2.1.1 AND/OR Tree . . . 6
  - 2.1.2 Basic Search Algorithms . . . 8
- 2.2 AND/OR Tree Search using Proof Numbers . . . 9
  - 2.2.1 Proof Number Search . . . 9
  - 2.2.2 Related Algorithms . . . 10
  - 2.2.3 Depth-First Proof-Number Search . . . 13
  - 2.2.4 Derivation of Depth-First Proof-Number Search . . . 14
- 2.3 New Perspective of Conspiracy Numbers . . . 17
  - 2.3.1 Conspiracy Numbers . . . 18
  - 2.3.2 Conspiracy Number Search . . . 20
  - 2.3.3 Analyzing Games with CNS . . . 22
- 2.4 Concluding Remarks . . . 25

**3 Deep Proof-Number Search** **27**
- 3.1 The Seesaw Effect . . . 28
- 3.2 Deep PN . . . 30
  - 3.2.1 The Basic Idea of DeepPN . . . 30
  - 3.2.2 Performance with Othello . . . 33
  - 3.2.3 Performance with Hex . . . 35
  - 3.2.4 Discussion . . . 37
  - 3.2.5 Focusing on the Depth and $R$ . . . 38
  - 3.2.6 Seesaw Effect and $R$ . . . 39
- 3.3 Conclusion and Future works . . . 42

**4 Aesthetics and Search Indicators** **43**
- 4.1 Aesthetic of Mating Problems . . . 44
  - 4.1.1 Mating Problems and Assessment Factors . . . 44
  - 4.1.2 Experiments using Kanju-Award Problems . . . 45
  - 4.1.3 Experiments using Tsume-shogi Competition . . . 46
  - 4.1.4 Comparison with Previous Works . . . 50

4.1.5 Discussion . . . . . 52
4.2 Concluding Remarks . . . . . 53

**5 Conclusion** **55**
5.1 Summary . . . . . 55
5.2 Answers to RQ1 and RQ2 . . . . . 56
5.3 Answer to Problem Statement . . . . . 56
5.4 Future Work . . . . . 56

**A Additional Experiments with DeepPN** **67**
A.1 Othello Experiments using WZebra . . . . . 67
A.2 DeepPN with Previous Works . . . . . 70

# List of Figures


2.1 OR node in AND/OR tree . . . . . 6
2.2 AND node in AND/OR tree . . . . . 6
2.3 Four types of additional node names in AND/OR tree . . . . . 7
2.4 An example of minimax tree and its conspiracy numbers . . . . . 19
2.5 An illustration of Positive_CN_Sum and Negative_CN_Sum . . . . . 22

3.1 An example of seesaw effect. . . . . 28
3.2 An example of suitable tree for Othello end-game position. . . . . 31
3.3 Othello: The # of iterations and # of nodes. . . . . 34
3.4 Othello: The dispersion of reduction rates for # of iterations and # of nodes . . . . . 35
3.5 Hex: # of iterations and # of nodes for Hex(4). . . . . 36
3.6 Hex: The detail of Figure 3.5b. . . . . 36
3.7 Hex: The dispersion of reduction rates for # of iterations and # of nodes 37
3.8 Othello: Relationship between search depth and reduction rates. . . . 38
3.9 Hex: Relationship between search depth and reduction rates. . . . . 39
3.10 Othello: The relationship between estimate frequency of seesaw effect in root node and reduction rates. . . . . 40
3.11 Hex: The relationship between estimate frequency of seesaw effect in root node and reduction rates. . . . . 41

4.1 Ranking of Competition in 2014 with Proof Number . . . . . 47
4.2 Ranking of Competition in 2014 with Disproof Number . . . . . 47
4.3 Ranking of Competition in 2014 with # of Nodes . . . . . 47
4.4 The relationship between aesthetics of tsume-shogi, difficulty of Magire and maximum disproof number. . . . . 53

A.1 Othello: The # of iterations and # of nodes. . . . . 68
A.2 Othello with WZebra: # of iterations and # of nodes for 16-ply Othello with WZebra. . . . . 68
A.3 Othello: The dispersion of reduction rates for # of iterations and # of nodes . . . . . 69
A.4 Othello with WZebra: The dispersion of reduction rates for # of iterations and # of nodes . . . . . 69

# List of Tables


2.1 An example of conspiracy numbers in the root node . . . . . 18

4.1 7-step compeition problems and general problems . . . . . 46
4.2 15-step Kanju Award-winning problems and general problems . . . . 46
4.3 A correlation coefficient R between ranking and search indicators in each year . . . . . 48
4.4 A Spearman's rank correlation coefficient ρ between ranking of competitions and search indicators using decreasing order in each year . . 49
4.5 The p-values using Student's t-test for Table 4.4. . . . . 49
4.6 The summary of built up decision trees using J4.8. . . . . 51
4.7 The precision values for each class using each input data . . . . . 52
4.8 The recall values for each class using each input data . . . . . 52
4.9 The F-measure values for each class using each input data . . . . . 52

A.1 Setting of WZebra . . . . . 67
A.2 Results of each search algorithm using 16-ply Othello endgame positions. 70
A.3 Results of each search algorithm using Hex(4). . . . . 71

# Chapter 1

# Introduction

In this thesis, we salute the breakthrough achieved by the team of Google DeepMind. We have learned many things of their breakthrough publication [61]. Big contributions can only be big through the existence of small contributions. This thesis focuses on search indicators, which perform in terms of evaluation, guidance and prediction. We attempt to improve the existing search techniques by multiple changes of the search indicators. Our findings may be small, but the impact is considerable in the small domains of Othello and Hex. Moreover, the search indicators have promises for solving small Go boards [10]. Finally, new search indicators can lead to the improvement of the aesthetic concepts as used so far [22, 23].

## 1.1 Background

Originally, researchers used tree search algorithms for optimization [43]. Thereafter they started to apply them for the development of computer-playing algorithms. These algorithms focused on game-tree search, in particular game-tree search for two-person zero-sum games [66], such as the game of chess, checkers and shogi (Japanese chess). For those games, one of the most popular game-tree search algorithms is minimax tree search [58]. In the 1960s and 1970s, many game-tree search algorithms were proposed, e.g., $\alpha - \beta$ algorithm [34]. Later, in the beginning of this century, Monte Carlo tree search (MCTS) was developed [8]. Moreover, many techniques which accompany the game-tree search algorithms were developed, e.g., in the field of machine learning [12] and evolving strategy [7, 9].

Consequently, the quality of the game-playing programs was improved by these new game-tree search algorithms and the corresponding related techniques. For example, the checkers program CHINOOK defeated the human world champion in 1994 [54] and the chess program DEEP BLUE [6] defeated the human world champion in 1997. In Japan, the shogi program BONKRAS won a match against a top shogi grandmaster in 2012 [70]. Many factors made the advances of the game-tree search algorithms possible; one of the most important elements is improving the quality of the search indicators. The history of the game-tree search algorithms is also the history of the search indicators. The advances and the achievements of the game-tree search algorithms are supported by the advances of the search indicators. In this thesis, we define the search indicator as follows.

**Definition 1.1.** *Search indicators are pointers based on the values of a node and the*

*information stored in the node during a game-tree search; the pointers are used for guiding the search. They take care of the search direction.*

For every type of game-tree search, the search indicators are the drivers for the game-tree search algorithms. In minimax tree search, the search indicator is the minimax value calculated by a given static evaluation function in each position. In MCTS, the main search indicator is a winning rate calculated by comparison after random playing. Each search indicator guides the search direction according to the purpose of the evaluation of node. There are three types of use of the evaluation function: (1) the evaluation of the position; (2) the guidance of the search by estimating the value of the position with respect to the chances to win, draw or lose the game; (3) the prediction of the outcome of the game. We discuss their all three below.

- The *evaluation* of a position is meant to represent the precise value of a position and the winning edge of the player.
  The minimax values in the minimax tree search algorithm and the winning rates in the MCTS have this characteristic. The value of a position has different means according to the purposes. If the search indicators show the advantages of a player compared to the opponent, then the search algorithms can use the search indicators for analyzing the current situation and developing a strategy. Generally, the value of a position is given by heuristics as implemented by the human players. Sometimes programmers use the results from analysing statistical data. However, to evaluate the value of a position heuristically may contain errors. These errors undermine the true value of a position and may lead to erratic moves in some cases. One of the ways to reduce the errors is to compose large game-trees and we future expectations as much as possible [68, 17]. Thus, the search indicators and its reliability are issues of our research.

- The *guidance* of the search is sometimes more important than knowing the exact evaluation of the position.
  The emphasis on guiding the search gives a specific role to the search indicators. This happens in addition to other functions. Here, we mention two specific indicators, viz. proof numbers [3] and conspiracy numbers [38]. The search indicators play only a role in guiding the search direction when expanding a game tree, i.e., the search indicators decide the ordering of visiting the nodes. If the search indicators find a winning value of a position, then the search algorithm tries to examine the position which has a winning value. In other words, the search algorithms try to develop a game tree which relies rely on the minimax value or the winning rate. In summary, the search indicators which guide the search direction support the strategy for efficient search. Some specific search indicators guide only the search direction. An example is the use of proof numbers. Proof numbers aim to solve the game, and its number show the nearness of proving a specific game outcome. In all cases, the search indicators guide the effective ordering of the visiting nodes for reaching the decided goals.

- The *prediction* of the distance in value between the current position and the specific goal is another specific way of guiding the search direction.
  Some search indicators have this role, e.g., the total search indicator $f*$ in AO* [47, 15] and the heuristic estimate $h$ in dfpn+ [42]. Generally, the main goals are winning the game, solving the game outcome, or finding a drawing strategy. If the search indicator can calculate the distance between the current position and

the specific goal perfectly, then it can find the solution. If the search indicators include some heuristic errors, but the calculated distances are almost correct, then the search algorithms can find the solution within an acceptable execution time. In contrast, if the heuristic errors are so overwhelming, then the search algorithm may be able to find the solution albeit after a long time, or it cannot find any solution at all. Thus, the quality of the search indicators in this role is quite important, and in many cases, using adequate heuristics leads to suitable values for each game. Thus, we have to consider the search indicator as carefully as possible in this case.

Above, we defined the whole range of using values (evaluation, guidance and prediction). All three uses coincide with the definition of the search indicators. In this thesis, we start focusing on three specific instances of the search indicators: proof numbers, minimax values and conspiracy numbers. These search indicators were composed for improving the performance when aimed at winning the game. This purpose has been formulated for researchers who would like to win a game or to solve a game. Still there are other goals. Three of them are improving human player skill [51], showing the explanations of the game [28] and composing a human-like computer players [11]. These three new examples of research are instrumental to compose new search algorithms by analyzing the game using the game-tree search progress. Among them, we have a particular interest in the search indicators that would shed a new light as soon as the goals intended to be reached. An example of new works is given by Khalid et al. [30]. They tried to find a new perspective of the conspiracy numbers [38] and succeeded to analyze the game progress patterns. Here, the definition and the characteristics of the original search indicators were not changed. So, Khalid et al. successfully reused the existing algorithms, methods, and techniques. Yet, we hope to propose a new utilization of the search indicators. This may imply a change of the meaning that they represent in the theory.

## 1.2 Problem Statement and Research Questions

The quality of the search algorithms is heavily influenced by the development of the search indicators, their refinement, and their adaptivity to the structure of the game-tree. The search indicators deal with the characteristics of the search algorithms and attempt to reach the goals imposed by the aim of the player (e.g., winning a game or achieving a draw). Conversely, if we focus on the search indicators then we may investigate the possibility to find a new way of the utilization. For instance, we may aim at new search algorithms which have a different utilization of the information available in a node by using the characteristics, and the purpose in a new perspective. For this new purpose, we try to reconsider the existing search indicators.

Based on these ideas, we formulate our problem statement and two research questions as follows.

**Problem Statement** To what extent is it possible to predict the behavior of the search algorithm when changing the set of search indicators?

**RQ1** What new behaviors can we expect if we change the way of searching by changing the search indicators?

**RQ2** Is it possible to anticipate on the benefits of a well thought, well defined change of the composition of the search indicators in a node?

## 1.3 Research Methodology

**RQ1** Generally, the definition and the behavior of the search indicators are decided by the original search algorithms. PN-search introduced by Allis et al. [3] uses the best-first search method using the proof numbers as the search indicator. PN-search produced many achievements, but some challenges still remain. One of the serious challenges is the seesaw effect problem [16]. This challenge is caused by using the method of the best-first search with the proof numbers. So, we may consider this challenge as an appropriate instantiation of our RQ1. We will investigate the seesaw effect problem using the proof numbers in such a way that the result in a different search behavior. Moreover, we will examine how it affects the original algorithm.

**RQ2** In the original definition, the proof numbers indicate the number of leaf nodes which need to prove a node. If we change the viewpoint somewhere, then we may state that the proof numbers show the level of difficulty for proving a node. Now, we conjecture that this viewpoint might be used for evaluating the difficulty of the puzzles (chess problems and Shogi problems). For RQ2, we analyze the mating problems of shogi (called "Tsume-shogi"). We focus on the relationship between the evaluation of tsume-shogi and its difficulty judged by the proof numbers. In this way, we try to consider the potentials of the search indicators using the different viewpoints.

Throughout our investigation process of handling the two research questions, we challenge the new potentials of the search indicators. Therefore, we consider two objectives: (1) solving the existing challenges, and (2) evaluating the puzzles. In order to achieve two objectives, we will change the behavior of the search indicators and try to consider the new viewpoint of the search indicators. We keep the original definition of the search indicators, and try to propose the new utilization. After the two challenges, we will evaluate the outcomes and formulate a summary on the potential of the search indicators.

## 1.4 Structure of the Thesis

There are 5 chapters for considering the search indicators. The comprehensive introduction and background of our research are described in Chapter 1. We formulate a problem statement and two research questions.

In Chapter 2, we introduce the existing works about the AND/OR tree search algorithms and the conspiracy number search as a background study. We then explain the search algorithms focusing on the search indicators.

In Chapter 3, we discuss RQ1. We propose how to develop new search indicators. We handle the seesaw effect using the proof numbers with the different behavior. We propose an Othello endgame solver and a Hex solver for verifying our proposal.

In Chapter 4, we discuss RQ2. We propose how to find a new meaning of the search indicators. We compare the data analysis of the tsume-shogi using the proof numbers and the evaluation that is used in ranking the competitions. We aim at confirming the usefulness of the new viewpoint of the search indicators.

Finally, in Chapter 5, based on our finding, we may conclude that the research outcomes from the related experiments. Lead to the answers of RQ1 and RQ2. Thereafter we answer the problem statement.

# Chapter 2

# Solving Game Positions and Search Indicators

In recent years, some computer-playing algorithms succeeded to calculate game-theoretical values in several games. The game-theoretical values propose the game outcomes before settling or playing. If we achieve to calculate the game-theoretical value in a game then the game is called "solved". A well known example for solving games is a result of Tic-Tac-Toe. The outcome of Tic-Tac-Toe must go to draw if both players play the game perfectly then. The basic game-playing algorithms can solve the small games, i.e., minimax tree search can solve Tic-Tac-Toe. But, the basic game-playing algorithms need too large working memory to solve the complex games. In 1994, Allis et al. [3, 4] developed a breakthrough algorithm named proof-number search (PN-search). They were inspired by the idea of conspiracy number search [38] and the AND/OR tree [1] for proposing PN-search. Development of PN-search supported to propose modified PN-search algorithms such as Df-pn [42], Dfpn+ [42], weak proof number search [63] and others [64, 5]. PN-search and its variants achieved to solve the complex games such as Checkers [55], Connect Four [2] and 6x6 Othello [59].

In this chapter, we explain the related works. We focus on the search algorithms related to solving games and its search indicators. We aim to understand the advances of the search indicators by studying the history of the game solving algorithms. Firstly, we explain the important components of the game solving algorithms. There are two components; the AND/OR tree and the basic search algorithms. Secondly, we show the existing algorithms with a focus on the AND/OR tree search and its search indicators. Then, we try to consider a new utilization of the search indicators. We hypothesize that studying the details of the search indicators supports to give new perspective. We focus on the definition, characteristics and effectiveness of the search indicators. Finally, we explain the researches which challenge to apply the new perspective of the search indicators to the practice. These researches use the conspiracy number search and its search indicator for analyzing the game progress patterns. We try to study the way of finding new perspective of the search indicators and making it practicable.

## 2.1 Solving Game Positions

If we focus on a position, consider next legal moves, and expect future positions, then we can build up a graph using the positions connected by the moves (called game tree).

If we explore all positions in a game tree and analyze it, then we can solve a target game, i.e., we can expect the game outcome (winning, drawing or losing) before playing. However, complex games have large search space and we need too large working memory to explore all positions. So, primitive searching algorithms cannot solve the complex games in real time. For this problem, some algorithms and corresponding techniques were proposed. The search algorithm using AND/OR tree (called AND/OR tree search) is one of the reasonable approaches for tackling the problem. In this section, we explain the AND/OR tree search algorithms and corresponding techniques.

### 2.1.1 AND/OR Tree

AND/OR tree [1] is a basic framework when solving games. Basically it is composed by two types of nodes which have a function like mathematical logic and store logical values for each node. Two types of nodes are AND node and OR node. We show, in Figure 2.1, an illustration of the OR node in AND/OR tree.

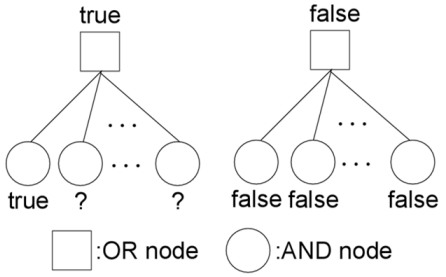



Figure 2.1: OR node in AND/OR tree

The OR node represents a position where the first player is to move. Each node in AND/OR tree has three types of values: *true*, *false* and *unknown*. *True* means the winning, mating or satisfying a given specific purpose, and its contrast value is *false*. *Unknown* means that the value of a target node cannot be decided. Deciding the value of an OR node to *true* needs to have at least one true child node as shown at the left side of Figure 2.1. If the values of all children nodes are *false* then the value of the OR node is false as shown at the right side of Figure 2.1. So, the behavior at OR nodes is similar to the OR function in mathematical logic.

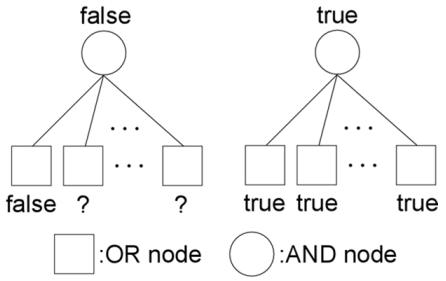



Figure 2.2: AND node in AND/OR tree

The AND node corresponds to a position where the opponent or the second player is to move. If at least one child node shows *false* then the value of the AND node is

decided on *false* as shown at the left side of Figure 2.2. If all children show *true* then the value of the AND node is decided on *true* as shown at the right side of Figure 2.2. So, the behavior at AND nodes is similar to the AND function in mathematical logic.

Moreover, AND/OR tree has four types of additional node names as shown in Figure 2.3, which are the root node, terminal node, leaf node and internal node.

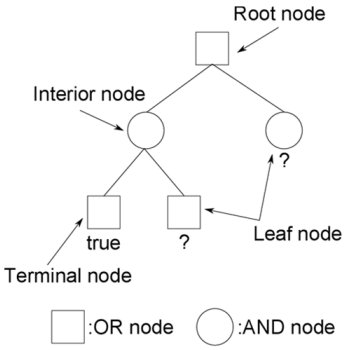



Figure 2.3: Four types of additional node names in AND/OR tree

- **Root node** is the topmost node in an AND/OR tree. The root node is OR node generally. Solving an AND/OR tree means that the value of the root node is decided on *true* or *false*.

- **Terminal node** is any node that does not have child nodes. The terminal node represents a position that completes the game. So, the value of a terminal node is true or false absolutely.

- **Leaf node** is any node that does not have child nodes and its value is not yet decided. So, the value of the leaf node is *unknown*. If the value is decided then the leaf node turns out to be the terminal node.

- **Internal (interior, inner) node** is any node that has child nodes. The value of an internal node is decided in the way that is shown in Figure 2.1 and Figure 2.2.

Each node of AND/OR tree can be identified by its value. The node which stores *true* value is called a proven node, whereas the node which stores *false* value is called a disproven node. The tree which satisfies the following conditions is called the proof tree [31].

1. The root node is in the proof tree.

2. For each interior OR node in the proof tree, at least one child of the OR node is in the proof tree.

3. For each interior AND node in the proof tree, all the children of the AND node are in the proof tree.

4. All terminal nodes in the proof tree are proven.

So, the value of the root node in the proof tree is *true*.

### 2.1.2 Basic Search Algorithms

We explain three basic search algorithms: depth-first search, best-first search and depth-first iterative deepening search. These search algorithms can be applied to a general game tree. However, we restrict ourselves to the behavior in the framework of AND/OR tree in this section. These search algorithms do not have any specific search indicators. Usually, the main elements of search algorithms include the type of game tree, search indicators and the basic search algorithm. So, the fundamental behavior of the search algorithms is determined.

#### Depth-First Search

Depth-first search (DFS) [50] is a search algorithm that selects one branch and pursues down until reaching a terminal node or a leaf node on maximum depth. After reaching a terminal node or leaf node, DFS backs up the value obtained to the most recent previous node, and continues to pursue the process repeatedly. In some case, we may need to set a maximum depth to stop the search. For maximum depth $d$ and the number of average branching factor $b$ in the AND/OR tree, the amount of calculation costs is $O(b^d)$ in the case where there is no search efficiency. Thus, for large value $d$, we have to take a risk that we may not complete the search within a limited time. On the other hands, if we set a small value $d$, then DFS may not be able to establish a proof tree. Moreover, the move ordering of child nodes in a node largely affects the search efficiency. Setting an appropriate maximum depth and the effective move ordering of the child nodes are the important issues when using DFS.

#### Best-First Search

Best-first search (BFS) [50] is a search algorithm that tries to alleviate the disadvantages of DFS. While searching, BFS selects the most promising node among all expanded nodes and examines it. A promising node may be identified by some heuristics like search indicators. For example, static evaluation function [58], proof number [2] and winning rate [8] are such search indicators. If the heuristic estimation is sufficiently accurate, then the search tree built by BFS could be smaller than the search tree by DFS. However, BFS needs to keep all unevaluated nodes in the working memory. So, there is a risk that it may suffer from an explosion of working memory. BFS has to continue the processes to find the most promising node, while updating the search indicators with a focus on a promising node and the root node. So, the calculation cost at each iteration cannot be ignored. These two problems depend on the accuracy of the heuristics estimation.

#### Iterative Deepening Depth-First Search

Iterative deepening depth-first search (IDDFS) is a search algorithm in which a depth-limited version of depth-first search is run repeatedly with increasing depth limits until the goal is found or the proof tree is established. So, IDDFS has the advantage compared to DFS in the sense that the best maximum depth is determined. Some search algorithms consider the way of increasing the depth limit for reducing the extra costs to reexpand and revisit the nodes explored. For example, Df-pn [42] uses a threshold of search indicators instead of the depth. This modification enables to improve the performance of original IDDFS.

## 2.2 AND/OR Tree Search using Proof Numbers

Mating problems in shogi (tsume-shogi) has been popular in Japan for the challenging task of solving and composing. Solving tsume-shogi is a good application for AND/OR tree search. A mating problem gives three outcomes: checkmate, cannot checkmate and unknown. These outcomes correspond to *true*, *false* and *unknown* in AND/OR tree framework, respectively. So, mating problems can take the AND/OR tree structure. In tsume-shogi, the average number of branching factors is about 5 [57]. It is much smaller than the average branching factors of shogi [20]. Nevertheless, mating problems which has 17 plies or more cannot be solved by simple brute-force search algorithms such as $\alpha-\beta$ algorithm [34]. Advanced AND/OR tree search algorithms overcome this problem and achieved a significant result by solving very difficult mating problems including one which has 1525 plies [57]. In this section, we give a brief sketch of AND/OR tree search algorithms using proof-numbers and related search indicators.

### 2.2.1 Proof Number Search

Proof Number Search (PN-search or PNS) is a best-first AND/OR tree search algorithm for finding the game-theoretical value in game trees [3, 4]. PN-search uses two search indicators: proof number and disproof number. The proof number implies the minimum number of unsolved leaf nodes which need to be proven in order to win in the root node. Similarly, the disproof number is the minimum number of unsolved leaf nodes which need to be disproven in order to lose in the root node.

In the terminal node $N$, the proof number (pn) and disproof number (dn) are decided as follows:

$$(N.pn, N.dn) = \begin{cases} (0,\infty) & \text{if N is True} \\ (\infty,0) & \text{if N is False} \\ (1,1) & \text{otherwise(N is unknown)} \end{cases}$$

In the internal node $N$, the value of $N$ is calculated by own children (say $C$) as follows:

$$(N.pn, N.dn) = \begin{cases} (\min\limits_{c\in C} c.pn, \sum\limits_{c\in C} c.dn) & \text{if N is an OR node} \\ (\sum\limits_{c\in C} c.pn, \min\limits_{c\in C} c.dn) & \text{if N is an AND node} \end{cases}$$

Note that the value of the root node can be calculated recursively from the terminal nodes. We show, in the following Pseudo-code 1 and 2, the algorithm of PN-search. The value $\nu$ represents the result of PN-search. The most_proving_node ($N$) is a function which returns the next search node. This function is defined in Pseudo-code.

The most proving node means the easiest node for proving or disproving. PN-search always considers the most proving node, because if the value of a terminal node is decided then the value of internal nodes can be decided recursively. Therefore, PN-search can be used to decide the value of the root node by examining the values of other nodes as soon as possible.

In PN-search, important search indicators are the proof number and disproof number. These two indicators guide the search direction while supporting to find the

**Pseudo code 1** Proof Number Search

```
create_root(root)
while proof(root) ≠ 0 and disproof(root) ≠ 0 do
    most_proving := most_proving_node(root)
    expand_node(most_proving)
    update_proof_numbers(most_proving)
end while
if proof(root) = 0 then
    return ν
else
    return ¬ν
end if
```

**Pseudo code 2** most_proving_node(J)

```
while is_an_internal_node(J) do
    if or_node(J) then
        J := leftmost_child_with_equal_proof_number(J)
    else
        J := leftmost_child_with_equal_disproof_number(J)
    end if
end while
return J
```

most proving nodes during the search. We understand that PN-search is the best first AND/OR tree search using proof and disproof numbers, and that the meaning of the proof and disproof number is the size of a subtree which includes a most proving node. PN-search refers to this size of each subtree, and tries to solve the target problem with the smallest effort. Thus, the proof and disproof number represent important characteristics of a target position while showing the difficulty of solving it. From another point of view the proof and disproof number represents the complexity of a target problem, which may relate to the time necessary for solving it. Finally, the effectiveness of using the proof and disproof numbers has been confirmed. PN-search and its variants have successfully solved several games. Indeed, such great achievements are well known evidences of the effectiveness of these search indicators.

### 2.2.2 Related Algorithms

We overview several related search algorithms in the framework of AND/OR tree search: AO*, PN* and PDS. AO* was proposed in 1980, which is a different approach to solving games. The idea of AO* was inspired from the optimization research. Later, the idea of AO* is referred as modified PN-search. PDS and PN* are the depth-first variants of PN-search. PDS was proposed in 1998 by Nagai [40], whereas PN* was originally proposed in 1995 by Seo [56] and later renamed as PN*.

#### AO*

AO* [44] was derived from A* algorithm which is a generalized Dijkstra's algorithm. A* is able to find a goal started from a certain position, i.e., A* can solve graph search

problems. If we separate a whole graph into sub-graphs to solve the sub-problems like "divide and conquer algorithm", then an original graph can compose an AND/OR tree. For solving the AND/OR tree, AO* was modified from A*. This algorithm uses three functions: $f(n)$, $h(n)$ and $c(n,m)$. $f(n)$ is a cost function, $h(n)$ is a heuristic function and $c(n,m)$ is a non-negative cost between node $n$ to $m$. For example, in minimax tree search, $f(n)$ corresponds to the minimax value, and $h(n)$ corresponds to evaluation function. Thus, we note that AO* is a best-first minimax search algorithm using $f(n)$. Below we show the algorithm of AO*.

1. If n is a terminal node then

$$f(n) = \begin{cases} 0 & \text{if n is True} \\ \infty & \text{if n is False} \end{cases}$$

2. If n is a leaf node then

$$f(n) = h(n)$$

3. If n is an inner OR node then

$$f(n) = \min_k (f(n_1) + c(n,n_1), \cdots, f(n_k) + c(n,n_k))$$

4. If n is an inner AND node then

$$f(n) = \sum_{i=1}^{k} (f(n_i) + c(n,n_i))$$

A heuristic function $h(n)$ should be given by designers. This function shows the distance between a target node and the goal. If $h(n)$ contains the perfect knowledge, then AO* can find a solution without "searching". In contrast, if $h(n)$ is poor, then AO* would need very long time for solving.

The search indicators in AO* are the total search indicator $f$, the heuristic function $h$ and the cost $c$. We try to understand the meaning of each search indicator. The heuristics function $h$ shows the distance between a target node and the goal. The definition of the goal depends on the target domain. For example, if we set the goal to solve a game, then we consider the heuristic function $h$ in a node as the likeliness for proving or disproving. The cost $c$ shows the distance between the connected two nodes. As an example, the cost $c$ shows the difficulty or the potential of moving between the two nodes. For solving games, the cost $c$ can be used to control the search direction such as DFS and BFS. Finally, the total search indicator $f$ is composed by the heuristic function $h$ and the cost $c$. The characteristics, the purpose and the effectiveness of AO* therefore depend on the total search indicator $f$ and the target domains.

**PN***

PN* algorithm was proposed as C* by Seo in his master thesis [56] and later its revised version as PN* appeared in [57]. This algorithm successfully manages less working memory than AO*. It is supposed that PN* is a special case of AO*. It means that PN* removes cost function $c(n,m)$, while using proof number $h(n)$ and employing IDDFS from AO*. Then, the proof number on PN* algorithm has the same meaning as

the proof number on PN-search. Generally, IDDFS uses the depth limit as threshold, whereas PN* employs the proof number as its threshold.

First, PN* starts from the root node with the proof number threshold $th_{pn} = 1$. In OR node, PN* expands child nodes, and visits each child node one by one. Then, each child node is given the same threshold as a parent node, i.e., they are given $th_{pn} = 1$ in this case. After visiting all child nodes, PN* records the proof number of the current OR node, and back to focus on a parent node. In AND node, PN* expands child nodes. Next, PN* calculates the proof number of the current node, and compare it with the proof number threshold. If the proof number of the current AND node is less than the proof number threshold, then PN* visits each child node one by one. Then, the proof number threshold in each child node is the same value as the proof number in each child node. In contrast, the proof number in the current AND node shows over the proof number threshold, then PN* records the proof number and back to focus on a parent node. If PN* goes back to the root node, then PN* examines the proof number of the root node. If the proof number of the root node shows 0 or infinity value, then PN* completes its searching and shows the outcome. Otherwise, PN* increases the proof number threshold to the same value as the proof number of the root node, and continues to search repeatedly.

For reducing the working memory, PN* employs transposition tables [13]. The transposition tables store all nodes which have been visited during the previous search, and share information of the nodes which meet the same situation. While searching, PN* looks over the transposition tables before examining the nodes. If the transposition tables contain information of the target nodes, then PN* takes the proof number without calculating and searching. If the transposition tables do not have information of the target nodes, then PN* initializes the target nodes as $pn = 1$. PN* successfully reduces the calculation costs and working memory using the ideas described above. PN* then outperformed AO*. The search indicator of PN* is the proof number, while the definition of this indicator is the same as PN-search. So, the meaning and the characteristics of the search indicator are the almost same as PN-search. As for the difference of the proof number between PN-search and PN*, we point out the trend of the proof number. PN* uses the proof number only, and then it focuses on the difficulty for proving, but not for disproving. Hence, there are some differences such as the largest proof number during the search. These values may provide the different meaning and characteristics from PN-search, e.g., the difficulty of solving a mating problem to devote its purpose to finding mate only.

### PDS

PDS was proposed by Nagai [40], which was inspired by Seo's algorithm [56] that uses proof numbers only. PDS uses both proof numbers and disproof numbers. In addition, PDS employs proof and disproof number thresholds and transposition tables. Thus, PDS outperforms PN-search. Here we mention four differences between PN* and PDS. (1) PDS continues to search until reaching the proof and disproof number thresholds. (2) PDS starts IDDFS from OR node and AND node. (3) The proof number and disproof number of each node are stored in the transposition table of PDS. (4) PDS calculates both thresholds while applying Equations (2.1) and (2.2).

$$n.\phi = \begin{cases} pn(n) & \text{n is an OR node} \\ dn(n) & \text{n is an AND node} \end{cases} \quad (2.1)$$

$$n.\delta = \begin{cases} dn(n) & \text{n is an OR node} \\ pn(n) & \text{n is an AND node} \end{cases} \quad (2.2)$$

Where $pn(n)$ and $dn(n)$ stand for the proof number and disproof number of a node $n$, respectively. Moreover, $n.\phi$ and $n.\delta$ denote the proof and disproof number thresholds in the node $n$, respectively. In each iteration, PDS increments a threshold. If $n.\phi \leq n.\delta$, then PDS increments $n.phi$; otherwise increment $n.\delta$.

The experiments performed with Othello show that PDS outperformed AO*, PN* and even PN-search [40]. The search indicators of PDS are the same as PN-search, but they are controlled by the threshold $n.\phi$ and $n.\delta$. The definition of the search indicators are similar to the PN* and PN-search. However, the trend of the proof and disproof numbers during the search may be different from PN* and PN-search. This means that there are some potentials that we can get new benefits different form PN-search.

### 2.2.3 Depth-First Proof-Number Search

Depth-First Proof-Number Search (Df-pn) is a depth-first variation of PN-search, which was invented by Nagai [42]. Df-pn employs the proof and disproof numbers as the same as PN-search, whereas the algorithm of Df-pn is more similar to PDS. Df-pn employs IDDFS, the proof and disproof number thresholds and transposition tables. Below the details of Df-pn algorithm are shown.

1. The thresholds of proof number and disproof number on the root node $r$ are calculated in the following way.

$$r.th_\phi = \infty$$
$$r.th_\delta = \infty$$

2. In each node $n$, DF-pn continues to search beneath $n$ until $n.\phi \leq n.th_\phi$ or $n.\delta \leq n.th_\delta$ is satisfied (Nagai called it ending condition).

3. In each node $n$, select the child $n_c$ with minimum $\delta$ and the child $n_2$ with second minimum $\delta$. (If there is another child with minimum $\delta$, that is $n_2$.)
Search beneath $n_c$ while applying Equations (2.3) and (2.4).

$$n_c.th_\phi = n.th_\delta + n_c.\phi - \sum n_{\text{child}}.\phi \quad (2.3)$$
$$n_c.th_\delta = \min(n.th_\phi, n_2.\delta + 1) \quad (2.4)$$

   Repeat this process until the ending condition holds.

4. If the ending condition is satisfied, the search process returns to the parent node of $n$. If $n$ is the root node, then the search is totally over.

Where $n.th_\phi$ and $n.th_\delta$ denote the thresholds of proof and disproof numbers, respectively. Both variables play the roles of the proof and disproof number threshold according to the type of node considered. Both thresholds in node $n$ are calculated by

Equations (2.1) and (2.2). PDS increments both thresholds in each iteration, whereas Df-pn calculates both thresholds using Equations (2.3) and (2.4). Equation (2.3) keeps a threshold value which is calculated at its parent node. Equation (2.4) chooses a smaller value than a threshold value of the current node and $\delta$ value of a child node which has the second smallest $\delta$.

Nagai [42] shows that Df-pn behaves almost the same as PN-search, whereas Df-pn has an advantage in taking a lower cost of visiting nodes than PN-search due to the characteristics of IDDFS, i.e., Df-pn does not go back to the root node in every iterations. Additionally, Df-pn employs transposition tables while searching for reducing the working memory. The merit of Df-pn enables to leads to the second breakthrough, and then Df-pn has become the front-runner algorithm for solving games. The search indicators in Df-pn are the same as PN-search. So, we assume that we can reuse the conclusions of PN-search for Df-pn.

### 2.2.4 Derivation of Depth-First Proof-Number Search

In this section, we present two variants of search algorithms derived from Df-pn: Weak proof number search (WPNS) [63] and Dfpn+ [42]. WPNS is a modification of Df-pn with an enhancement to tackle the so-called *double counting problem*. For the purpose WPNS incorporates a new search indicator named *weak proof number*. Dfpn+ is an enhanced version of the original Df-pn. Dfpn+ incorporates two new search indicators: heuristic function $h$ and cost function *cost*, by which the performance was successfully improved. Below we study some more details of the two search algorithms.

#### Weak Proof Number Search

Df-pn is one of the most efficient AND/OR tree search algorithms, but some problems still remain. One problem is the double counting problem that is caused by the directed acyclic graphs (DAGs). WPNS [63] tackles this problem while incorporating a new search indicator *weak proof number* to be more efficient than Df-pn especially in some specific case like DAG. Df-pn employs the transposition table for search efficiency. The transposition table stores information of the nodes visited, and shares with the nodes having the same hash value. So, the transposition table can reduce the number of nodes which are expanded, because the nodes which have the same hash value are unified into one node. The transposition table supports to search efficiently, because the search algorithms can reuse the information of nodes. However, this efficient mechanisms occur some problems. One of the problems is double counting problem. Assuming that a parent node $A$ has two child nodes and that two child nodes share a child node $B$, then node $A$ looks holding two nodes $B$. Then, node $A$ counts up the search indicators as much as the number of nodes $B$ which looks holding. However, node $B$ exists only one in the game tree. So, this multiple counting in node $A$ is wrong, which is called double counting problem.

WPNS is a modified version of Df-pn for solving the problem, inspired by the branch number search (BNS) [45]. WPNS uses the proof number and number of unsolved child nodes. Below, we show the algorithm of WPNS.

1. If n is a terminal node then

$$n.\phi = \begin{cases} 0 & \text{n is an OR node} \\ \infty & \text{n is an AND node} \end{cases}$$
$$n.\delta = \begin{cases} \infty & \text{n is an OR node} \\ 0 & \text{n is an AND node} \end{cases}$$

2. If n is a leaf node then

$$n.\phi = 1$$
$$n.\delta = 1$$

3. If n is an inner node whose successor nodes are $n_i (1 \leq i \leq K)$ then

$$\begin{aligned} n.\phi &= \min_{1 \leq i \leq K} n_i.\delta \\ n.\delta &= \max_{1 \leq i \leq K} n_i.\phi + (k-1) \end{aligned} \tag{2.5}$$

Note that Equation (2.5) makes WPNS different from PN-search and Df-pn. Df-pn sums up $\phi$ values of child nodes when calculating $n.\delta$. On the other hands, in the algorithm of WPNS $n.\delta$ is calculated by a maximum $\phi$ value among all child nodes and the number of unsolved child nodes. This enhancement supports to mitigate the double counting problem. In the domain of Othello, there are the same positions around endgame, so Df-pn often faces the double counting problems. For this reason, WPNS outperforms Df-pn in Othello endgame positions. Meanwhile, in tsume-shogi, the frequency of the double counting problems depend on each problem. So, the results of WPNS are mixed with better and worse cases. We consider that WPNS is a solver fitting to the double counting problems.

The search indicators of WPNS are the proof number, disproof number and the proof number plus the number of unsolved nodes (called weak proof number). The meaning of the proof and disproof number is the size of the subtree which needs to prove or disprove for solving a node. The weak proof number sums up the maximum proof number in the unsolved child nodes and the number of unsolved child node minus 1. In AND node, the weak proof number shows almost the same value as the most difficult node for proving among all the child nodes. The number of unsolved child nodes is used for the tie-break and identifying the approximate size of the subtree. Thus, the characteristics of the search indicators in WPNS show the most complexity situation for proving or disproving a node. From another point of view, the weak proof number shows the limitation of the difficulty for solving a node.

#### DFPN+

Dfpn+ [42] is another enhanced version of Df-pn. Dfpn+ is compatible with Df-pn perfectly. On the one hand, Df-pn is a really generic algorithm for AND/OR tree search. On the other hand, Df-pn may not always fit to each game as sufficiently efficient search. Dfpn+ fixed this weak point and tried to add extensibility. Dfpn+ employs two new search indicators while searching: $h_{\text{(dis)proof(n)}}$ and $cost_{\text{(dis)proof}}(n)$. $h_{\text{(dis)proof(n)}}$ associates with the distance between node $n$ and (dis)proof solutions. This

is an alternate value of proof and disproof numbers, and it shows the heuristic estimate decided by the programmers. In practical, $h$ function shows the amount of likeliness to prove (or disprove) the nodes. Moreover, $cost_{\text{(dis)proof}}(n, n_{child})$ shows the cost from node $n$ to node $n_{\text{child}}$. This function is decided by programmers and it enhances the (dis)proof number thresholds used in the original Df-pn. Dfpn+ has some functions in addition to $h$ and $cost$.

First we show the negamax algorithm.

1. In an OR node $n$

$$\begin{cases} h_\phi(n) = h_{proof}(n) \\ h_\delta(n) = h_{disproof}(n) \\ cost_\phi(n) = cost_{proof}(n) \\ cost_\delta(n) = cost_{disproof}(n) \end{cases}$$

2. In an AND node $n$

$$\begin{cases} h_\phi(n) = h_{disproof}(n) \\ h_\delta(n) = h_{proof}(n) \\ cost_\phi(n) = cost_{disproof}(n) \\ cost_\delta(n) = cost_{proof}(n) \end{cases}$$

Secondly the values of each node are decided in the following ways.

1. If node $n$ is a terminal node then

$$\begin{aligned} n.\phi &= \begin{cases} 0 & \text{n is an OR node} \\ \infty & \text{n is an AND node} \end{cases} \\ n.\delta &= \begin{cases} \infty & \text{n is an OR node} \\ 0 & \text{n is an AND node} \end{cases} \end{aligned}$$

2. If node $n$ is a leaf node then

$$\begin{aligned} n.\phi &= h_\phi(n) \\ n.\delta &= h_\delta(n) \end{aligned}$$

3. If node $n$ is an inner node then

$$\begin{aligned} n.\phi &= \min_{n_c \in \text{children of} n} (n_i.\delta + cost_\phi(n, n_i)) \\ n.\delta &= \sum_{n_c \in \text{children of} n} (n_i.phi + cost_\delta(n, n_i)) \end{aligned}$$

Finally, we show the whole algorithm of Dfpn+.

1. The threshold of proof number and disproof number on root node $r$ are calculated as follows:

$$\begin{aligned} r.th_\phi &= \infty \\ r.th_\delta &= \infty \end{aligned}$$

2. In each node $n$, the search process continues to search beneath $n$ until $n.\phi \leq n.th_\phi$ or $n.\delta \leq n.th_\delta$ is satisfied (ending condition).

3. In each node $n$, select the child $n_c$ with minimum $(n_i.\delta + cost_\phi(n,n_i))$ and the child $n_2$ with second minimum one. (If there is another child with minimum one, that is $n_2$.)
   Search beneath $n_c$ with assigning

$$n_c.th_\phi = n.th_\delta + n_c.\phi - \sum(n_i.\phi + cost_\delta(n,n_i)) \quad (2.6)$$
$$n_c.th_\delta = \min(n.th_\phi, n_2.\delta + cost_\phi(n,n_2) + 1) - cost_\phi(n,n_c) \quad (2.7)$$

   Repeat this process until the ending condition holds.

4. If the ending condition is satisfied, the search process returns to the parent node of $n$. If $n$ is the root node, then the search is totally over.

Dfpn+ employs $h$ and *cost* instead of the proof and disproof numbers. If Dfpn+ uses the following conditions, then Dfpn+ works the same as original Df-pn.

$$cost = 0$$
$$h = 1$$

The effectiveness of Dfpn+ depends on $h$ and *cost*. If both functions are fitted to target game correctly, then Dfpn+ works better than Df-pn. But, Dfpn+ has the same issues as AO* about the heuristics estimate $h$. Programmers have to note the design of $h$ as much as possible.

The search indicators of Dfpn+ are the $h$ and *cost*. The meaning of both values depends on the kind of target games. In Othello, the original proof number and the evaluation function were used for composing $h$ [42]. In this case, $h$ has the characteristics, e.g., showing the more likeliness value than the proof and disproof numbers for proving and disproving. From another viewpoint, we assume that $h$ expects the game outcome more precisely than the proof and disproof numbers. In Othello, *cost* for visiting the deep node in the AND/OR tree was utilized. The characteristics of *cost* is to control the behavior of the search algorithms. So, we assume that *cost* relates to the shape of the proof tree.

## 2.3 New Perspective of Conspiracy Numbers

Conspiracy Number Search (CNS) [38] was invented as a game-independent best-first search which expands the game tree non-uniformly to establish a stable value in the root node. The algorithm was based on the concept of conspiracy numbers which, in a sense, show how unlikely the root value would change to a certain value. Conspiracy numbers were also used in alpha-beta-conspiracy [39]. Theirs usage were different and less computationally intensive. However, both algorithms were not very successful [52], and conspiracy numbers did not receive much attention after that. Khalid et al. [30] revisited the conspiracy numbers for identifying the critical positions, and focused on the trend of the conspiracy numbers during a game for analyzing game patterns [29]. In this section, we explain CNS and new approaches. They might give us a new insight into the research questions in this study.

### 2.3.1 Conspiracy Numbers

Conspiracy numbers of the root or interior nodes of a search tree for some value $v$ are defined as the least number of conspirators, that are leaves that must change their evaluation value to $v$ in order to change the minimax value of the interior node or root [65]. Conspiracy numbers are calculated by the number of leaf nodes, the evaluation function, and minimax values. Conspiracy number shows the difficulty for changing the current minimax value in a node to target value. Large conspiracy number means that changing the current minimax value of a node would require the 'conspiracies' between many nodes, and small conspiracy number would mean otherwise. For example, if conspiracy number of the root node is 3, then we have to examine 3 leaf nodes for changing the minimax value of the root node. We show, in Table 2.1 and Figure 2.4 [36], definitive illustrations of conspiracy numbers.

Table 2.1: An example of conspiracy numbers in the root node

| Value | CN | Nodes to Change |
|---|---|---|
| -3 | 2 | (E and (F or G)) |
| -2 | 2 | (E and (F or G)) |
| -1 | 2 | (E and (F or G)) |
| 0 | 1 | (E or J) |
| 1 | 0 | |
| 2 | 1 | (J or K) |
| 3 | 2 | (E and (J or K)) or (F and G) |

Figure 2.4 shows a simple minimax tree and conspiracy numbers. In this game tree, each node has a minimax value between $-3$ to 3 (left side values) and there are conspiracy numbers in each node (right side values). Minimax values are shown inside the nodes. Table 2.1 shows the conspiracy number of the root node in Figure 2.4 (the values shown beneath the nodes). In the root node, if we want to change minimax value 1 to $-3$, then the conspiracy number is 2. This 2 means that we have to check at least 2 leaf nodes ($E$ and ($F$ or $G$)) in the game tree for changing the current score to $-3$. The minimax value of the root node is given by children. If we try to change the minimax value of the root node to $-3$, then we have to change nodes $B$ and $C$ to $-3$ or less. Similarly, the minimax values of nodes $B$ and $C$ are given by children, so we also have to change the minimax value of these child nodes to $-3$. In this case, nodes $E$ and ($F$ or $G$) are chosen as the conspiracy nodes, and the conspiracy number is decided on 2. Additionally, if we try to change the minimax value of the root node to 1, then the conspiracy number is 0. Because, we do not need to examine leaf nodes for changing the minimax value of the root node.

Below we show how to calculate conspiracy numbers.

$$CN(T,v) = \begin{cases} 0 & \text{if } v = m \\ 1 & \text{if } v \neq m \\ \infty & \text{if terminal node} \end{cases}$$

Where $m$, $v$ and $T$ denote a minimax value of a node, a target value and a leaf node, respectively. If the node $T$ does not need to change own minimax value, then the conspiracy number is 0, otherwise, conspiracy number is 1. If the node $T$ is a terminal

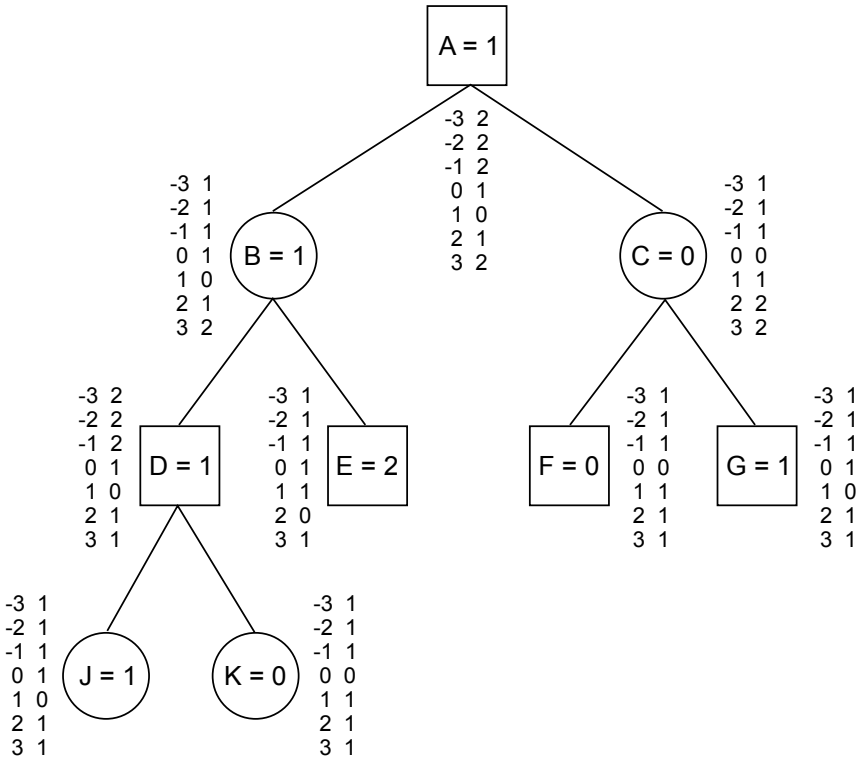



Figure 2.4: An example of minimax tree and its conspiracy numbers

node, i.e., the value of the terminal node $T$ is decided, then we cannot change its value anymore, so the conspiracy number of the terminal node $T$ is $\infty$.

In an inner node, the conspiracy number is calculated by its child nodes. If the target value of $v$ is larger than the current minimax value $m$, then conspiracy number is calculated in the following way.

$$\uparrow CN(T,v) = \begin{cases} 0 & \text{for all } \ v \leq m \\ \min_{\text{all child nodes}_i} \uparrow CN(T_i,v) & \text{for all } \ v > m \end{cases}$$

Where $\uparrow CN(T,v)$ means the conspiracy number in a node $T$ which focuses on increasing minimax value $m$ to $v$. If we try to change the current minimax value to a larger value, then one child node has to change its value to $v$. So, the conspiracy number of a max node tries to get the minimum conspiracy number in its child nodes. If the current minimax value $m$ is the same or larger than the target value $v$, then the conspiracy number of an inner node $T$ is 0.

Otherwise, if the value of $v$ is smaller than $m$, then the conspiracy number is calculated in the following way.

$$\downarrow CN(T,v) = \begin{cases} 0 & \text{for all } \ v \geq m \\ \sum_{\text{all child nodes}_i} \downarrow CN(T_i,v) & \text{for all } \ v < m \end{cases}$$

Where $\downarrow CN(T,v)$ means the conspiracy number of an inner node $T$ which focuses on decreasing minimax value $m$ to $v$. If we try to change the current minimax value $m$

to smaller value, then it has to change all child nodes which have a larger minimax value than $v$. So, the conspiracy number is calculated by the summation of conspiracy numbers of child nodes. If $v$ is larger than $m$, then the conspiracy number of an inner node $T$ is 0.

In a min node, the conspiracy number are calculated in the following way.

$$\uparrow CN(T,v) = \begin{cases} 0 & \text{for all } v \leq m \\ \sum_{\text{all child nodes}_i} \downarrow CN(T_i,v) & \text{for all } v > m \end{cases}$$

$$\downarrow CN(T,v) = \begin{cases} 0 & \text{for all } v \geq m \\ \min_{\text{all child nodes}_i} \uparrow CN(T_i,v) & \text{for all } v < m \end{cases}$$

The conspiracy numbers have a characteristic of monotonicity property. If $v < w$, then $\uparrow CN(T,v) \geq \uparrow CN(T,w)$ and $\downarrow CN(T,w) \leq \downarrow CN(T,v)$. These characteristics can be seen in Figure 2.4 and Table 2.1. The set of conspirators $CN(T,v)$ can conspire the value of the root node between $v$ and $m$. For example, a node $E$ can conspire a value of the root node between 1 and $-3$. Pseudo-code of calculating conspiracy numbers is given in Algorithm 3.

### 2.3.2 Conspiracy Number Search

Conspiracy Number Search (CNS) is a search algorithm which tries to guarantee the accuracy of the minimax value of a certain node. The minimax value is calculated by the domain-dependent evaluation function composed by programmers, so the estimation of the positions may contain some errors. One of the solutions for reducing these errors is to build up large game tree [68], but we need to prepare large working memory. Another approach is to use the conspiracy numbers. The conspiracy numbers can show the stability of the minimax value, and CNS tries to increase this stability and guarantee the minimax value of a node.

The nodes can take a specific range of minimax values according to its conspirators. For example, node $A$ in Figure 2.4 can take the minimax values between 0 and 2 in this case. When we put the lowest value in this range to $t_{\min}$, the highest value in this range to $t_{\max}$ and the value of the root node to $t_{\text{root}}$, then the relationship between these values is shown below.

$$t_{\min} \leq t_{\text{root}} \leq t_{\max}$$

In a node, if the range of $[t_{\min}, t_{\max}]$ is wide and its conspiracy number is small, then the minimax value of a node lacks much stability and we cannot trust it even though the value fits so well. Thus, the purpose of CNS is described below.

1. Changing the range of $[t_{\min}, t_{\max}]$ to narrow
2. Increasing the conspiracy number more than a conspiracy threshold (CT) defined by programmers

If $CN(T,v) < CT$, then we put $v$ to a likely value. And, we put $[t_{\min}, t_{\max}]$ to a range of likely values in a node. The function (1) tries to fix a likely value. If CNS could successfully fix a likely value, then the minimax value in a node has strong stability. However, for satisfying this situation, CNS might have to expand too much leaf nodes. So, function (2) gives the second goal to CNS. If a conspiracy number shows less value than CT, then we consider that the minimax value of that node has acceptable stability.

**Pseudo code 3** Counting Conspiracy Numbers

```
procedure CN(Node J, int v)
    int c
    /*V shows the current minimax value of a node J*/
    if V(J) == v then
        c := 0
    else if isTerminal(J) then
        c := ∞
    else if isLeaf(J) then
        c := 1
    else if isMaxNode(J) && v < V(J) then
        c := 0
        for all child nodes j in J do
            if v < V(j) then
                c := c + cn(j, v)
            end if
        end for
    else if isMinNode(J) && v > V(J) then
        c := 0
        for all child nodes j in J do
            if v > V(j) then
                c := c + cn(j, v)
            end if
        end for
    else
        c := ∞
        for all childe nodes j in J do
            c = min(cn(j, v), c);
        end for
    end ifreturn c
end procedure
```

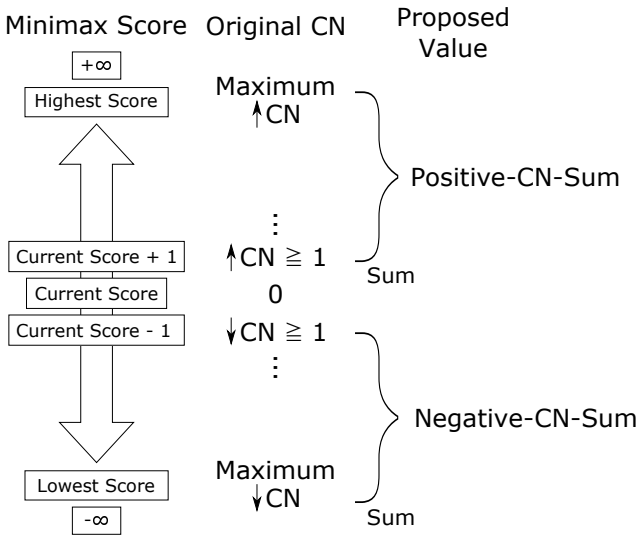



Figure 2.5: An illustration of Positive_CN_Sum and Negative_CN_Sum

CNS expands leaf nodes according to these two goals, and try to reduce the heuristic errors in the evaluation function. Until today, $\alpha - \beta$ search using the conspiracy numbers called $\alpha - \beta$ conspiracy number search [39] and a modified CNS called Controlled Conspiracy Number Search (CCNS) [37] were proposed.

### 2.3.3 Analyzing Games with Conspiracy Numbers

CNS and the conspiracy numbers were applied to building strong computer players at the earlier stage. Due to the expensive calculation costs, these research domains changed to be passive. Recently, some studies focus on the conspiracy number from the different viewpoint. One research proposed a way of finding the important game situation [30], whereas another research proposed a way of analyzing the game patterns using the conspiracy numbers [29]. We assume that both researches give some insightful feedback into our research questions. So in this section, we present such researches.

#### Identifying Critical Position

A new perspective of the conspiracy numbers was recently proposed in [30]. A notion of critical position was defined as a likeliness position to result in either winning, losing or draw before settling. In other words, the critical position is a certain point among the game progress which can measure the game outcome. Then, the conspiracy numbers were used for identifying the critical positions. Secondly, new search indicators named "Positive_CN_Sum" and "Negative_CN_Sum" were introduced. Originally both values were named MaxCN and MinCN, but we give MaxCN and MinCN a new name Positive_CN_Sum and Negative_CN_Sum, respectively, for the easy to understand. Psitive_CN_Sum is calculated by $\uparrow CN$ and Negative_CN_Sum is calculated by $\downarrow CN$. Thus, the critical positions are identified by both values. We show an illustration of both values in Figure 2.5. Both values are calculated by the summation of

the conspiracy numbers using the current minimax value and the highest or the lowest minimax value.

Below we show the equations to calculate the values of Positive_CN_Sum and Negative_CN_Sum.

$$\begin{aligned} Positive\_CN\_Sum &= \sum_{i=\text{Current score}}^{\text{Highest score}} \uparrow CN(root,i) \\ Negative\_CN\_Sum &= \sum_{i=\text{Current score}}^{\text{Lowest score}} \downarrow CN(root,i) \end{aligned}$$

Where the target node is the root node, and the target score is somewhere between the current score and the highest and the lowest score. The highest score and the lowest score depend on the game. Positive_CN_Sum shows the difficulty estimation for increasing the minimax value of the root node, and Negative_CN_Sum shows the difficulty estimation for decreasing the minimax value of the root node. If the Positive_CN_Sum is very large, then the root node is facing very difficult situation to increase the minimax value. In contrast, if the Positive_CN_Sum is very small, then the root node is facing too easy situation to increase the minimax value. Moreover, Negative_CN_Sum makes a role to show the difficulty of decreasing the minimax value.

Finally, we show the way of applying the new perspective of the search indicators to identifying the critical positions. If Positive_CN_Sum and Negative_CN_Sum are abruptly changing, then the game progress is also changing. Especially, if Positive_CN_Sum and Negative_CN_Sum abruptly decrease after a certain move, then there is a critical position and each player has to reconsider own tactics. Specifically, if $|Positive\_CN\_Sum_{p_{i+1}} - Positive\_CN\_Sum_{p_i}| > |Positive\_CN\_Sum_{p_{i+1}}|$ where $p$ implies the ply and $i$ equals to the number of steps (or moves) ,then the current position faces the rapid development and the root value will increase. In other words, the possibility of losing or winning is high since the likeliness of the root value to change is high, but abruptly decreasing Positive_CN_Sum implies that the outcomes is inevitable either win or lose. So, that stage is the critical position and an in-game resignation might be recommended even though the outcome looks like unclear.

If $|Negative\_CN\_Sum_{p_{i+1}} - Negative\_CN\_Sum_{p_i}| > |Negative\_CN\_Sum_{p_{i+1}}|$ where $p$ implies that the ply and $i$ equals to the number of steps (or moves), i.e., the Negative_CN_Sum changes to the half or twice after a step, then there is a critical position. In this case, each player has to reconsider own tactics. If the Positive_CN_Sum is abruptly inclined, then the root value is stabilizing in the current score, so the current player cannot hope for a large score, i.e., the situation changes from even/better to worst (from winning to a draw or a draw to losing). If the Negative_CN_Sum is abruptly inclined, then the root value is limited downward, i.e., the current player faces a chance position in which the tactic might change from worst to even/better (from losing to a draw or from a draw to winning).

The new functions of conspiracy numbers are not only to ensure the efficient search but also to monitor the game progress as well. The focus has been on the value and progress of the Positive_CN_Sum and Negative_CN_Sum for deciding strategies of each player. Observing the stability of minimax value to increase or decrease can determine the future progress of the game and the value of changing implies the timing of reconsidering the current tactic.

### Computing Approximation of Conspiracy Numbers

Further investigation in this direction was made in [29], in which a new indicator called $CN_{variance}$ was proposed, but we use a different name $CN_{difference}$ in this thesis for easy to understand. $CN_{difference}$ is calculated by Positive_CN_Sum and Negative_CN_Sum, which would predict the future outcome of the game. $CN_{difference}$ is calculated as follows:

$$CN_{difference} = Positive\_CN\_Sum - Negative\_CN\_Sum$$

Positive_CN_Sum shows the difficulty to increase the minimax value in the node, whereas Negativ_CN_Sum shows the difficulty to decrease the minimax value in the root node. So, small Positive_CN_Sum shows the easy case for increasing the minimax value in the node. Thus, Positive_CN_Sum and Negative_CN_Sum successfully show the new insight that is different from original conspiracy numbers. However, there is a weak point such as it cannot see the combined difficulty for increasing and decreasing the minimax value in the root node. $CN_{difference}$ solved this problem and it has some meaning in the root node as described below.

- $CN_{difference}$ is positive
  This case implies that Positive_CN_Sum is larger than Negative_CN_Sum, and we have to examine many leaf nodes for increasing the minimax value. For the current player, the future progress of the current game might change to worse after the current position. And, second player has the chance for getting higher score than current score.

- $CN_{difference}$ is negative
  This case means that Negative_CN_Sum is larger than Positive_CN_Sum, and we have to examine many leaf nodes for decreasing current score, i.e., it is easier to increase the current minimax value than decreasing the score. So, there are many chances and the future progress of the game supports the current player. If the current node is a min node, then this situation gives disadvantage to the second player.

- $CN_{difference}$ is zero
  Positive_CN_Sum and Negative_CN_Sum is equal in this case. The current position has the same chance for increasing and decreasing the current minimax value. Thus, we cannot predict the future progress of the game from this position. In other words, the current player faces a turning point of the game progress, and we recommend to reconsider the current strategy. Khalid et al. [29] defined that such a position is the critical position.

- $CN_{difference}$ is infinity
  The conspiracy numbers show $\infty$ value in the terminal nodes. So, this case represents that search space reached the terminal nodes, i.e., the game is solved. Positive $\infty$ is calculated by Positive_CN_Sum and Negative $\infty$ is calculated by Negative_CN_Sum. The $\infty$ conspiracy number means that we have to examine infinity leaf nodes for changing current score, so there are no available or valid moves to play further.

Khalid et al. [29] recorded the progress of the minimax value, Positive_CN_Sum, Negative_CN_Sum and $CN_{difference}$ while playing the Othello games. They concluded

that $CN_{difference}$ can expect the future trend of the minimax value. Actually, the trend of the minimax value follows the previous trend of $CN_{difference}$ in their experiments. $CN_{difference}$ predicts the outcome of the game around endgame position, and it can represent the critical position for deciding the timing of resignation. If the minimax value shows no advantage for each player around endgame positions, but $CN_{difference}$ shows large value, then the outcome of that game is almost decided. So, the players are recommended to resign for mitigating worthless time. Otherwise, there is a chance for reconsidering the current tactic. Thus, $CN_{difference}$ can help players construct a strategy while playing the game and provides the critical positions for resignation or timing for changing the current tactic.

#### Other Related Work

Quang et al. [67] proposed a new way to calculate the approximate conspiracy numbers and apply to the minimax search. Recent researches show new potentials of the use of the conspiracy numbers. But, some issues still remain, e.g., expensive calculation cost of the conspiracy numbers during a search. They tackled this problem and proposed a way of calculating approximate conspiracy numbers. Additionally, they apply the conspiracy numbers to improving the evaluation function. they hypothesized that the evaluation function added the conspiracy numbers might make benefits such as identifying critical position. Then, they confirmed that the idea could improve performance of minimax tree search and the conspiracy number can be used for additional estimation of the positions.

## 2.4 Concluding Remarks

The advances of solving the games still continue. Some advances are given by the search indicators. The proof number is one of the most efficient search indicators so far. The search algorithms using the proof and disproof numbers successfully solved several games. Modifying the idea of the proof number brings different benefit, meaning and characteristics, e.g., WPNS, Dfpn+. Then, we confirmed that the definition of the search indicators is the basis of deciding the behavior of the search algorithms. Moreover, we tried to find the new perspective of the search indicators focusing on the definition and the characteristics. For example, the proof number in a node is defined by the number of leaf nodes which need to prove a target node. The characteristics of the proof number is able to show the size of a subtree for solving. Then, we found the new perspective of the proof number, i.e., the proof number can show the difficulty for solving a node. We assumed that every search indicators have the other utilization that is different from the original meaning. Then, we hypothesize that the original definitions of the search indicators support to lead the new perspectives. Additionally, some researches related CNS show the promising results to support our hypothesis. The original definition of the conspiracy numbers aims to build a strong computer player only. But, researchers found a new perspective of the conspiracy number, and confirmed its effectiveness. Then, they also focused on the original definition of the CNS, and successfully proposed new acceptable utilization. These researches support to reinforce our hypothesis and research outcomes. And, CNS uses the domain-dependent evaluation function for calculating the conspiracy numbers. But, the AND/OR tree searches does not use it. So, we expect that the new perspective of the search indicators in the AND/OR tree search show different contributions from existing researches.

# Chapter 3

# Deep Proof-Number Search

This chapter is an updated and abridged version of work previously published in

1. T. Ishitobi, A. Plaat, H. Iida, J. van den Herik (2015). Reducing the Seesaw Effect with Deep Proof Number Search, The 14th International Conference on Advances in Computer Games (ACG2015), pp 185–197.

Proof Number Search (PN-search) was developed by Allis et al. in 1994 [3]. It is one of the most powerful algorithms for solving games and complex endgame positions. PN-search focuses on an AND/OR tree and tries to establish the game-theoretical value in an efficient way, in a greedy least-work-first manner. Each node has a proof number (pn) and disproof number (dn). This idea was inspired by McAllister's conspiracy numbers, the number of children that needs to change their value in a node to change its value [38]. A proof number shows the scale of difficulty in proving a node, a disproof number does analogously for disproving a node. PN-search tries to expand a most-proving node, which is the most efficient one for proving (disproving) a node. PN-search is a best-first search in the sense that it follows a least-work-first order.

The success of PN-search prompted researchers to create many various PN-search, e.g., PN* [57], PDS [40] and Df-pn [41]. The history of these algorithms are described by Kishimoto et al. [33]. Using one of these algorithms, many games and puzzles were solved, e.g., Checkers [55, 53], and Tsume-shogi (the mating problem of Japanese chess) [57]. The algorithms used many well-known techniques such as transposition tables. On the one hand researchers sought to benefit from large computer power and memory [18], and on the other hand, researchers worked at some of the problems of PN-search, such as the seesaw effect (see section 3.1).

In this chapter, we propose a new proof-number algorithm called Deep Proof-Number Search (DeepPN). DeepPN tries to solve the seesaw problem with a different approach together with iterative deepening. And, we composed a new search indicator modified from the original proof number. In the experimental section, we use endgame positions of Othello and the game of Hex as benchmarks, and try to measure the performance of DeepPN.

The remainder of this study is as follows. We briefly summarize the details of the seesaw effect in section 3.1. The algorithm of DeepPN and its performance, are discussed in section 3.2. In section 3.3, we give our conclusion and suggestions for future work.

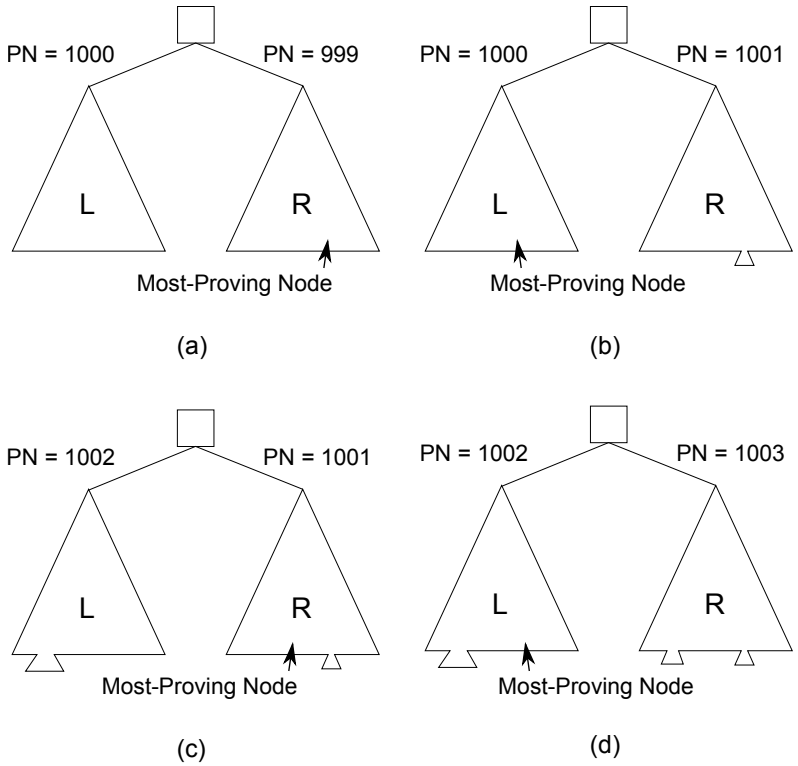



Figure 3.1: An example of seesaw effect.
(a) An example game tree, (b) Expanding most-proving node, (c) and (d) Continuing to expand most-proving nodes

## 3.1 The Seesaw Effect

The various PN-search address many issues of the algorithm, and have been used to solve many games. However, there are still some problems with PN-search that remain. Pawlewicz and Lew [46], and Kishimoto et al. [31, 32] showed one such weak point, of Df-pn. The weak point has been named the seesaw-effect by Hashimoto [16]. We provide a short sketch below.

To explain the seesaw effect, we show an example in Figure 3.1. In Figure 3.1a, the root node has two large subtrees. The size of both subtrees is almost the same. Furthermore, assume that the proof number of subtree L is larger than the proof number of subtree R. In this case, PN-search will focus on subtree R, will continue the searching, and will expand the most-proving node. When PN-search expands a most-proving node, the shape of a game-tree changes as shown in Figure 3.1b. By expanding the most-proving node, the proof number of subtree R becomes larger than the proof number of subtree L. Because of this, the position of the most-proving tree changes from subtree R to subtree L. Similarly, when the search expands the most-proving node in subtree L, the proof number of subtree L changes to a larger value than the proof number of subtree R. Thus, the search switches its focus from subtree L to subtree R. This changing continues to go back and forth and looks like a seesaw. Therefore, it is named the seesaw effect. The seesaw effect happens when the two trees are almost equal in size.

If the seesaw effect occurs, the performance of PN-search and Df-pn deteriorates significantly [41]. Df-pn tries to search efficiently by staying around a most-proving node as in Figure 3.1a. However, when the seesaw effect occurs, Df-pn should go back to the root node, and switch focus to another subtree and start to find a new most-proving node existing in subtree L. If the seesaw effect occurs frequently, the performance of Df-pn becomes close to that of PN-search, because Df-pn loses the power of its depth-first behavior.

For PN-search, the algorithm uses the proof numbers to search efficiently in a best-first manner. If the seesaw effect occurs frequently, PN-search will concentrate alternatively on one subtree. PN-search will then expand subtrees L and R equally and it cannot reach the required depth. In games which need to reach a large fixed depth for solving, this effect works strongly against efficiency.

The causes of the seesaw effect are mostly (1) the shape of the game tree and (2) the way of searching. Concerning the shape of game tree, there are two characteristics: (1a) a tendency for the number of child nodes to become equal and (1b) many nodes with equal values exist deep down in a game tree. In (1a), if the number of a child node in each node becomes almost the same, then the seesaw effect may occur easily. For (1b), this is the case in games such as Othello and Hex. In many cases, these games need to search a large fixed number of moves before settling, and it is difficult to assess upon a win, loss, or draw before a certain number of moves has been played. In the game tree of these games, the nodes can establish their value after a certain depth has been searched. Thus, when the seesaw effect occurs and the search cannot reach the required depth, it cannot determine the status of the subtrees. Instead of see-sawing between subtrees, the search should stick with one subtree and search more deeply. A game tree that has these characteristics is called a *suitable tree* by Hashimoto. Games such as Othello, Hex and Go are able to build up a suitable tree easily. For (2), the way of searching, i.e., the best-first manner, causes the seesaw effect. The most-proving node of PN-search and Df-pn is determined using proof numbers. Thus, in the Figure 3.1, Df-pn has to go back to the root node again and again, and PN-search and Df-pn cannot reach a required depth in the subtree.

One solution for the seesaw effect is the ”1 + ε trick” proposed by Pawlewicz and Lew [46]. They focused on Df-pn and changed the equation for calculating the threshold. To paraphrase their explanation, they add a margin determined by ε to the thresholds. This margin is calculated by the size of other subtrees, and it is recalculated in each seesaw. By the added margin for the thresholds, Df-pn can reach node in a specific branch more deeply than the original algorithm. Hence, the frequency of the seesaw effect is reduced. Consequently, Df-pn with the $1 + \varepsilon$ trick works better than the original Df-pn. However, we expected that this trick has at least three problems. First, the trick breaks a rule about the most-proving node. The original thresholds keep the definition of most-proving node, but $1 + \varepsilon$ just adds a margin to the thresholds. Second, if the game tree changes become too large, then also the margin becomes too large, because the margin is calculated by the size of the other subtree. On the one hand, a large margin can reduce the frequency of the seesaw effects, on the other hand, if a subtree to be searched is found not to lead to any result, then the search cannot change the subtree until reaching that margin. Third, the $1 + \varepsilon$ trick only reduces the frequency of the seesaw effect and does not completely solve the problem.

And, another solution is Dfpn+ proposed by Nagai [42]. Dfpn+ uses the cost function *cost* to tackle the seesaw effect problem. The cost function *cost* makes the weight between the connected two nodes, and it can control the ordering of visiting nodes. In other words, the cost function *cost* changes the position of the most-proving node as

like as programmer. Nagai set *cost* for focusing on the deep depth of the game tree, and tried to reduce the frequency of the seesaw effect. This approach succeeded to search efficiently in Othello endgame positions, and it implies that the behavior of the depth-first search in Othello endgame positions is effectiveness. Dfpn+ with *cost* is a good approach for the seesaw effect problem, but some problems still remain. The problems of Dfpn+ are similar as the problems of $1 + \varepsilon$ trick, but the biggest one is that the effective cost function depends on the target games. We hope to find more generic way for solving the seesaw effect problem.

## 3.2 DeepPN

In this section, we explain a new algorithm based on proof numbers named Deep Proof-Number Search (DeepPN). DeepPN is modeled after the original PN-search, and all nodes have proof numbers and dis-proof numbers. Additionally, for DeepPN, each node is also a so-called *deep value*. The deep values are determined and updated by the terminal node analogously to the proof and disproof numbers. DeepPN has been designed to: (1) combine best-first and depth-first search, and (2) to try and solve the problem of the seesaw effect. For evaluating the performance of DeepPN, we use endgame positions of Othello and Hex.

### 3.2.1 The Basic Idea of DeepPN

In the original PN-search, the most-proving node is defined as follows [3].

**Definition** For any AND/OR tree $T$, a most-proving node of $T$ is a frontier node of $T$, which by obtaining the value true reduces $T$'s proof number by 1, while by obtaining the value false reduces $T$'s disproof number by 1.

This definition implies that the most-proving node sometimes exists in a plural form in a tree, i.e., there are many fully equivalent most-proving nodes. For example, if the child nodes have the same proof or disproof number then both subtrees have each a most-proving node. The situation that the child nodes has the same proof (disproof) number in an OR (AND) node is called a tie-break situation. Now, we have the question about which most-proving node is the best for calculating the game-theoretical value. PN-search chooses the leftmost node with the smallest proof (disproof) number, also in a tie-break situation. In particular, the proof and disproof number do not take other information into account, and therefore PN-search cannot choose a more favorable most-proving node in a tie-break situation.

Determining the best most-proving node in a tie-break situation is a difficult task, because the answer depends on many aspects of the game. However, when focusing on games which build up a suitable tree, we may develop some solutions. In a suitable tree, the "best" most-proving node is indicated by its depth number. Let us look at the example (given in Figure 3.2).

This game tree is based on Othello. The game end is shown by "Game End" in Figure 3.2. All level-two nodes are most-proving nodes, because the proof numbers of child nodes under the root node are the same (i.e., 2). So, we have a tie-break situation. Now, in the next search step, PN-search will focus on the most-proving node that exists in left side as produced by the original PN-search algorithm. However, if the search focuses immediately on the most-proving node of the right side, then the search will be more efficient, because the nodes on the left side do not reach the game end and their

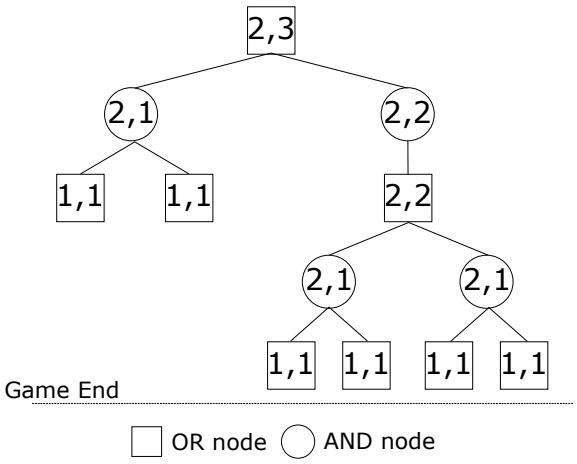



Figure 3.2: An example of suitable tree for Othello end-game position.
This game tree has uniform depth of 4, and the terminal nodes are reached at game end.

value cannot be found yet. In contrast, nodes that exist at the right side reach the game end, and if we try to expand these nodes, then the game value of each node is known. In this example, we follow the idea that a most-proving node in the deepest tree of a suitable game tree, is the best.

To test this idea, we performed a small experiment. We prepared an original PN-search and a modified PN-search. In a tie-break situation, PN-search focuses on a most-proving node that exists in the leftmost node, and the modified PN-search focuses on the *deepest* most-proving node. For checking performance, we prepared 100 Othello endgame positions. The performance of the modified PN-search is better than the results of the original PN-search (about 10% reduction). These results suggest that the *deepest* most-proving node works advantageously for finding the game-theoretical value.

In addition, the example of Figure 3.2 shows the essence of the seesaw effect. If the game end exists and has a depth of more than 4, then the search for a proof number goes back and forth between the two subtrees. Even if the game end is of depth 4, then the search that focuses on the right subtree will change its focus on the left subtree. But, when modifying PN-search, the small seesaw effect is suppressed. This phenomenon of modifying PN-search suggests a new heuristic. The search depth of nodes can be used for solving the seesaw effect in a suitable game tree. In fact, this is what $1 + \varepsilon$ trick [46] in effect tries to accomplish, to stay deep in a suitable game tree. Now, let us try to think of a new technique. For instance, consider the moves that the modified PN-search plays when finding the deepest most proving node. We noticed that these moves combined best-first with depth-first behavior. The modified PN-search works in a best-first manner, and in a tie-break situation, PN-search work depth-first for the most-proving nodes. Depending on how often tie-breaks occur, the algorithm works more frequently best-first than depth-first. The resulting improvement, when measured in number of iterations and nodes leads to a small result. Thus, we will design a new algorithm that can change the ratio of best-first manner and depth-first manner. Its description is an follows. This system is named Deep Proof-Number Search (DeepPN). Here, $n.\phi$ means proof number in OR node and disproof number in AND node. In

contrast, $n.\delta$ means proof number in AND node and disproof number in OR node.

1. The proof number and disproof number of node $n$ are now calculated as follows.

$$n.\phi = \begin{cases} n.pn & \text{(n is an OR node)} \\ n.dn & \text{(n is an AND node)} \end{cases}$$
$$n.\delta = \begin{cases} n.dn & \text{(n is an OR node)} \\ n.pn & \text{(n is an AND node)} \end{cases}$$

2. When $n$ is a terminal node

   **(a)** When $n$ is proved (disproved) and $n$ is an OR (AND) node, i.e., OR wins

   $$n.\phi = 0,\ n.\delta = \infty$$

   **(b)** When $n$ is disproved (proved) and $n$ is an AND (OR) node, i.e., OR does not win

   $$n.\phi = \infty,\ n.\delta = 0$$

   **(c)** When $n$ is unsolved, i.e., its value is unknown

   $$n.\phi = 1,\ n.\delta = 1$$

   **(d)** When $n$ is terminal node, then $n$ has deep value

   $$n.deep = \frac{1}{n.depth} \tag{3.1}$$

3. When $n$ is an internal node

   **(a)** The proof and disproof number are defined as follows

   $$n.\phi = \min_{n_c \in \text{children of n}} n_c.\delta$$
   $$n.\delta = \sum_{n_c \in \text{children of n}} n_c.\phi$$

   **(b)** The deep values, $DPN(n)$ and $n.deep$ are defined as follows.

   $$n.deep = n_c.deep \quad \text{where} \quad n_c = \underset{n_i \in \text{unsolved children}}{\arg\min} DPN(n_i) \tag{3.2}$$
   $$DPN(n) = (1 - \frac{1}{n.\delta})R + n.deep(1-R) \qquad (0.0 \le R \le 1.0) \tag{3.3}$$

The proof and disproof number are the same as in the original PN-search. The improvement is the new term, i.e., the concept of the *deep* value. The deep value in a terminal node is calculated by Equation (3.1). The deep value is designed to decrease inversely with depth. In an internal node, calculating the deep value has only a limited complexity. First, we define a function named DPN (see Equation 3.3). DPN has two features: (a) $n.\delta$ is normalized and designed to become larger according to the growth of $n.\delta$ and (b) a fixed parameter $R$ is chosen. $R$ has a value between 0.0 and 1.0. If

$R$ is 1.0 then DeepPN works the same as PN-search, and if $R$ is 0.0 then DeepPN works the same as a primitive depth-first search. Therefore, the normalized $\delta$ fulfills the role of best-first guide and the *deep* acts as a depth-first guide. This means that by changing the value of $R$, the ratio of best-first and depth-first search of DeepPN can be adjusted. Second, in an internal node, the deep value is updated by its child nodes using Equation (3.2). The deep value of node $n$ is decided by a child node $n_c$ which has smallest $DPN(n_c)$. A point to notice is that the updating value is only *deep*, not $DPN(n_c)$. Additionally, when $n_c$ is solved, then the deep value of $n_c$ is ignored in arg min.

In DeepPN, an expanding node in each iteration is chosen as follows.

$$select\_expanding\_node(n) := \underset{n_c \in \text{children of n except solved}}{\arg\min} DPN(n_c) \tag{3.4}$$

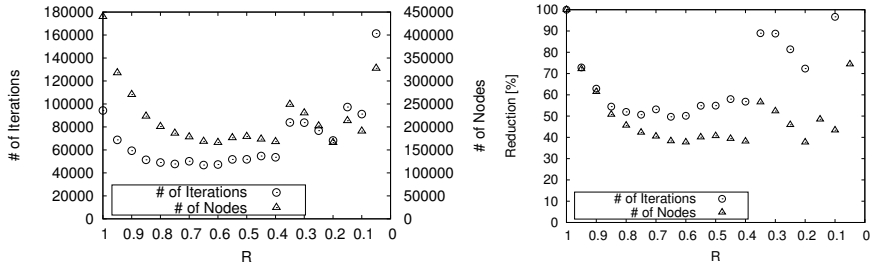
This sequence is repeated until the terminal node is reached. That terminal node is the node that is to be expanded. If $R = 1.0$, then this expanding node is the most-proving node.

The original PN-search guarantee that it can prove an AND/OR tree. In DeepPN, this guarantee depends on the characteristics of the game and amount of $R$. If we use the primitive depth-first search for solving the games which do not have the bottom in its game tree, then we cannot solve that game. DeepPN using strong depth-first manner also has the same problem. If the best-first manner affect more than the depth-first manner, then DeepPN can prove the target game tree. But, if the strength of the depth-first manner exceeded the best-first manner, then DeepPN cannot guarantee the same work as the original PN-search. However, if the game had the certain number of depth for the game end or the programmer set the threshold of the search depth, then DeepPN can prove the target game again.

### 3.2.2 Performance with Othello

For measuring the performance of DeepPN, we prepared a solver using the DeepPN algorithm and Othello endgame positions. We configured a primitive DeepPN algorithm for investigating the effect of DeepPN only, without any supportive mechanisms such as transposition tables and $\varepsilon$-thresholds. We prepared 1000 Othello endgame positions. They are constructed as follows. The positions are taken from the 8 x 8 board. We play 44 legal moves at random from the begin position. This implies that 48 squares from the 64 are covered. So, the depth of the full tree to the end is 16.

In all our experiments DeepPN is applied to these 1000 endgame positions. Our focus is the behavior of R (see Equation (3.3)). For $R = 1.0$, DeepPN works the same as PN-search and shows the same results. For $R = 0.0$, DeepPN works the same as a primitive depth-first search. When $R$ between 1.0 to 0.0, then DeepPN behaves as a mix between best-first and depth-first. We changed $R$ from 1.0 to 0.0 by increments of 0.05. We focus on the values of two concepts, viz. the number of iterations and the number of nodes. The number of iterations is given by counting the number of traces of finding the most-proving node from the root node, i.e., the number of times for expanding a most-proving node. This value indicates an approximate execution time unaffected by the specifications of a computer. The number of nodes is an indication of the total number of nodes that are expanded by the search. This value is an approximation of the size of memory needed for solving. We show the results in Figure 3.3.




(a) The variation (b) The reduction rate

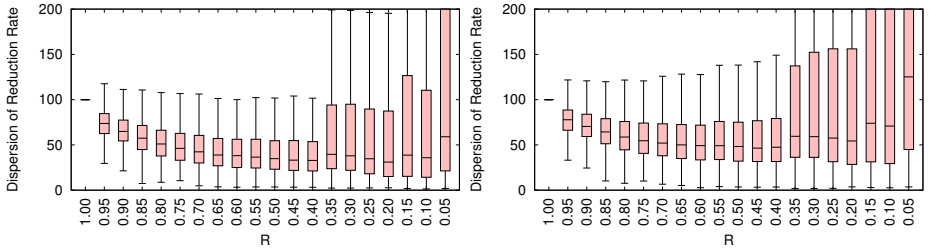
Figure 3.3: Othello: The # of iterations and # of nodes.
$R = 1.0$ is PN-search, $R = 0.0$ is depth-first search, and $1.0 > R > 0.0$ is DeepPN. Lower is better.

Figure 3.3a shows the variation of (1) the number of iterations and (2) the number of nodes. Each point is a mean value calculated from the results of 1000 Othello endgame positions. $R = 1.0$ shows the results of PN-search, and this value is the base for comparison. As $R$ goes to 0.8, the number of iterations and nodes decrease almost by half. From $R = 0.8$ to 0.6, the number of iterations stops decreasing, but the number of nodes decreases slowly. From $R = 0.6$ to 0.4, the decrease stops, and the number of iterations starts increasing again slowly. In $R = 0.35$, both numbers increase rapidly. We see that for $R$ of around 0.4, the balance between depth-first and best-first behavior appears to be optimal. We surmise that DeepPN is stuck in one subtree and cannot get away since the algorithm is too strongly depth-first. For $R = 0.35$ to 0.2, the number of iterations and nodes is decreasing. Around $R = 0.2$, the balance was broken again, and is decreasing towards 0.1. Finally, DeepPN performs worse when $R$ approaches 0.0 closely. In $R = 0.0$, almost no Othello end game position can be solved, and this value is omitted from Figure 3.3a.

In Figure 3.3a, the scale of the number of iterations and nodes are different. To ease our understanding, Figure 3.3b shows the amount of the reduction rate. This reduction rate is normalized by the result of PN-search, i.e., the reduction rate of $R = 1.0$ is 100%. Each point is calculated by the figure3.3a. In Figure 3.3b, the number of iterations decreased about 50% in $R = 0.6$ and the number of nodes decreased about 38% in $R = 0.6$. Thus, DeepPN reduced the number of iterations ($\approx$ time) to half and the number of nodes ($\approx$ space) to two-fifth. In $R = 0.05$, the number of iterations increased to over 100%, which is not shown.

Finally, we show two graphs about the dispersion of our experimental results in Figure 3.4.

The box-plots are the dispersion of the reduction rates in each $R$. These plots are calculated by the 1000 reduction rates compared by the results of $R = 1.0$, so the all reduction rates in $R = 1.0$ are 100%. From the lower, the graphs show the minimum reduction rate, the first quartile, the median reduction rate, the third quartile, and maximum reduction rate. For easy to see, the outliers are not printed. The median reduction rate in Figure 3.4 is little different from Figure 3.3. Because, the Figure 3.3b is composed by the median value using 1000 experimental results. But, these Figures 3.4 are composed by the median value using 1000 reduction rates. Then, we tried to see the Figure 3.4a about the number of iterations, then we could find that the reduction rates




(a) # of iterations (b) # of nodes

Figure 3.4: Othello: The dispersion of reduction rates for # of iterations and # of nodes

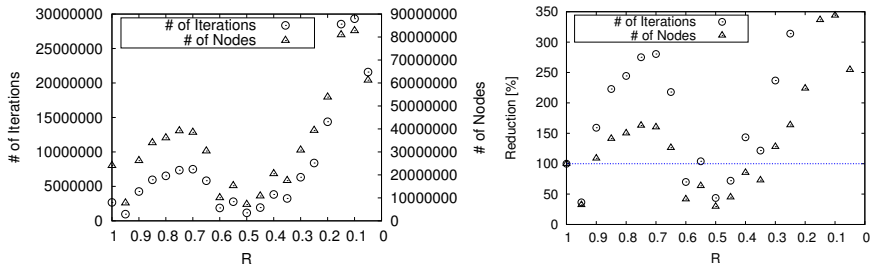
change to better when $R$ goes to 0.40 from 1.00. And, the dispersion looks narrow between $R = 1.00$ and 0.40. The maximum reduction rate exists over 100% in any case, this means that DeepPN has the potential to show the worse results than original PN-search in any case. After $R = 0.40$, the reduction rates spread widely. This result is related the Figure 3.3. DeepPN can solve most Othello endgame positions efficiently between $R = 0.40$ and 0.20, but there are some very worse results, and the dispersions change to wide. After $R = 0.20$, the dispersion change to wide more, and the third quartile breaks 100%. If $R$ close to 0.00 then reduction rates spread widely, and we cannot say that DeepPN works well. In Figure 3.4b about the number of nodes, the reduction rates change to worse faster than the results of the number of iterations. The third quartile break 100% after $R = 0.40$, and dispersions change to wide more than the result of the number of iterations. Additionally, the maximum reduction rates start to increase after $R = 0.6$. If DeepPN cannot work well in an Othello endgame position, then the number of nodes change to large more than the number of iterations. So, this means that the effectiveness of DeepPN start to retrogress after $R = 0.6$.

The question remains when DeepPN works most efficiently in the Othello endgame position for 16-ply. The answer depends on the group of Othello endgame positions. However, if we have to choose the best $R$, then a value of around 0.60 is a good compromise for most cases.

### 3.2.3 Performance with Hex

For measuring the performance of DeepPN, we also prepared a solver for Hex. As for the experiments of Othello, we created a primitive DeepPN algorithm for checking the effect of DeepPN only. The Hex program is a simple program that does not have any other mechanisms such as an evaluation function. Our Hex program uses a 4 x 4 board (called Hex(4)), and tries to solve that board using DeepPN. Our focus is on the behavior of R (see Equation 3.3). Concerning the characteristics of $R$, please see section 3.2.1 or 3.2.2. We changed $R$ from 0.0 to 1.0 by 0.5, and tried to solve Hex(4) 100 times in each $R$. The legal moves of Hex are sorted randomly in every configuration, viz. there is the possibility that each result is different. The results in each $R$ are calculated by the median of the 100 experiments. Next we focused on two concepts: (1) number of iterations and (2) number of nodes. About the characteristics of both values, please see the section 3.2.2. The experimental data are given in Figure 3.5.

Figure 3.5a shows the changes in the number of iterations and nodes. We can see that the results of DeepPN decrease (improve) in some positions compared by PN-




(a) The variation (b) The reduction rate

Figure 3.5: Hex: # of iterations and # of nodes for Hex(4).
$R = 1.0$ is PN-search, $R = 0.0$ is depth-first search, and $1.0 > R > 0.0$ is DeepPN. Lower is better.

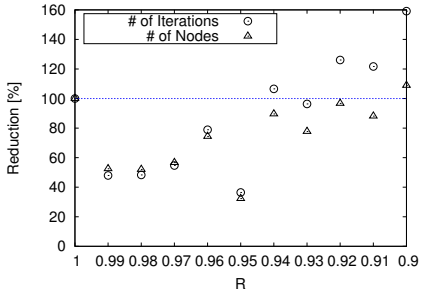



Figure 3.6: Hex: The detail of Figure 3.5b.
This figure is zoomed $1.0 \geq R \geq 0.9$. The lower is better.

search. This is not the case for $R = 0.0$, because we cannot solve Hex(4) for this R when we limit ourselves to 500 million nodes. For ease of understanding, we prepare another graph in Figure 3.5b. There we show the reduction rates normalized by the result of PN-search, i.e., the result of PN-search has 100% reduction rate.

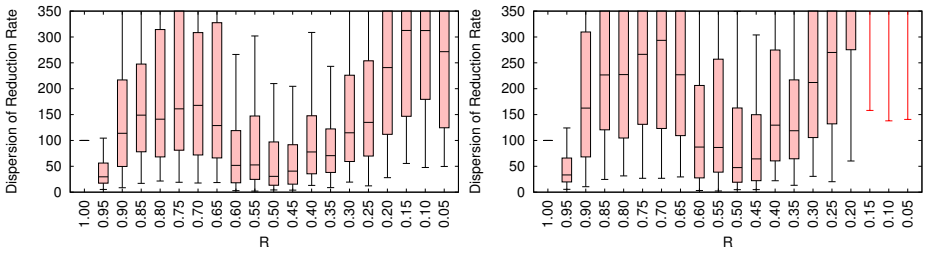
Figure 3.5b shows that the number of iterations and nodes is reduced by a 36% and 32% reduction rate at $R = 0.95$ and $R = 0.5$. The result has two downward curves: from $R = 1.0$ to 0.7 and from $R = 0.7$ to $R = 0.0$. The first curve starts from $R = 1.0$ and decreases toward 0.95. After $R = 0.95$, the results start to increase and grow to over 100% after 0.85. The second curve starts from $R = 0.7$ and the results starts to decrease again. Finally, the results are increasing again toward $R = 0.0$, like Othello.

For understanding the details of how DeepPN works around $R = 0.95$, we tried to change $R$ by 0.01 between from 1.0 to 0.9. The results are shown in Figure 3.6.

By looking at the results, we can see that DeepPN works almost twice as good as PN-search from $R = 0.99$ to 0.95 (except $R = 0.96$). Form $R = 0.95$ to 0.90, we have almost the same result as original PN-search.

Finally, we show two graphs about the dispersion of our experimental results in Figure 3.7.

In Figure 3.7, we show the dispersions of the number of iterations and nodes in each $R$. There are the box-plots about the dispersion of reduction rate in each $R$. These plots are calculated by the 100 reduction rates compared by the results of $R = 1.0$, so




(a) # of iterations (b) # of nodes

Figure 3.7: Hex: The dispersion of reduction rates for # of iterations and # of nodes

all reduction rates in $R = 1.0$ are 100%. For easy to see, the outliers are not printed.

The dispersions in each $R$ are wider than Othello experiments. In the ranges which reduction rates show the good results, the dispersions are narrow. In other area such as DeepPN show bad results, the dispersions are very wide. And, both ranges which has good results show little different results, viz. $R = 0.95$ show better result than around $R = 0.45$.

Therefore, in Hex(4), the optimum value of $R$ is around $R = 0.95$. We can see that depth-first does not work so well for Hex(4) as it does for Othello, although there is an improvement over pure best-first.

### 3.2.4 Discussion

DeepPN works efficiently in 16-ply Othello endgame positions, and in Hex(4). It can reduce the number of iterations and nodes almost by half compared to PN-search. It must be noted that the optimum balance of $R$ is different in each game and for each size of game tree. We can see that for both games a certain amount of depth-first behavior is beneficial, but the changes are not the same. The precise relation is a topic of future work.

In our experiments, we encountered positions that showed increasing (worse) results. We suspect that a reason for this problem may be (1) the length of the shortest correct path and (2) the holding problem.

Concerning (1), in Othello, the shortest correct path is almost the same for each position, because Othello has a fixed number of depth to the end. However, in Hex(4), the shortest winning path may exist before a depth of 16. If we happen to find a balance between depth and best-first, then DeepPN will change the subtree it focuses on time. For example, when $R = 0.95$, then DeepPN quickly finds the shortest path. But after $R = 0.95$, DeepPN misses that path and arrives in regions that are more deeply in the trees. Finding a good value of $R$ in Hex is more difficult than in Othello.

Concerning (2), the depth-first search can remain stuck in one subtree (holding on to the subtree). If this holding subtree cannot find the game-theoretical value, then the number of iterations and nodes become meaningless. When DeepPN employed a strong depth-first manner, then we found many increasing results in Othello endgame positions. Also, in Hex(4), DeepPN cannot work efficiently around $R = 0.0$. Finding an optimal $R$ is a topic of future work.

For this two hypothesis, we did additional experiments in the following subsections.

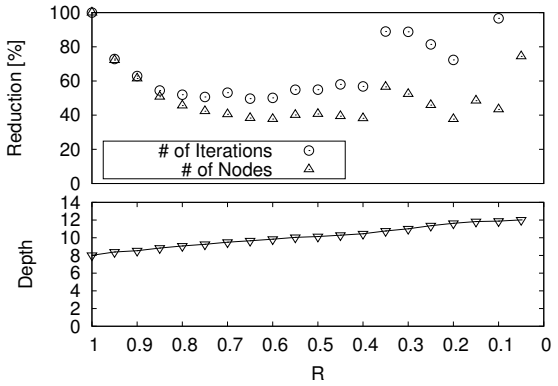



Figure 3.8: Othello: Relationship between search depth and reduction rates.

### 3.2.5 Focusing on the Depth and $R$

Concerning (1), we compared between the reduction rates and the average search depth of the most-proving nodes as follows Figure 3.8.

Figure 3.8 shows the relationship between the search depth and the reduction rates. The reduction rates is the same as the Figure 3.3b. The search depth is calculated as follows. First, we recorded the search depth in each most-proving node while DeepPN is solving an Othello endgame position. If DeepPN expand a most-proving node, then we count up the existence of a most-proving node in the certain depth. Second, we calculate the average depth from recorded data as follows.

$$\text{Average depth} = \frac{\sum_{i=1}^{\text{max depth}} (i * \text{number of most-proving node})}{\text{number of iterations}}$$

The iterations means that the number of times for expanding a most-proving node. After solving all Othello endgame positions, we calculated the median search depth. So, the search depth in Figure 3.8 show that the the ratio which DeepPN focuses on the depths in each $R$.

In $R = 1.0$, DeepPN focuses on the depth 8 in the game trees. The maximum depth in 16-ply Othello endgame position is 16, so we see that original PN-search focuses on the broad depth evenly. This means that original PN-search focuses on each depth in order from shallow depth to the deep depth. So, we understood that original PN-search faces the seesaw effect, and it cannot reach the deep depth easily. After changing $R$ and mixing the depth first behavior to original PN-search, then the search depth start to increase and the reduction rate start to decrease. The search depth continue to increase between $R = 1.0$ to 0.05, so DeepPN focuses on the deeper depth according to mix the depth-first manner strongly. In $R = 0.6$, DeepPN works well and it focuses on the around 10 depth. If the focusing on the game does not have the fixed number of depth for game end, then this search depth can show that the terminal depth of the shortest correct path. However, 16-ply Othello endgame positions have fixed number of depth at 16, so this 10 depth cannot show the actual search depth, and it can show the strength of the manner of the depth-first search. After $R = 0.6$, the search depth continue to increase, but DeepPN cannot work well in proportion to this increasing. Thus, Figure 3.8 shows that if we want to solve 16-ply Othello endgame position using DeepPN, then it is good to focus on the deep depth using $R = 0.6$.

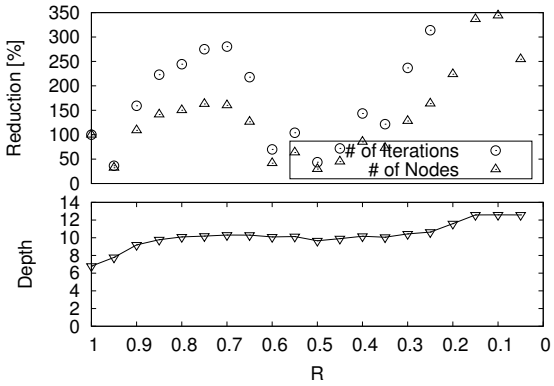



Figure 3.9: Hex: Relationship between search depth and reduction rates.

And also, we showed the search depth in Hex experiments as Figure 3.9.

In Hex(4), the search depth also increase between $R = 1.0$ and $R = 0.05$. Its start from 6 depth and goes to 12 depth. When DeepPN works well then it focuses around 8 depth and 10 depth. But, Hex(4) has the fixed number of depth at 16 for game end, so this values are not actual the search depth in the game tree, and these show the strength of the manner of the depth-first search. We tried to find the shortest correct paths of Hex(4), then we see that these exist after 10 moves from starting position. So if DeepPN can focus around 10 depth then it can work well, and this results are shown in $R = 0.95$. And if DeepPN overlook the shortest correct paths and it cannot reach the bottom of the game tree, then the number of iterations and nodes show the large value than original PN-search such as around $R = 0.80$. Moreover, if DeepPN overlook the shortest correct paths and it focuses on the deep node as priority even DeepPN reach the bottom of the game tree, then DeepPN hold on to a subtree strongly and it cannot show the good results like after $R = 0.50$.

Therefore, the search depths are increased according to the decrease in $R$, and the reduction rates show the good cases while this increases. So, we conclude that the best $R$ depends on the characteristics of the game such as the existence of the shortest correct path and the bottom of the game tree. If DeepPN focused around the depth of the shortest correct paths or the bottom of the game tree, then DeepPN can work better than the original PN-search. In contrast, if DeepPN focused on the depth unrelated the shortest correct paths or DeepPN mixes the manner of depth-first search strongly so much, then DeepPN cannot show the good works. The number of depth about the shortest correct path and the bottom of the game tree are the good hint for finding the best $R$.

### 3.2.6 Seesaw Effect and $R$

Concerning (2), we try to check the frequency of the seesaw effect. For this purpose, we record the number of best move changes as the root node. We consider that this value relate to the frequency of the seesaw effect in the root node, i.e., this value shows the estimate frequency of the seesaw effect. The experimental results are shown as follows Figure 3.10.

The above graph is the same as Figure 3.3b. The below graph shows the estimate frequency of the seesaw effect in the root node. Each plot is calculated as follows. We

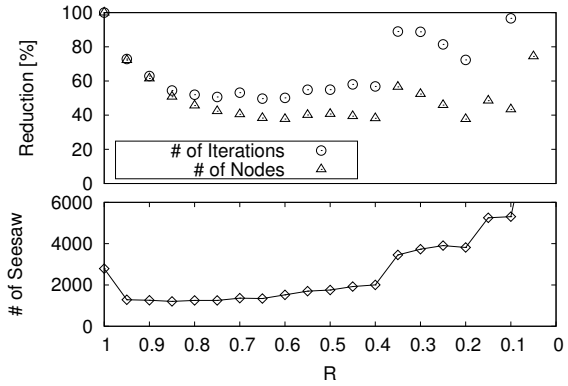



Figure 3.10: Othello: The relationship between estimate frequency of seesaw effect in root node and reduction rates.

focused on the root node while solving an Othello endgame position, and try to record the history of visiting child nodes. In the root node, DeepPN try to find a child node which has the smallest proof number. We recorded the chosen child nodes in each iteration, and if DeepPN changed the focusing on the child node then we count up it. In other words, we counted up the times of seesaw, so this value can show the estimate frequency of the seesaw effect in the root node. The below graph is composed by the median values of the estimate frequency of the seesaw effect calculated by results using 1000 Othello endgame positions.

In $R = 1.0$, the original PN-search changed focusing subtree at about 3000 times. After $R = 1.0$, the estimate frequency of the seesaw effect decreased rapidly to around 1200. Between $R = 0.95$ and 0.75, the estimate frequency of the seesaw effect stop to change, and when $R$ goes to 0.40 from 0.75, then the values start to increase slowly. In $R = 0.35$, the estimate frequency of the seesaw effect increase rapidly linked worth reduction rates. Between $R = 0.35$ and 0.20, the changes of the estimate frequency of the seesaw effect is stopped, but after $R = 0.20$, the plots show large values. If we compared both graphs, then we could see that the values are changed altogether. If the reduction rates decrease, then the estimate frequency of the seesaw effect also decreases, and if the reduction rates increase, then the estimate frequency of the seesaw effect also increases. So, if the estimate frequency of the seesaw effect is reduced by DeepPN then the searching efficiency is also improved in Othello experiments.

Next, we show the number of best move changes (the estimate frequency of the seesaw effect) in the root node in Hex(4) experiment as follows Figure 3.11.

In $R = 1.0$, the estimate frequency of the seesaw effect is about 3500, but after this, the estimate frequency of the seesaw effect decrease rapidly. Between $R = 0.95$ and $R = 0.05$, the estimate frequency of the seesaw effect decrease gently according to $R$. So, there are almost no seesaw effects between $R = 0.95$ and 0.05, but the reduction rates are changed. This decreasing the estimate frequency of the seesaw effect is caused by the depth-first manner. If DeepPN hold to a subtree as the strong depth-first manner, then the seesaw effect will be reduced. In $R = 0.05$, the estimate frequency of the seesaw effect under 10, but the reduction rates show really worth result. In $R = 1.00$, the estimate frequency of the seesaw effect is about 3500, but it show the better result than $R = 0.05$. In $R = 0.95$, the estimate frequency of the seesaw effect is about 300, and it show the better result than $R = 0.05$ and also $R = 1.00$. So, we understood that

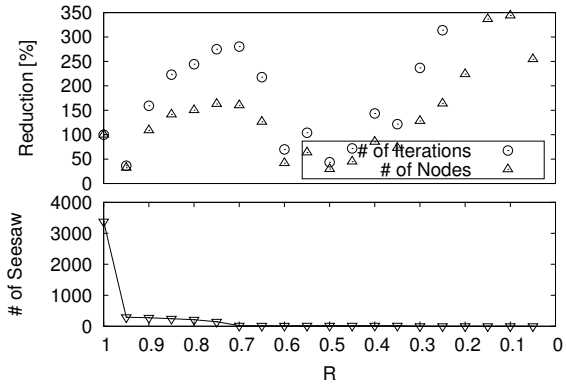



Figure 3.11: Hex: The relationship between estimate frequency of seesaw effect in root node and reduction rates.

DeepPN can work well if the seesaw effect is reduced then, but reducing the seesaw effect worsen the efficiency of the searching in some cases. Because, the seesaw effect is caused by the best-first manner and the reducing it caused by the depth-first manner. The strong depth-first manner ignore an advantage of the best-first manner, and it work reach the primitive depth-first search. So, we have to reserve minimum frequency of the seesaw effect for efficient searching, because it is the advantageous of the best-first manner.

In contrast, the estimate frequency of the seesaw effect increases according to the decrease in $R$ of the Othello experiments. At first, the depth-first manner using $R = 0.95$ succeed to decrease the estimate frequency of the seesaw effect. But, after $R = 0.95$, DeepPN could not reduce the seesaw effect and the estimate frequency of the seesaw effect start to increase. In generally, the estimate frequency of the seesaw effects are reduced according to mix the depth-first manner. Because, the primitive depth-first search can show the minimum frequency of the seesaw effect. However, the Othello experiments show different perspectives from this theory, and about this, we thought that the unnecessary nodes are related. The strong depth-first manner expands the leaf nodes which are unnecessary for original PN-search. The $R$ closed 0.0 focused on the deep value at first, and after expanding many unnecessary nodes, it starts to focus on the proof numbers. Then, each subtree has become larger, and the situation which can cause the seesaw effect is ready. So, the strong depth-first manner with the weak best-first manner causes the seesaw effect like the best-first manner only. This phenomenon depends on the characteristics of the game, and we cannot see it in Hex experiments. We suspected that this reason is related to the number of the winning processes, i.e., the number of the unnecessary nodes in the whole game tree. We considered that the many nodes which cannot relate to prove the game tree exists in the whole game tree in Othello.

Thus, the holding on to a subtree shows the good work and also bad work. The estimate frequency of the seesaw effect can be reduced if we use the strong depth-first manner and holding on to a subtree then. However, in the certain games such as Othello, the depth-first manner cannot reduce the seesaw effect, and it causes additional seesaw effect in some cases. And, even if we could reduce the frequency of the seesaw effect, then DeepPN does not guarantee the good works. This is related the concerning (2), and it can show the holding on to a subtree causes bad results. Anyway, if DeepPN

mix the best-first manner and depth-first manner rightly and it reserve the certain number of the seesaw effect then DeepPN can show the good works. And we can confirm it around $R$ close to 1.0 then.

## 3.3 Conclusion and Future works

In this work, we proposed a new search algorithm based on proof numbers, named DeepPN. DeepPN has three search indicators (pn, dn, *deep*) and a single parameter, $R$, that allows a choice between depth-first and best-first behavior. DeepPN employs two types of values, viz. proof numbers and deep values which register the depth of nodes. For measuring the performance of DeepPN, we tested DeepPN on solving Othello endgame positions and on the game of Hex. We achieved two indicative results in Othello and Hex. The algorithm owes its success to Equation (3.3) in which best-first and depth-first search are applied in a "balanced" way. The results show that DeepPN works better than PN-search in the games which build up a *suitable* tree.

We have three main topics for future work. First, we have to investigate how to find a good balance for $R$. In our experiments, the best results are produced by different values of $R$. Second, DeepPN is too primitive to solve complex problems. Df-pn+ and the use of a transposition table give a good hint for this problem. The idea of deep value may also be applied to other ways of searching. Third, we need to find a reason about the deterioration of search results in Othello endgame positions and in the game of Hex. By investigating this problem, we expect that DeepPN can become a quite useful algorithm for explaining the intricacies.

Previous work on depth-first and best-first minimax algorithms used null-windows around the minimax value to guide the search in a best-first manner [49, 48]. There, a relation between the SSS* and the Alpha-beta algorithm was found. In this work, we focus on best-first behavior guided by the *size* of the minimax tree, as is the essence of proof-number search. It will be interesting to see if both kinds of best-first behavior can be combined in future work in a new kind of conspiracy number search.

# Chapter 4

# Aesthetics of Mating Problems and Search Indicators

This chapter is an updated and abridged version of works previously published in

1. T. Ishitobi, H. Iida (2012). Analysis of Tsume-Shogi Problems using Proof-Number, *The 17th Game Programming Workshop*, pages 163–166, Hakone, Japan, in Japanese

2. T.Ishitobi, A.Cincotti, H.Iida (2013), Shape-Keeping Technique and Its Application to Checkmate Problem Composition, *Artificial Intelligence and Game Aesthetics: the 2013 AIIDE Workshop (WS-13-19)*, pages 7–10.

3. T.Ishitobi, A.Cincotti, H.Iida (2013). Shape-Keeping Heuristics, *Computer Games and Intelligence Workshop*, Yokohama.

Shogi is an ancient Japanese game just like Chess, but it is more complex [20] [25]. The checkmate problems in Shogi (Tsume-Shogi) are the mating problems which can be seen as a sub-domain of Shogi. There are a few noticeable differences and similarities between Shogi and Western Chess. In Shogi, a player can reuse (drop) pieces which have been captured from the opponent. Tsume-shogi commences from devised positions, similar to checkmate problems in Western Chess. The objective of the mating problem is to checkmate the enemy king. Differently from a checkmate problem in Western Chess, in Shogi the player must check the king continuously. For more detailed descriptions about Shogi and tsume-shogi, please refer to [20, 57]. Tsume-Shogi has a long history and rich tradition. Consequently, the aesthetics of the tsume-shogi have long been the subject of study, and excellent tsume-shogi have been collected to publish in a book style such as Tsumuya-Tsumazaruya [26] and Shogi-Zuko [27]. Tsume-shogi contests have sometimes been held, and the best problems have been selected and awarded by the judge committee which consists of Shogi master players and popular voting.

What are the aesthetic criteria of tsume-shogi? An investigation has been made by Koyama [35]. It presents the results of analyzing the database of 12697 tsume-shogi with solutions and analyzes the problems from the viewpoint of static aspects including the number of pieces on the board, the location of King at the initial position, and the freedom of King at the checkmate position, and dynamic aspects including the frequency of moves, sacrifice, the choice of moves, evaluations of scoring functions.

However, it is a difficult task to assign an appropriate score of aesthetics of tsume-shogi since there are many factors to be evaluated. In this chapter, we study the values that may be related to the aesthetic of tsume-shogi, in particular with a focus on the search indicators.

## 4.1 Aesthetic of Mating Problems

In a tsume-shogi competition, mating problems composed are ranked. The quality of the aesthetics of each tsume-shogi is evaluated and ranked by the judge committee or public voting based on their own sense. High quality tsume-shogi would collect many votes by public, and receive a prize in the top ranking. Satisfying tsume-shogi competitions means that tsume-shogi have the aesthetics which are evaluated by public in common. The aesthetics of tsume-shogi may be composed by various criteria and considerations. Koyama [35] observed that the aesthetics of tsume-shogi depend on the arranging of the pieces on the board, the space around King (or the freedom of King at the checkmate position), and other factors. However, the assessment cannot be represented as numerical values, and therefore it is difficult to quantify these criteria. To tackle this problem, we focus on the search indicators for evaluating the aesthetics of tsume-shogi. The search indicators are used for guiding the search direction during search process. Then, we hypothesize that the search indicators relate the assessment of tsume-shogi because of their new perspective. For this hypothesis, we analyze the search indicators while solving tsume-shogi. We then compare the analysis data and ranking of the competitions for finding the elements related to the aesthetics of tsume-shogi.

### 4.1.1 Mating Problems and Assessment Factors

When we focus on the search indicators to evaluate the aesthetics of tsume-shogi, then we raise a question: which search indicators are suitable? Iida [19] mentioned about the strength of computer shogi in 2006, saying that the performance of computer shogi would become similar to masters as a computer becomes very strong. Inspired by this observation, we focus on Proof-Number Search (PN-search) [3]. PN-search can solve tsume-shogi very efficiently like human experts which can solve tsume-shogi very well. So, we assume that this search algorithm as tsume-shogi solver may correspond to a master-level computer shogi which would show a human-like playing performance. For this reason, we focus on two search indicators used in PN-search; the proof number and the disproof number.

The large proof number means that PN-search needs more efforts for proving a node in question, or it is just hopeless for proving. So, the proof numbers show the degree of difficulty for proving a node. However, a given tsume-shogi must be solved, and the proof number in the root node must go to zero eventually even if the proof numbers while searching are too large. So, we assume that proof numbers in the root node while searching show the degree of difficulty for solving the tsume-shogi. Then, the proof numbers might relate to the difficulty of the Henka (possibilities at positions where the defender is to move) in tsume-shogi. Because, we have to examine all Henka moves in each step for proving the defender's loss. This is the same meaning as the proof number, so the size of the proof number shows the level of difficulty for solving a tsume-shogi. In contrast, the disproof number might relate to the difficulty of Magire (possibilities at positions where the attacker is to move) in a tsume-shogi. In some

cases, we try to disprove several positions for finding a solution, like the process of elimination. Then, we try to examine Magire moves, and this idea is like the function of the disproof numbers. So, the size of the disproof number shows the level of difficulty for eliminating Magire moves. Thus, proof and disproof number relate to the degree of difficulty of tsume-shogi. We hypothesize that proof and disproof numbers can be indicators which relate to the aesthetics of tsume-shogi.

Before confirming our hypothesis, we consider the relationship between the elements which may relate to the aesthetics of tsume-shogi. Iqbal [21] proposed a formula to evaluate the aesthetics of mating problems in chess as shown in Equation (4.1).

$$P = \sum_{k=1}^{m} C_k \;\Rightarrow\; A = \sum_{i=1}^{n} Pi \tag{4.1}$$

Where $P$ is principle, $C$ is component, and $A$ is overall value. The principles $P$ are the elements which correlate to aesthetics such as the summation of the piece's power, the arranging of pieces, and others. Components $C$ are the pieces which compose a principle $P$. If a principle shows the summation of the piece's power, then each $C$ shows a power of each piece. The overall value $A$ is composed by the summation of each principle, and this value shows the total aesthetics of a target. Iqbal [22, 23, 25] introduced the principles which relate to the aesthetics of chess problems (such as the spread out the pieces), and tried to combine these for evaluating the chess problems. Recently, Iqbal [24] tried to assign the principles automatically using the aesthetics of other domains.

The principles for evaluating chess problems show the good outcomes, so we referred Iqbal's idea. Then, we hypothesize that the search indicators take one principle in this case. An important issue is that the ratio of the search indicators to an overall value $A$. To tackle this problem, we conduct some experiments. First, we collect the data about the search indicators while solving each tsume-shogi. Second, we compare the ranking of the competitions with the collected data, and try to analyze the relationship between the difficulty of tsume-shogi and the evaluation of tsume-shogi. Finally, we conclude on the usefulness of our hypothesis; 1) Can the search indicators evaluate the aesthetics of tsume-shogi? and 2) How many rates of the search indicators as a principle $P$ to an overall value $A$?

### 4.1.2 Experiments using Kanju-Award Problems

Tables 4.1 and 4.2 show three search indicators of 7-step contest problems, 15-step Kanju-Award winning problems and general problems. The ranking of the 7-step contest problems were decided by the vote in public where most participants were amateur shogi players. The Kanju-Award is the most prestigious award in the domain of tsume-shogi, decided by experts. Among the collections of tsume-shogi, only 15-step problems are selected in this study. General problems are tsume-shogi composed for the look-ahead exercises of shogi players, and are included for reference but not for study in this experiment. We collected forty 7-step problems and twenty 15-step problems from the tsume-shogi books and calculated the average of these data. It is noteworthy that contest and Kanju-Award winning problems were evaluated by experts, whereas general problems are evaluated by amateur shogi players.

The maximum proof number (MAX PN) and maximum disproof number (MAX DN) are the values recorded during the search until the problem is solved. The proof and disproof number is related to the difficulty of Magire and Henka, respectively.

Table 4.1: 7-step compeition problems and general problems

| Ranking | MAX PN | MAX DN | Iterations |
|---|---|---|---|
| 1st | 116 | 3258 | 235574 |
| 2nd | 32 | 3985 | 58276 |
| General | 19 | 1720 | 62136 |

Table 4.2: 15-step Kanju Award-winning problems and general problems

| Year | MAX PN | MAX DN | Iterations |
|---|---|---|---|
| 1998 | 47 | 42427 | 1066572 |
| 2001 | 49 | 36417 | 1023204 |
| 2005 | 67 | 117561 | 4462292 |
| 2011 | 25 | 62046 | 741920 |
| General | 32 | 5528 | 173553 |

Therefore the maximum proof and disproof number shows the peak of difficulty about the Henka and the Magire. The number of iterations shows how many times the root node is renewed by PN-search, i.e., the number of times for examining the most-proving nodes. We compare these numbers, and the results are shown below. Among general problems, good tsume-shogi have larger proof and disproof numbers. The maximum proof numbers of 7-step contest problems appear in the same order with the contest ranking order. The 15-step Kanju-Award winning problems have larger proof and disproof numbers compared to general problems too. Thus, it is observed that proof and disproof numbers are important search indicators to evaluate the aesthetics of tsume-shogi.

### 4.1.3 Experiments using Tsume-shogi Competition

Search indicators play an important role for the assessment of the aesthetics of tsume-shogi. For more detailed analysis of relationship between the search indicators and the aesthetics of tsume-shogi, we analyzed tsume-shogi competition organized by the Tsume-shogi Paradise magazine [62] [60]. We obtained the maximum proof and disproof numbers and tried to compare these values with the ranking of the competitions. The details of the competitions are as follows. First, the magazine produced some tsume-shogi entries about 40 to 60. These tsume-shogi entries were sent by the public, and chosen by experts. Second, the magazine started to collect the votes (3, 2 and 1 point) and the answers of each tsume-shogi from the public readers. Finally, the magazines presented the ranking of these tsume-shogi sorted by the evaluation of the readers. This ranking is decided by the score, calculated by the votes and answers (wrong answer scored 3 point).

As for the number of iterations, it did not show us any good relation with the aesthetics of tsume-shogi. It is found that this value highly depends on the search algorithms, so we conclude that it is unsuitable for analyzing tsume-shogi. Then, we provide the number of visited nodes as a new value. This value means that the overall number of nodes expanded in working memory while searching. The number of nodes shows the size of a game tree, so it is expected that this value shows the synthetic difficultly of the puzzles. We show the correlation between the search indicators and the ranking in Figures 4.1, 4.2 and 4.3. The broken lines represent approximate lines. For

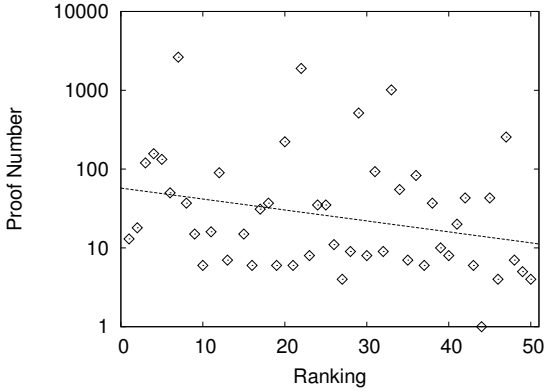



Figure 4.1: Ranking of Competition in 2014 with Proof Number

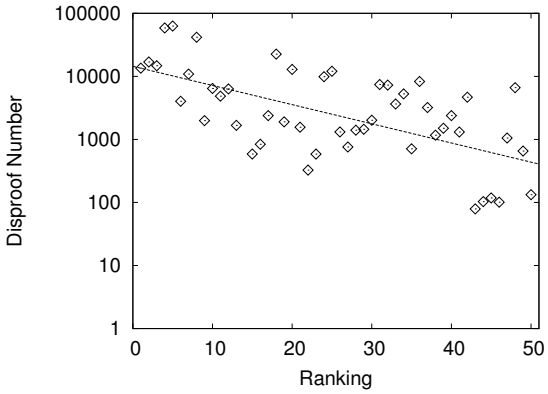



Figure 4.2: Ranking of Competition in 2014 with Disproof Number

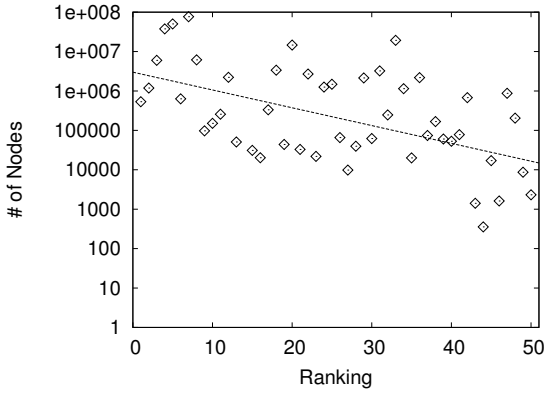



Figure 4.3: Ranking of Competition in 2014 with # of Nodes

Table 4.3: A correlation coefficient R between ranking and search indicators in each year

| Year | Max PN | Max DN | # of Nodes |
|---|---|---|---|
| 1992 (5-step) | -0.1238 | -0.1412 | -0.0946 |
| 2013 (7-step) | 0.0633 | -0.1703 | -0.1004 |
| 2007 (9-step) | -0.0445 | -0.3452 | -0.3056 |
| 2012 (9-step) | -0.1665 | -0.3586 | -0.2854 |
| 2014 (9-step) | -0.1570 | -0.4992 | -0.3589 |
| 2011 (11-step) | -0.0490 | -0.1270 | -0.2160 |

easy to see, a vertical line is set as logarithmic axis. From each figure, we can see that each plot is spread in the graph, but approximate lines show the decreasing tendency according to the ordering of the ranking. For the analysis, we collected the data from the year 2007, 2012 and 2014 competition for 9-step tsume-shogi, 1992 competition for 5-step tsume-shogi, 2013 competition for 7-step tsume-shogi, and 2011 competition for 11-step tsume-shogi. 1992 competition was organized by "Shogi Sekai", but the system of the competition is the same as others. Some competitions include tsume-shogi in which there are two Kings (called "Ryo-gyoku") but only a few cases. In this case, we ignored these "Ryo-gyoku" problems for our analysis. The correlation coefficient $R$ of each competition and each search indicator are shown in Table 4.3.

Table 4.3 shows the correlation coefficient values $R$ between the ranking and the search indicators in each competition. From this table we can see that each $R$ has small minus correlation. This minus correlation implies that the evaluated tsume-shogi have the large values, and these values decrease according to the ordering of the ranking. Table 4.3 shows that the maximum proof numbers have the smallest correlation $R$. The correlation between the maximum disproof numbers and the number of visiting nodes also shows smaller correlations, but larger than the correlation between the maximum proof number and the number of visiting nodes. Among them, the correlations of the maximum disproof numbers have the largest value in each year except the 2011 competition. From this experiment, it is not expected that there exists a good relationship between the ranking and the search indicators, because of the small correlation coefficient value $R$. We suppose that this problem happens due to the ordering of the ranking. The ordering of the ranking is decided by the score of each tsume-shogi, then it lacks the information about the amount of the score. For example, we can see that the first winning tsume-shogi obtained a larger score than the second one, but we cannot see the difference between both tsume-shogi problems. To tackle this problem, we calculate the Spearman's rank correlation coefficient. The rank correlation ignores the amount of the differences between the values of the search indicators, and it focuses only on the ordering of these values. We made the ranking of the search indicators in the decreasing order, and compared the results of the competitions. The results are shown in Figure 4.4. Moreover, the p-value using Student's t-test for each $\rho$ is shown in Table 4.5.

The 5-step tsume-shogi competition held in 1992 shows the small rank correlation in each search indicator. The maximum disproof number shows the best correlation, and the next one is the number of visiting nodes and the maximum proof number. In this experiments, we set the significance level as 0.05, so the values without "*" in the Table 4.5 are removed. Then, the rank correlation of the maximum proof number is removed in the 1992 competition. For the 7-step tsume-shogi competition held

Table 4.4: A Spearman's rank correlation coefficient ρ between ranking of competitions and search indicators using decreasing order in each year

| Year | Max PN | Max DN | # of Nodes |
|---|---|---|---|
| 1992 (5-step) | 0.2922 | 0.3753 | 0.3638 |
| 2013 (7-step) | 0.4593 | 0.5190 | 0.4855 |
| 2007 (9-step) | 0.1597 | 0.4692 | 0.3759 |
| 2012 (9-step) | 0.2528 | 0.4579 | 0.4151 |
| 2014 (9-step) | 0.3122 | 0.5865 | 0.4907 |
| 2011 (11-step) | 0.2472 | 0.2898 | 0.3205 |

Table 4.5: The p-values using Student's t-test for Table 4.4.

| Year | Max PN | Max DN | # of Nodes | # of Samples |
|---|---|---|---|---|
| 1992 (5-step) | 0.088 | * 0.026 | * 0.031 | 35 (40) |
| 2013 (7-step) | ** 0.001 | *** | *** | 49 (50) |
| 2007 (9-step) | * 0.235 | *** | ** 0.004 | 57 (60) |
| 2012 (9-step) | 0.086 | ** 0.001 | ** 0.004 | 47 (50) |
| 2014 (9-step) | * 0.029 | *** | *** | 49 (50) |
| 2011 (11-step) | 0.097 | 0.051 | * 0.030 | 46 (50) |

If value was under 0.05 and 0.01 then we use "*" and "**" before values. And if value was under 0.001 then we use "***". # of samples show the used samples (the actual samples in a competition).

in 2013, the rank correlations show a good relationship between the ranking and the search indicators. Especially, the maximum disproof number shows the best correlation coefficient among three values, and the next one is the number of visiting nodes and the maximum proof number. Similarly as the 2013 competition, the 9-step competitions held in 2007, 2012 and 2014 show the good rank correlation coefficient values. The maximum disproof number also shows the best correlation, and the next one are the number of nodes and the maximum proof number as the same as the 7-step competition. The correlation of the maximum proof number in 2012 is removed only in 9-step tsume-shogi competitions. The 11-step tsume-shogi competition in 2011 does not show any good rank correlations. The highest correlation is shown by the number of visiting nodes, but its value is small. The maximum proof and disproof number are rejected by t-test, so the search indicators have no significant relationships with judging tsume-shogi in 11-step competition.

The 2011 competition shows the exceptional result, but other competitions show almost the same results. The maximum disproof number shows the best relationship to the ranking in this experiment, and the number of nodes shows the second best, and the maximum proof number shows the worst correlation. The rank correlation of the maximum proof number is very small, and these are rejected in some cases. In the 2011 competition, the rank correlations show the different results from 5 to 9-step tsume-shogi competitions. So, in the short step tsume-shogi competition, the maximum disproof number relates well to the assessment of tsume-shogi, but this fact cannot be seen in 11-step tsume-shogi competition. The maximum proof number shows the difficulty of the Henka, the maximum disproof number shows the difficulty of Magire, and the number of visiting nodes shows the whole difficulty of tsume-shogi. So, in short step tsume-shogi competitions, the difficulty of Magire is regarded important for

evaluating the whole difficulty of tsume-shogi. And, the difficulty of Henka is not focused so much for evaluating tsume-shogi. We understand that those who participated the competition as evaluators compare tsume-shogi with a focus on the difficulty of Magire, and try to order the evaluating of tsume-shogi. But, the rank correlation coefficients of the maximum disproof number show around 0.5. So the maximum disproof number can relate to the evaluating of tsume-shogi well, but with just that evidence, it is difficult for showing the evaluation perfectly.

### 4.1.4 Comparison with Previous Works

In the experiments performed in this section, we realized that we can use the search indicators for evaluating the aesthetics of tsume-shogi. Here, we compare our results with the previous works done by Koyama [35]. Koyama focused on tsume-shogi competitions just like ours, and collected the following principles of tsume-shogi.

- The number of pieces on the board for the attacker's side at the initial position.
- The number of pieces on the board for the defender's side at the initial position.
- The number of pieces in the attacker's hand at the initial position.
- The location of the defender's King at the initial position.
- The number of pieces around the defender's King at the mating position. This value is called "the physical closeness".
- The number of times for moving the defender's King.
- The number of times for sacrificing pieces.
- The number of times for checking the defender's King using a piece which has not been moved. This is called "Aki-Oute".
- The number of times for checking the defender's King using two and more pieces in a move. This is called "Ryo-Oute".

We compare the above values with the search indicators. Koyama collected data of tsume-shogi in addition to above values, but we ignore these in our experiments because of reproducibility. For comparison, we used Weka [14] which has been developed by University of Waikato, New Zealand. It is a system which we can use the machine learning algorithms easily and freely. Koyama used many values for analyzing tsume-shogi competitions, but the correlations between each value and the ranking is very low when we confirmed it then. He analyzed the evaluated tsume-shogi using above values, and try to find the characteristics related to the evaluation, so each simple value may not be able to relate to the assessment of tsume-shogi well. So, we try to mix those values, but it is difficult to combine each value effectively. For this problem, we used Weka and tried to find good combination of those values using the decision tree. And also, we tried to build up the decision tree using the search indicators and compare our results with Koyama's.

For making the decision tree, we prepared the data as follows. First, we picked up the data of the 9-step tsume-shogi competition held in 2007. Because the 2007 competition has the large samples ($n = 57$) in our experiments. We expected to combine each competition for increasing the number of samples. However, there is a potential

Table 4.6: The summary of built up decision trees using J4.8.

| | Koyama | The search indicators | Both |
|---|---|---|---|
| Correctly Classified Instances | 40.3509 % | 54.386 % | 52.6316 % |
| Incorrectly Classified Instances | 59.6491 % | 45.614% | 47.3684 % |
| Kappa statistic | 0.1061 | 0.3189 | 0.2878 |
| Mean absolute error | 0.4117 | 0.3321 | 0.3253 |
| Root mean squared error | 0.5703 | 0.4813 | 0.5327 |
| Relative absolute error | 92.6314 % | 74.724 % | 73.2037 % |
| Root relative squared error | 120.912 % | 102.0617 % | 112.9488 % |
| Total Number of Instances | 57 | 57 | 57 |

that the score of each tsume-shogi depends on each competition, so we use only one competition in this experiment. Second, we classify the ranking such as first, second and third for making the decision tree. This means that the first is the set of tsume-shogi ranked first o 20th, and second is the set of tsume-shogi ranked 21st to 40th and third is the set of tsume-shogi ranked 41st to 60th in the 2007 competition. In this experiment, we tried to find the element which is related to the decision process.

Using Weka, we implemented J4.8 algorithm with default options. The input data are the Koyama's data set shown above, the search indicators, and both set. Koyama's data set are using 10 attributes, and the search indicators have 3 attributes. Every input values are discrete values. The output data are the classes such as 1st, 2nd and 3rd, decided by the ranking. This means that 1st is the set of tsume-shogi ranked 1st to 20th, and 2nd is the set of tsume-shogi ranked 21st to 40th and 3rd is the set of tsume-shogi ranked 41st to 60th in the 2007 competition. We have chosen the learning data set which contains 57 tsume-shogi problems in the 2007 competition, because it has the largest samples in our experiments. We expected to combine each competition for increasing the number of samples. However, there is a potential that the score of each tsume-shogi depends on each competition, so we use only one competition in this experiment. We used cross-validation for testing, and it folds 10 samples. Then, the results of Weka are shown in Tables 4.6 and 4.9.

First, we focused on the summary of the outcomes of decision tree. About the correctly classified instances show around 50% in each input set. Koyama's set shows the worst correctly classified instances and the search indicators show the improved values. When we use the search indicators only, then the classified instances show the biggest value. Next, we focused on the values such as the precision, the recall and the f-measure. If we compare the precision and the recall values, then we can see that there are not so large differences. When using the search indicators only, the 1st and 3rd class have the relationship such as the trade-off about the precision and the recall. For comparing the quality of decision tree, we focused on the f-measure calculated by the precision and the recall values. If we compare the Koyama's set and the search indicators, then we see that the search indicator shows better than Koyama's set in every cases. Especially, the 1st and 3rd classes have large f-measure values. For the 2nd class, Koyama's set and the search indicators cannot show good value in this experiment. If we combine the Koyama's set and the search indicators, then we can get balanced f-measure values in each class. The f-measure values of 1st and 3rd class are decreased, but 2nd class shows the improved value than using just one set. So, the search indicators can support to classify tsume-shogi well for 1st and 3rd, but it lacks the information for classifying the 2nd class. If we add the Koyama's set to the search

Table 4.7: The precision values for each class using each input data

| Class | Koyama | The search indicators | Both |
|---|---|---|---|
| 1st | 0.500 | 0.538 | 0.588 |
| 2nd | 0.278 | 0.375 | 0.455 |
| 3rd | 0.421 | 0.733 | 0.556 |

Table 4.8: The recall values for each class using each input data

| Class | Koyama | The search indicators | Both |
|---|---|---|---|
| 1st | 0.556 | 0.778 | 0.556 |
| 2nd | 0.250 | 0.300 | 0.500 |
| 3rd | 0.421 | 0.579 | 0.526 |

Table 4.9: The F-measure values for each class using each input data

| Class | Koyama | The search indicators | Both |
|---|---|---|---|
| 1st | 0.526 | 0.636 | 0.571 |
| 2nd | 0.263 | 0.333 | 0.476 |
| 3rd | 0.421 | 0.647 | 0.541 |

indicators, then the weakness of the search indicators is covered and it could classify tsume-shogi well.

### 4.1.5 Discussion

The discussion of the experiments is divided into three main topics: (1) the method used in our experiments, (2) the results of our experiments, and (3) the relationship between the search indicators and the aesthetics of tsume-shogi.

First, as for the method used in our experiments we have to consider the search algorithms. In this study, we used the original PN-search for reproducibility. The search indicators such as the maximum proof and disproof numbers depend on the search algorithms. So, we cannot guarantee that the other various PN-search algorithms can show the same results. But, the amount of the maximum proof and disproof numbers depend on the shape of the proving tree. If there are two tsume-shogi (say A and B), and A has the higher complexity of Magire than B, then we expect that the maximum disproof number of A takes larger values than B. Thus, even if we use different algorithms, then the relationship of the recorded search indicators in each tsume-shogi may not be so changed. In our experiment, we can say that the maximum disproof number which is used in PN-search can show a good correlation for the ranking of the competitions. And the search indicators using PN-search show better correlation than Koyama's works. The question "Which search algorithm can show better results than the original PN-search?", is one of the important future works.

Second, we mention about the results of our experiments. In our experiments, the maximum disproof number shows a good correlation to the ranking of the competitions. And, because the maximum disproof number relates to the difficulty of Magire in tsume-shogi, those who participated the competition as evaluators regarded the Magire as an important factor in short step tsume-shogi competition. Then, we have questions such as "Why people did not focus on the maximum disproof number in 11-step tsume-shogi competition?". In other words, the disproof numbers are more important elements in the short step tsume-shogi competition than longer step tsume-shogi com-

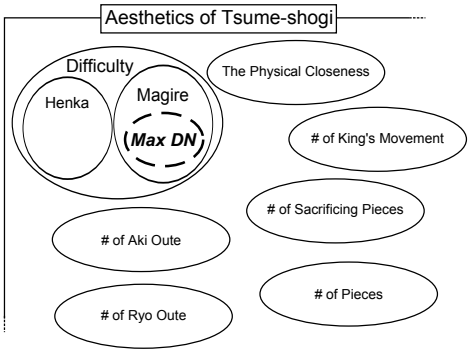



Figure 4.4: The relationship between aesthetics of tsume-shogi, difficulty of Magire and maximum disproof number.
Magire is the attacker's plausible candidate moves. Henka is the defender's plausible candidate moves.

petition. For this question, we expected that it relates to the difficulty of composing tsume-shogi. We thought that it is difficult to include arts, techniques and the beautiful arranging of the pieces for the short step tsume-shogi. So, players may regard the difficulty of tsume-shogi as an important factor. In contrast, tsume-shogi with long step solution sequence can include many elements related to the aesthetics. Thus, evaluators may consider those elements instead of the difficulty only. Therefore, the disproof number is one of the important elements in tsume-shogi, and it can be true only in the case of the short step tsume-shogi competition.

Third, we consider the relationship between the search indicators and the aesthetics. In our experiments, we classified tsume-shogi into 1st, 2nd and 3rd classes based on each ranking, i.e., the high, middle and low ranked tsume-shogi. In the test using Weka, the input with the search indicators can classify the 1st and 3rd class tsume-shogi well. So, the search indicators can judge the highly evaluated tsume-shogi and the low evaluated tsume-shogi. However, the search indicator cannot find 2nd ranked tsume-shogi well. If we add the Koyama's data set to the search indicators, then it improves to find the middle evaluated tsume-shogi. So, the search indicators are the values related to the assessment of tsume-shogi, and these are the principles of evaluating the aesthetics of tsume-shogi. Additionally, the maximum proof number relates to the difficulty of Henka, and the maximum disproof number relates to the difficulty of Magire in tsume-shogi. So, we can conclude that the search indicators show the difficulty of tsume-shogi, the search indicators relate to the assessment of tsume-shogi, and the difficulty of tsume-shogi is one of the principles for evaluating the aesthetics of tsume-shogi. This conclusion is depicted in Figure 4.4.

## 4.2 Concluding Remarks

Below we summarize our main contributions in this study.

- It seems a promising approach to focus on the relationship between aesthetic and complexity indicated by the proof and disproof numbers for the assessment of tsume-shogi.

- The indicator of the disproof number looks more promising than other factors

such as the proof number and the number of visiting nodes during search when using tsume-shogi with relatively short steps in the competition. This means that the difficulty of Magire is more critical component for aesthetic.

- Our experiments show that it is possible to distinguish highly recognized tsume-shogi from normal problems based on the disproof number. This performance is better than the previous works done by Koyama.
- Our approach is hard to evaluate mid-level tsume-shogi appropriately. But, some other elements related to the aesthetics of tsume-shogi support to solve this problem.

# Chapter 5

# Conclusion

In this chapter, we give the conclusion in this thesis. We answer our research questions and problem statement. Then, some future works are discussed.

## 5.1 Summary

Below we summarize our research results of each chapter.

- **Chapter 2: Solving Game Positions and Search Indicators**
  In this chapter, we studied the related works to the topics in this thesis, including the variants of the AND/OR tree search and new approach with respect to conspiracy number search. We focused on the search indicators and its elements such as definition, characteristics, behavior and effectiveness. Then, it is found that if we can explain the search indicators clearly, then the new perspectives of the search indicators are given.

- **Chapter 3: Deep Proof-Number Search**
  The seesaw effect problem is caused by PN-search using the proof number and the best-first search. To tackle this problem, we tried to change the behavior of the search indicators. We proposed a new search algorithm named DeepPN, which can adjust the rate of best-fist manner and depth-first manner. By this method, DeepPN is able to reduce the seesaw effect problem in Othello endgame positions and Hex. These experimental results show that the search indicators with different behaviors make new benefits.

- **Chapter 4: Aesthetics of Mating Problems and Search Indicators**
  We assumed that the proof number is able to show the degree of difficulty for solving tsume-shogi. With this idea, we tried to analyze the aesthetics of tsume-shogi. Then, it is found that difficulty of Magire and the ranking of tsume-shogi have the good correlations. The search indicators are able to judge the assessment of tsume-shogi in some cases.

## 5.2 Answers to RQ1 and RQ2

We perceived the potential of the search indicators. Thus, we answer our research questions below.

**RQ1**
One of the achievements in this thesis is that the search indicators with the new behaviour are able to solve the issues of the search algorithms. To be more precise, the search indicators have correct behaviours fitting to each game. If the behaviour of the search indicators is not appropriate, then the benefits of the original search algorithms are also insufficient. In contrast, if the behaviour of the search algorithms fit to a given game, then the search algorithms are able to mitigate the issues, and those works are changed to well accordingly. So, we conclude that the search algorithms trusted that the most efficient have the potential of composing more efficient search algorithms fitted each game. Then, the search indicators can support to give insightful information for improving the original algorithms.

**RQ2**
We found the relationships between the search indicators and the aesthetics of tsume-shogi. New perspective of proof and disproof numbers are able to discriminate the assessment of tsume-shogi. This outcome was proposed by three steps; 1) studying the original search indicators, 2) finding the new perspective of the search indicators and 3) considering the target to apply the new perspective. We assumed that the most important step is 3). In our case, 1) we considered that the proof number can show the difficulty of the puzzles, 2) we apply this idea to analyse the tsume-shogi competition, and 3) we produced that the disproof numbers are able to discriminate the aesthetics of tsume-shogi.

## 5.3 Answer to Problem Statement

The search indicators were used only for efficient search, but our research implies that there are other potentials. The important utilization are the fitting to the game, the evaluating of the mating problems. In any utilization, we respect the original definition of the search indicators, but it just changes the behavior or the viewpoint. We assume that we can find additional utilization to our outcomes if we focus on the search indicator and consider the new perspective then. Now, we are facing the turning point of the game research. Recently, the research domains different from building strong computer players have started actively. The search indicators might help to answer those different purposes. In this thesis, we provided some new perspectives for those researches.

## 5.4 Future Work

Future works are given below.

- As for the DeepPN algorithm, there are three problems to tackle in the future, which include two technical problems and one problem related to the effectiveness of the DeepPN. Currently, the DeepPN uses a constant value for adjusting own work. The best constant value depends on each game, then we cannot take

the best value automatically so far. DeepPN was composed based on the PN-search, but we know the modified PN-search called Df-pn. So, the two technical problems are: (1) the way of adding a system which can adjust own work automatically, and (2) adopting the Df-pn method for more efficient search. As for another problem, we did not conduct the evaluation experiments to see the performance of DeepPN for many other games except Othello and Hex. We might see further feedback while collecting the experimental data using other games for this problem.

- Analyzing tsume-shogi competitions using the proof number was presented in this thesis with some good results, but some problems still remain. One of the largest concerns is to see the difference between computer and human. Our experimental data were collected by computers. Thus, our research can evaluate the assessment of tsume-shogi by computer-like only, and we did not understand the human assessment way. One of the hints for solving this problem is to compose a mating problem and submit it to the competition. Human experts would evaluate our mating problems while comparing with other problems created by human composers.

# Bibliography


[1] U. M. Alberto Martelli. Additive AND/OR graphs. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, IJCAI'73, pages 1–11, San Francisco, CA, USA, 1973. Morgan Kaufmann Publishers Inc.

[2] L. V. Allis. A knowledge-based approach to connect-four. The game is solved: White wins. In *Master thesis, Vrije Universiteit*, 1988.

[3] L. V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, University of Limburg, 1994.

[4] L. V. Allis, M. van der Meulen, and H. J. Van Den Herik. Proof-number search. *Artificial Intelligence*, 66(1):91–124, 1994.

[5] D. M. Breuker. *Memory versus Search in Games*. PhD thesis, Maastricht University, 1998.

[6] M. Campbell, A. J. Hoane, Jr., and F.-h. Hsu. Deep Blue. *Artificial Intelligence*, 134(1-2):57–83, Jan. 2002.

[7] K. Chellapilla and D. B. Fogel. Evolving neural networks to play checkers without relying on expert knowledge. *IEEE Transactions on Neural Networks*, 10(6):1382–1391, Nov. 1999.

[8] R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *5th International Conference, CG 2006*, pages 72–83. Springer-Verlag, 2006.

[9] O. E. David, H. J. van den Herik, M. Koppel, and N. S. Netanyahu. Genetic algorithms for evolving computer chess programs. *IEEE Transactions on Evolutionary Computation*, 18(5):779–789, 2014.

[10] J. U. Erik van der Werf, Jaap van den Herik. Solving Go on small boards. *ICGA Journal*, 26(3):92–107, 2003.

[11] N. Fujii, Y. Sato, H. Wakama, K. Kazai, and H. Katayose. Evaluating human-like behaviors of video-game agents autonomously acquired with biological constraints. In *Advances in Computer Entertainment - 10th International Conference, ACE 2013, Boekelo, The Netherlands, November 12-15, 2013. Proceedings*, pages 61–76, 2013.

[12] J. Fürnkranz. Machine learning in games: A survey. In J. Fürnkranz and M. Kubat, editors, *Machines that Learn to Play Games*, chapter 2, pages 11–59. Nova Science Publishers, Huntington, NY, 2001.

[13] R. D. Greenblatt, D. E. E. III, and S. D. Crocker. The greenblatt chess program. In *AFIPS Fall Joint Computing Conference*, volume 31 of *AFIPS Conference Proceedings*, pages 801–810. AFIPS / ACM / Thomson Book Company, Washington D.C., 1967.

[14] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.

[15] E. A. Hansen and S. Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.

[16] J. Hashimoto. *A Study on Game-Independent Heuristics in Game-Tree Search.* PhD thesis, Japan Advanced Institute of Science and Technology, 2011.

[17] K. Hoki and T. Kaneko. Large-scale optimization for evaluation functions with minimax search. *J. Artif. Int. Res.*, 49(1):527–568, Jan. 2014.

[18] K. Hoki, T. Kaneko, A. Kishimoto, and T. Ito. Parallel dovetailing and its application to depth-first proof-number search. *ICGA Journal*, pages 22–36, 2013.

[19] H. Iida. The challenge of shogi: a new champion Bonanza, in Special Issue on A New Trend in Computer Shogi. *IPSJ Magazine (in Japanese)*, 47(8):890–892, Aug. 2006.

[20] H. Iida, M. Sakuta, and J. Rollason. Computer shogi. *Artificial Intelligence*, 134(1-2):121–144, Jan. 2002.

[21] A. Iqbal. Is aesthetics computable? *ICGA Journal*, 29(1):32–39, 2006.

[22] A. Iqbal. Aesthetics in mate-in-3 combinations, Part I Combinatorics and weights. *ICGA Journal*, 33(3):140–148, 2010.

[23] A. Iqbal. Aesthetics in mate-in-3 combinations, Part II Normality. *ICGA Journal*, 33(4):202–211, 2010.

[24] A. Iqbal, M. Guid, S. Colton, J. Krivec, S. Azman, and B. Haghighi. *The Digital Synaptic Neural Substrate: A New Approach to Computational Creativity.*, volume abs/1507.07058. SpringerBriefs in Cognitive Computation, 2015.

[25] A. Iqbal, H. M. J. F. van der Heijden, M. Guid, and A. Makhmali. Evaluating the aesthetics of endgame studies: A computational model of human aesthetic perception. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(3):178–191, 2012.

[26] K. Ito. *Tsumuya-tsumazaruya, Shogi-Muso, Shogi-Zuko*. Tokyo: Heibonsha, 1975. in Japanese.

[27] Y. Kadowaki. *Zoku-tsumuya-tsumazaruya. Koten shogi no keihu*. Tokyo: Heibonsha, 1978. in Japanese.

[28] T. Kaneko. Evaluation of real-time commentary generated by computer shogi program. *IPSJ Journal*, 53(11):2525–2532, nov 2012. in Japanese.

[29] M. N. A. Khalid, E. M. Ang, U. K. Yusof, H. Iida, and T. Ishitobi. Identifying critical positions based on conspiracy numbers. In B. Duval, H. J. van den Herik, S. Loiseau, and J. Filipe, editors, *ICAART (Revised Selected Papers)*, volume 9494 of *Lecture Notes in Computer Science*, pages 100–127. Springer, 2015.

[30] M. N. A. Khalid, U. K. Yusof, H. Iida, and T. Ishitobi. Critical position identification in games and its application to speculative play. In S. Loiseau, J. Filipe, B. Duval, and H. J. van den Herik, editors, *ICAART (2)*, pages 38–45. SciTePress, 2015.

[31] A. Kishimoto. *Correct and Efficient Search Algorithms in the Presence of Repetitions*. PhD thesis, University of Alberta, January 2005.

[32] A. Kishimoto. Search versus knowledge for solving life and death problems in Go. In *Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 1374–1379. AAAI Press, 2005.

[33] A. Kishimoto, M. H. Minands, M. Müller, and J.-T. Saito. Game-tree search using proof numbers: The first twenty years. *ICGA Journal*, 35(3):131–156, 2012.

[34] D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

[35] K. Koyama and K. Yasuhito. Quantitative analysis of impression on tsume - shogi (mating problems of japanese chess). *SIGAI of Information Processing Society of Japan*, pages 29–42, 1993. in Japanese.

[36] L. Lister. *Analysis of the Conspiracy Numbers Algorithm*. Technical report (University of Alberta. Department of Computing Science). Thesis (M.Sc.)– University of Alberta, 1990.

[37] U. Lorenz, V. Rottmann, R. Feldmann, and P. Mysliwietz. Controlled conspiracy number search. *ICCA Journal*, 18(3):135–147, 1995.

[38] D. A. McAllester. Conspiracy numbers for min-max search. *Artificial Intelligence*, 35(3):287–310, July 1988.

[39] D. A. McAllester and D. Yuret. Alpha-beta-conspiracy search. *ICGA Journal*, 25(1):16–35, 2002.

[40] A. Nagai. A new AND/OR tree search algorithm using proof number and disproof number. In I. Frank, H. Matsubara, M. Tajima, A. Yoshikawa, R. Grimbergen, and M. Müller, editors, *Complex Games Lab Workshop*. Electrotechnical Laboratory, Machine Inference Group, Tsukuba, Japan, 1998.

[41] A. Nagai. A new depth-first search algorithm for AND/OR trees. Master's thesis, The University of Tokyo, 1999.

[42] A. Nagai. *Df-pn Algorithm for Searching AND/OR Trees and its Applications*. PhD thesis, University of Tokyo, 2002.

[43] N. J. Nilsson. Searching problem-solving and game-playing trees for minimal cost solutions. In *IFIP Congress (2)*, pages 1556–1562, 1968.

[44] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1980.

[45] F. Okabe. Application for solving tsume shogi problem by route branch number. *The 20th Game Programming Workshop*, 2005(15):9–16, nov 2005. in Japanese.

[46] J. Pawlewicz and L. u. Lew. Improving depth-first pn-search: 1+ε trick. In H. van den Herik, P. Ciancarini, and H. Donkers, editors, *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pages 160–171. Springer Berlin Heidelberg, 2007.

[47] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[48] A. Plaat, J. Schaeffer, W. Pijls, and A. De Bruin. Best-first fixed-depth game-tree search in practice. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 1 of *IJCAI'95*, pages 273–279, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[49] A. Plaat, J. Schaeffer, W. Pijls, and A. de Bruin. SSS* = alpha-beta + TT. *CoRR*, abs/1404.1517, 2014.

[50] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[51] A. Sadikov, M. Mozina, M. Guid, J. Krivec, and I. Bratko. Automated chess tutor. In H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, editors, *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 2006.

[52] J. Schaeffer. Conspiracy numbers. *Artif. Intell.*, 43(1):67–84, Apr. 1990.

[53] J. Schaeffer. Game over: Black to play and draw in checkers. *ICGA Journal*, 30(4):187–197, 2007.

[54] J. Schaeffer. *One Jump Ahead: Computer Perfection at Checkers*. Springer-Verlag, 2009.

[55] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, and M. M. R. Lake. Checkers is solved. *Science*, pages 1518–1522, 2007.

[56] M. Seo. The C* algorithm for AND/OR tree search and its application to a tsume-shogi program. Master's thesis, University of Tokyo, 1995.

[57] M. Seo, H. Iida, and J. W. H. M. Uiterwijk. The PN*-search algorithm: Application to tsume-shogi. *Artif. Intell.*, 129(1-2):253–277, 2001.

[58] C. E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.

[59] Shi-JimYen, Tsan-ChengSu, Jr-ChangChen, and Shun-ChinHsu. Solving 6x6 Othello on volunteer computing system. In *The 19th Game Programming Workshop 2014*, volume 2014, pages 117–121, oct 2014.

[60] *Shogi Sekai*. Japan Shogi Association, Sept. 1992. in Japanese.

[61] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[62] *Tsume-shogi Paradise*. Japan Tsume-shogi Association, Mar 2014. in Japanese.

[63] T. Ueda, T. Hashimoto, J. Hashimoto, and H. Iida. Weak proof-number search. In H. van den Herik, X. Xu, Z. Ma, and M. Winands, editors, *Computers and Games*, volume 5131 of *Lecture Notes in Computer Science*, pages 157–168. Springer Berlin Heidelberg, 2008.

[64] H. J. van den Herik and M. H. Winands. Proof-number search and its variants. In *Oppositional Concepts in Computational Intelligence*, pages 91–118. Springer Berlin Heidelberg, 2008.

[65] M. van der Meulen. Conspiracy-number search. *ICCA Journal*, 13(1):3–14, 1990.

[66] J. von Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100:295–320, 1928.

[67] T. I. Vu Ngoc Quang, Terrillon Jean-Christophe Georges and H. Iida. Using conspiracy numbers for improving move selection in minimax game-tree search. In *The 8th International Conference on Agents and Artificial Intelligence (ICAART 2016)*, volume 2, pages 400–406, 2015.

[68] O. Wilson, A. Parker, and D. Nau. Error minimizing minimax: Avoiding search pathology in game trees. In *In Proceedings of International Symposium on Combinatorial Search (SoCS-09*, 2009.

[69] WZebra. *http://radagast.se/othello/download.html*.

[70] K. Yonenaga. *ware yaburetari kompiyu ta kisen no subete o kataru*. Chuokoron-Shinsha, 2012. in Japanese.

# Publications


T. Ishitobi, H. Iida. Analysis of Tsume-Shogi Problems using Proof-Number, *The 17th Game Programming Workshop*, pp. 163–166, Hakone, Japan, in Japanese, 2012.

T. Ishitobi, A. Cincotti, H. Iida. Shape-Keeping Technique and Its Application to Checkmate Problem Composition, *Artificial Intelligence and Game Aesthetics: the 2013 AIIDE Workshop (WS-13-19)*, pp. 7–10, 2013.

T. Ishitobi, A. Cincotti, H. Iida. Shape-Keeping Heuristics, *Computer Games and Intelligence Workshop*, Yokohama, 2013.

T. Ishitobi, A. Plaat, H. Iida, J. van den Herik. Reducing the Seesaw Effect in Proof Number Search, *14th International Conference Advances in Computer Games 2015*, Lecture Notes in Computer Science 9525, Springer, pp. 185–197, 2015.

T. Ishitobi, H. Iida, The Mating Problems Composition with Insightful Knowledge, *The 20th Game Programming Workshop*, pp. 126–131, Hakone, Japan, in Japanese, 2015.

H. Iida, T. Nakagawa, S. Sone, A. Muangkasem, T. Ishitobi. Safety Lead Curve and Entertainment in Games, ICTA2011, Information Systems and Technologies. 2011.

H. Iida, T. Nakagawa, A. Horikawa, S. Sone, A. Muangkasem, T. Ishitobi. On A Construction Procedure of Pyramids, AlaSim International 2012.

H.Minatoya, H.Iida, T.Nakagawa, T. Ishitobi, T.N.Nossal, T.Suzuki. Judo and Information Dynamics, AlaSim International 2012.

H.Iida, T.Nakagawa, N.Q.Huy, S.M.Hasai, A.N.Husna, A.Muangkasem, S.Sone, T. Ishitobi. Game information dynamics and its application to Congkak and Othello, International Conference on Information Society (i-Society 2012), pp. 415–422, 2012.

H.Iida, T.Nakagawa, S.Sone, A.Muangkasem, T. Ishitobi, T.Okaneya. Information Dynamics in FIFA Women's World Cup Germany 2011 Final, Proceedings of NICOGRAPH International 2012, pp.44–49.

M. N. A. Khalid, U. K. Yusof, H. Iida and T. Ishitobi, Critical Position Identification in Games and Its Application to Speculative Play, the 7th International Conference on Agents and Artificial Intelligence (ICAART 2015), 2, pp. 38–45, Lisbon, Portugal, 2015.

V. N. Quang, J.C. Terrillon, T. Ishitobi and H. Iida, Using Conspiracy Numbers for Improving Move Selection in Minimax Game-Tree Search, In Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016) - Volume 2, pp. 400–406, 2016.

M. N. A. Khalid, E. M. Ang, U. K. Yusof, T. Ishitobi and H. Iida. Identifying Critical Positions based on Conspiracy Numbers, In B. Duval et al. (Eds.): ICAART 2015, LNAI 9494, pp. 100-127, 2015.

# Appendix A

# Additional Experiments with DeepPN

In this appendix, we show the additional experiments of DeepPN. We show two data mainly; (1) the experimental results using Othello endgame positions composed by WZebra [69], and (2) the relationship between DeepPN and other search algorithms.

## A.1 Othello Experiments using WZebra

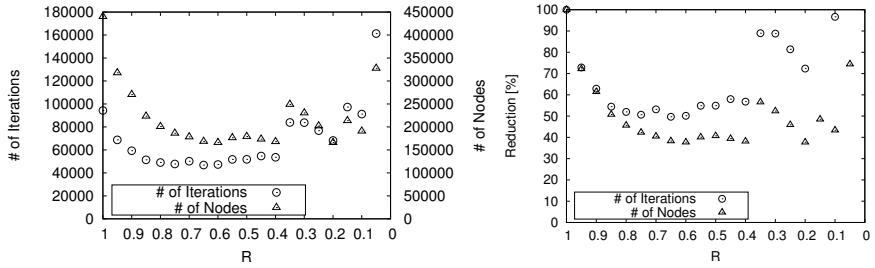
WZebra is one of the popular and strong computer players of Othello. In Chapter 3, we used 1000 Othello endgame positions using randomness for measuring the performance of DeepPN. However, we had a question that if we use more realistic endgame positions, then whether DeepPN can also work well or not. For this question, we prepare 100 Othello endgame positions using WZebra. We setup WZebra as shown in Table A.1. The prepared endgame positions are composed by self-play of WZebra, and we used 44 moves from starting position. An endgame position is composed by 4 initial stones and 44 stones played by WZebra, so this is a 16-ply Othello endgame position.

First, we show, in Figure A.1, the experimental results in Chapter 3 and the experimental results using WZebra in Figure A.2. Figure A.2a shows the variation of (1) the number of iterations and (2) the number of nodes, and Figure A.2b shows the reduction rates based on the result when $R = 1.0$. Each point represents a median value calculated from the results of 100 Othello endgame positions using WZebra. The case $R = 1.0$ shows the results of PN-search, and this value is the base for comparison.

In the last experiments using random Othello endgame positions (called random

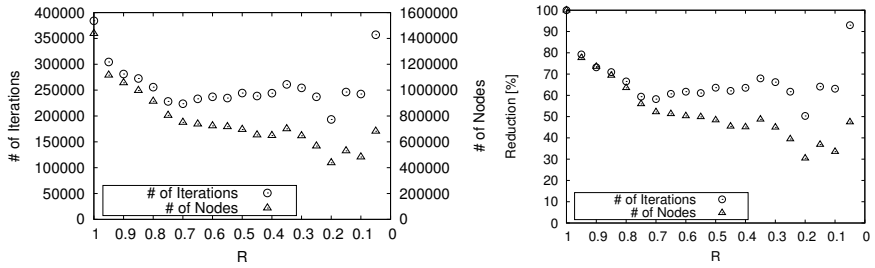
Table A.1: Setting of WZebra

| Name of Setting | Chosen Option |
|---|---|
| Search Depth | 24 moves + last 26 perfect |
| Time | No time limit (default) |
| Book | Use opening book (default) |
| Midgame | Small randomness (default) |
| Position table | 8MB |




(a) The variation (b) The reduction rate

Figure A.1: Othello: The # of iterations and # of nodes.
$R = 1.0$ is PN-search, $R = 0.0$ is depth-first search, and $1.0 > R > 0.0$ is DeepPN. Lower is better.



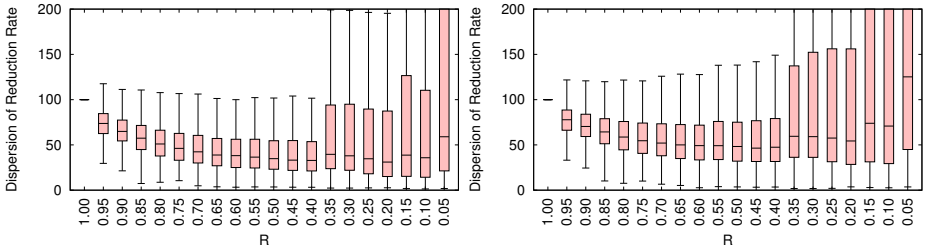

(a) The variation (b) The reduction rate

Figure A.2: Othello with WZebra: # of iterations and # of nodes for 16-ply Othello with WZebra.
$R = 1.0$ is PN-search, $R = 0.0$ is depth-first search, and $1.0 > R > 0.0$ is DeepPN. Lower is better.

positions), the reduction rates show that the number of iterations decreases to 50% and the number of nodes decreases to 40%. In this experiment using Othello endgame positions composed by WZebra (called WZebra positions), the reduction rates show that the number of iterations decreases to 50% and the number of nodes decreases to 30%. However, in WZebra positions, the best reduction rates exist in $R = 0.20$ differ from random positions. The decreasing reduction rates between $R = 1.0$ and $R = 0.7$ looks like the same as the result of random positions, but the amount of decreasing is little smaller than the random positions. After $R = 0.7$, the number of iterations starts to increase, but the number of nodes are still decreasing. In $R = 0.35$, the both values change to worth an instant, but after $R = 0.35$, both values decrease sharply than before. Finally, DeepPN shows the best result in $R = 0.20$, and after it, works of DeepPN change to worth. If we compare the results of WZebra positions with random positions, then the results of WZebra positions show the more stable efficient works than the results of random positions.

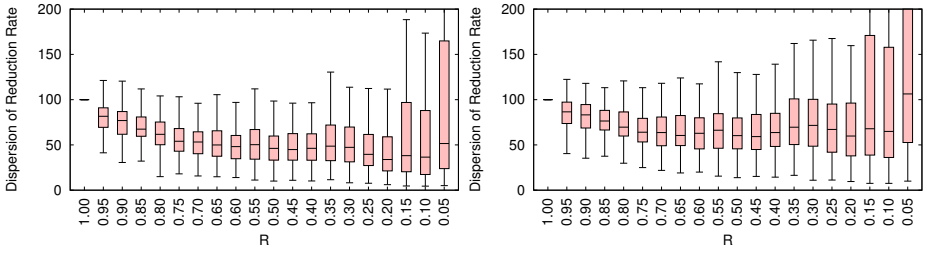
Next, we show, in Figure A.3, the dispersion of reduction rates using random positions and WZebra positions in Figure A.4.

There are the box-plots about the dispersion of reduction rates in each $R$. These plots are calculated by the 100 reduction rates compared with the results of $R = 1.0$, so the all reduction rate in $R = 1.0$ is 100%. From the lower, the graphs show the




(a) # of iterations (b) # of nodes

Figure A.3: Othello: The dispersion of reduction rates for # of iterations and # of nodes




(a) # of iterations (b) # of nodes

Figure A.4: Othello with WZebra: The dispersion of reduction rates for # of iterations and # of nodes

minimum reduction rate, the first quartile, the median reduction rate, the third quartile, and maximum reduction rate. For easy to see, the outliers are not printed.

In WZebra positions, between $R = 0.95$ and 0.20, the many reduction rates exist under the 100% and the dispersions are more narrow than the results of random positions. After $R = 0.25$, the reduction rates and its dispersion change to very worth like the results of random positions. Thus, we can see that DeepPN works well stably in practical positions. About this fact, we considered that the random positions have the settlement positions before the game end. This means that if strong players play the Othello then the timing of the settlement cannot be expected before the game end. In the 16-ply Othello endgame positions, we have to put 16 stones for proving the game outcomes. So, DeepPN works well when it uses a strong depth-first manner then. However, the endgame positions using randomness include some special settlement positions such as a game where player loses all stones before the game end. In this case, the shortest correct path exists before 16 depth, and DeepPN with strong depth-first manner overlooks it like Hex experiments. Therefore, if we try to decide the best $R$, then we have to consider the depth of the settlement positions and the characteristics of the game. Anyway, in WZebra positions, DeepPN works well around $R = 0.60$ like random positions.

## A.2 DeepPN with Previous Works

In this section, we compare DeepPN with previous works. We prepare DeepPN, Df-pn, 1 + ε trick with Df-pn, and Dfpn+ solvers. Here, we did not use some techniques such as the transposition table, and we refer the standard algorithms in each solver. The situations of the experiments are the same as Chapter 3. We prepared 1000 random Othello endgame positions with 44 legal moves. And we prepared Hex with 4 x 4 board (called Hex(4)) which has no stones and the random ordering for getting each legal moves. For Othello experiment, we use the $R = 0.60$ for DeepPN, because we found that this value is the best for solving 16-ply Othello endgame positions. Similarly, we use the $R = 0.95$ for the Hex(4) in DeepPN. And, we prepare DeepPN using $R = 1.00$ for comparing. About the 1 + ε trick, we adjusted ε between 0.0 and 1.0 for finding the best value to 16-ply Othello and Hex(4). Then, we found that $\varepsilon = 1/8$ is the best for 16-ply Othello and $\varepsilon = 1$ is the best for Hex(4). And about the Dfpn+, we adjusted the cost function. In the original paper of Dfpn+, Nagai adjusted the $cost_\phi$ between 0 to -6, so we also tried to adjust the cost function between 0 to -6. Then, we found that $cost_\phi = -1$ is the best for 16-ply Othello endgame position and $cost_\phi = -5$ is the best for Hex(4).

We focus on the values of two concepts, viz. the number of iterations and the number of nodes. The number of iterations is given by counting the number of times for expanding a most-proving node. This value indicates an approximate execution time unaffected by the specifications of a computer. The number of nodes is an indicator of the total number of nodes that are expanded by the search. This value is an approximation of the size of memory needed for solving. We show the experimental results in Tables A.2 and A.3.

Table A.2: Results of each search algorithm using 16-ply Othello endgame positions.

| | The various | | The reduction rates | |
|---|---|---|---|---|
| | Iterations | Nodes | Iterations | Nodes |
| DeepPN ($R = 1.00$) | 94319 | 440233 | 100.00% | 100.00% |
| Df-pn | 90555 | 428416 | 96.01% | 97.32% |
| DeepPN ($R = 0.60$) | 47285 | 166103 | 50.13% | 37.73% |
| 1 + ε ($\varepsilon = 1/8$) | 91459 | 419436 | 96.97% | 95.28% |
| Dfpn+ ($cost_\phi = -1$) | 65690 | 124148 | 69.65% | 28.20% |

Each value shows the median value calculated by 1000 results. The reduction rates are based on the results of DeepPN with $R = 1.00$.

First we focus on the number of iterations and nodes in the experiments of 16-ply Othello endgame position in Table A.2. We set the base for the experimental results of DeepPN using $R = 1.00$ in this table, i.e., the base is the original PN-search for calculating the reduction rates. Df-pn shows almost the same result as DeepPN using $R = 1.00$. Nagai [41] proposed that the manner of Df-pn is the same as the original PN-search. If Df-pn and PN-search met the tie-break situations, i.e., both algorithms had some most-proving nodes in an iteration then each algorithm chooses the different most-proving node in some cases. If we modified the original PN-search to choose the same most-proving node as Df-pn, then both algorithms show the same results. We did not modify the original PN-search, so the experimental results show little differences, but their data is the almost same. Importantly, the actual execution times of both

Table A.3: Results of each search algorithm using Hex(4).

| | The various | | The reduction rates | |
|---|---|---|---|---|
| | Iterations | Nodes | Iterations | Nodes |
| DeepPN ($R = 1.00$) | 2671655 | 24049320 | 100.00% | 100.00% |
| Df-pn | 3086234 | 26348012 | 115.52% | 109.93% |
| DeepPN ($R = 0.95$) | 973190 | 7764919 | 36.43% | 32.29% |
| $1 + \varepsilon$ ($\varepsilon = 1$) | 794448 | 4381527 | 29.74% | 18.22% |
| Dfpn+ ($cost_{\phi} = -5$) | 2483241 | 22660719 | 92.95% | 94.23% |

Each value shows the median value calculated by 10 trial runs. Reduction rates are based on the results of DeepPN with $R = 1.00$.

algorithms are different. In each iteration, PN-search has to start at the root node in any time, but Df-pn can start at an inner node in some cases. So, Df-pn could reduce the cost of tracing the nodes, and it shows the faster execution time than PN-search. Currently, DeepPN is based on PN-search, so its actual execution times are slower than the algorithms based on Df-pn. If we successfully modify DeepPN into Df-pn with the deep value (DeepDFPN), then DeepDFPN would show the same result as DeepPN and it must reduce the execution time like relationship between PN-search and Df-pn. In this experiment, we use the number of iterations for showing the execution time except the cost of tracing nodes. The number of nodes shows the amount of using working memory, and this does not have different meaning in each algorithm.

Df-pn with $1+\varepsilon$ trick ($\varepsilon = 1/8$) cannot show better works than DeepPN using $R = 1.0$ and its result like the results of plane Df-pn. In the paper of $1+\varepsilon$ trick, authors apply it to the Atari Go and Lines of Action, and they show the good results. But, in Othello experiments, $1+\varepsilon$ trick cannot produce any benefit. DeepPN using $R = 0.60$ and Dfpn+ using $cost_{\phi} = -1$ show better results than DeepPN using $R = 1.00$. DeepPN using $R = 0.60$ shows the best (smallest) number of iterations, and Dfpn+ using $cost_{\phi} = -1$ shows the best number of nodes in those results. So, both algorithms can be suitable for solving Othello endgame positions.

Second, we try to focus on the experimental results using the Hex(4). Then, DeepPN using $R = 1.0$ and Df-pn show almost the same results. Df-pn shows a little more worth result than DeepPN using $R = 1.0$, it may be caused by the different manners of both algorithms in a tie-break situation. About our programs, in a tie-break situation, PN-search chooses a most-proving node existed the left-most side, and Df-pn chooses the nearest one from the current focusing node. This difference may create the different result in this experiment. Dfpn+ using $cost_{\phi} = -5$ shows better results than DeepPN using $R = 1.00$, but differences are small. In Othello experiments, Dfpn+ could show good results. But, in this case, the results of Dfpn+ is not so different from PN-search. DeepPN using $R = 0.95$ and Df-pn with $1+\varepsilon$ trick using $\varepsilon = 1$ show well works. $1+\varepsilon$ trick shows the best result in Hex(4), and DeepPN shows the second best result. $1+\varepsilon$ trick could not work well in Othello experiments, but in Hex(4), that work shows almost the three-fold efficiency results. DeepPN using $R = 0.95$ could not amount to $1+\varepsilon$ trick, but the results is well acceptable.

So, the previous works coped with the seesaw effect show good results in each experiment. Dfpn+ shows good results in experiments using Othello endgame positions, and $1+\varepsilon$ trick shows the good work in Hex(4). However, both algorithms cannot work well similarly in Othello and Hex. Then, DeepPN shows the stable works in both ex-

periments, and its results could reach the best works of Dfpn+ and $1+\varepsilon$ trick. So, we could conclude that the DeepPN is a more generic way for reducing the seesaw effect than Dfpn+ and $1+\varepsilon$ trick at least in Othello and Hex. Therefore, the modifying the DeepPN and composing DeepDFPN is one of the important future works.