Spring 2012

Spatial-temporal reasoning applications of computational intelligence in the game of Go and computer networks

Tae-Hyung Kim

Department: Electrical and Computer Engineering

SPATIAL-TEMPORAL REASONING APPLICATIONS OF COMPUTATIONAL INTELLIGENCE IN THE GAME OF GO AND COMPUTER NETWORKS

by

TAE HYUNG KIM

A DISSERTATION

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY
in
ELECTRICAL ENGINEERING

2012

Approved by

D.C. Wunsch II, Advisor
Steven L. Grant
Yahong Rosa Zheng
Jagannathan Sarangapani
Madria Sanjay

PUBLICATION DISSERTATION OPTION

This dissertation is composed of the following three papers which were reformatted in the style used by the university.

The first paper presented in pages 52-66 titled "Recursive and Nonrecursive Algorithms for the Group Size Counting Problem in Computer Go" is accepted to Artificial Neural Networks in Engineering (ANNIE), St. Louis, MO, USA, Nov. 2010.

The second paper presented in pages 67-86 titled "Robotic Go: Exploring a Different Perspective on Human-Computer Interaction with the Game of Go" is accepted to IEEE International Conference on Systems, Man, and Cybernetics, San Antonio, TX, USA, 2009.

The third paper presented in pages 87-104 titled "Reconfigurable Disruption Tolerant Routing via Reinforcement Learning" is accepted to International Joint Conference on Neural Networks, Atlanta, GA, USA, 2009.

ABSTRACT

Spatial-temporal reasoning is the ability to reason with spatial images or information about space over time. In this dissertation, computational intelligence techniques are applied to computer Go and computer network applications. Among four experiments, the first three are related to the game of Go, and the last one concerns the routing problem in computer networks.

The first experiment represents the first training of a modified cellular simultaneous recurrent network (CSRN) trained with cellular particle swarm optimization (PSO). Another contribution is the development of a comprehensive theoretical study of a 2x2 Go research platform with a certified 5 dan Go expert. The proposed architecture successfully trains a 2x2 game tree. The contribution of the second experiment is the development of a computational intelligence algorithm called *collective cooperative learning* (CCL). CCL learns the group size of Go stones on a Go board with zero knowledge by communicating only with the immediate neighbors. An analysis determines the lower bound of a design parameter that guarantees a solution. The contribution of the third experiment is the proposal of a unified system architecture for a Go robot. A prototype Go robot is implemented for the first time in the literature. The last experiment tackles a disruption-tolerant routing problem for a network suffering from link disruption. This experiment represents the first time that the disruption-tolerant routing problem has been formulated with a Markov Decision Process. In addition, the packet delivery rate has been improved under a range of link disruption levels via a reinforcement learning approach.

ACKNOWLEDGMENTS

First, I would like to thank my family and Dr. Donald C. Wunsch II for their patience while I completed my doctorate program. I would have been unable to complete my program without their support and care. I am also thankful to Mrs. Wunsch for her kindness. The financial support from the Missouri S&T Intelligent Systems Center, the Mary K. Finley Missouri Endowment, the National Science Foundation (contract number ECCS-0725382), the DARPA Disruption Tolerant Networking Program (contract number W15P7T-05-C-P211 through a subcontract to BBN Technologies), and EFRI grant #0836017 is gratefully acknowledged.

Additionally, I am grateful to Dr. Kelvin T. Erickson (the department chair) for allowing me extensive teaching opportunities while earning my degree. I would like to thank my committee members (listed in alphabetic order) for investing their time and efforts: Dr. Jagannathan Sarangapani, Dr. Sanjay Madria, Dr. Steve Grant, and Dr. Yahong Rosa Zheng.

I also express my gratitude to the following faculty members for teaching me both in and out of the classroom: Dr. Levent Acar, Dr. Minsu Choi, Dr. Chang-Soo Kim, Dr. Sahra Sedighsarvestani, Dr. Bijaya Shrestha, Dr. Ganesh Kumar Venayagamoorthy, Dr. Chengshan Xiao, Dr. Maciej Zawodniok, and Dr. Larry D. Pyeatt (from Texas Tech University).

I also thank the following people for their friendship and support: Sangjun Lee, Yonseok Cho, Dr. Young Min Yoo, Dr. Jayong Koo, Dr. Hoyong Kim, Dr. Gichun Kang, Sejun Kim, the Missouri S&T Baduk/Go/Weiqi club members, Dr. John E. Seiffertt III, Dr. Xindi Cai, Dr. Rui Xu, Dr. Wenxin Liu, Duksoo Lim, and J. Adam Nisbett.

TABLE OF CONTENTS

Page

PUBLICATION DISSERTATION OPTION iii

ABSTRACT iv

ACKNOWLEDGMENTS v

LIST OF ILLUSTRATIONS x

LIST OF TABLES xii

LIST OF ALGORITHMS xiii

SECTION

1. INTRODUCTION 1

1.1. OBJECTIVE 1

1.2. GAME OF GO AND COMPUTER GO RESEARCH 1

1.2.1. Game of Go 1

1.2.2. Game Rules and Norms 2

1.2.3. The Grading System to Measure the Computer Go Program's Strength 4

1.2.4. Why Research Computer Go? 8

1.2.5. Why is Go Difficult for a Computer? 9

1.3. HOW A COMPUTER CRACKS GO 11

1.3.1. Computer Go Programs: Overview 11

1.3.1. Popular Approaches Used for Go Engines 13

1.3.2.1. The search-based approaches and advances in the reinforcement learning-based approach 14

1.3.2.2. The non-search-based approaches 22

1.4. PARTICLE SWARM OPTIMIZATION BACKGROUND 27

1.5. DISSERTATION PROBLEM STATEMENT 29

1.6. TECHNICAL APPROACHES AND CONTRIBUTIONS 31

1.6.1. Technical Approaches 31

1.6.2. Contributions 32

1.7. ORGANIZATION 33

2. MODIFIED CELLULAR SIMULTANEOUS RECURRENT NETWORKS WITH CELLULAR PARTICLE SWARM OPTIMIZATION 34

2.1. INTRODUCTION 34

2.2. GAME OF GO 36

2.2.1. The 2x2 Game Rules 36

2.2.2. 2x2 Go Game Tree 37

2.3. CELLULAR SIMULTANEOUS RECURRENT NETWORK 40

2.3.1. Conventional Cellular Simultaneous Recurrent Network 40

2.3.2. Modified Cellular Simultaneous Recurrent Network 43

2.4. CELLULAR PARTICLE SWARM OPTIMIZATION 45

2.5. SIMULATION CONFIGURATION, RESULTS AND DISCUSSIONS 46

2.5.1. Configuration 46

2.5.2. Performance Metric 48

2.5.3. Selected Result 48

2.6. CONCLUSIONS 50

2.7. FUTURE RESEARCH 51

PAPER

1. RECURSIVE AND NON-RECURSIVE ALGORITHMS FOR THE GROUP SIZE COUNTING PROBLEM IN COMPUTER GO 52

ABSTRACT 52

1.1. INTRODUCTION 52

1.2. GROUP SIZE COUNTING PROBLEM IN COMPUTER GO 54

1.3. THE PROPOSED RECURSIVE AND NON-RECURSIVE ALGORITHMS 56

1.3.1. The Recursive Approach 56

1.3.2. The Proposed Non-Recursive Approach with Cellular Structure 58

1.4. SIMULATIONS AND DISCUSSIONS 62

1.5. CONCLUSIONS 64

1.6. REFERENCES 65

2. ROBOTIC GO: EXPLORING A DIFFERENT PERSPECTIVE ON HUMAN-COMPUTER INTERACTION WITH THE GAME OF GO 67

ABSTRACT 67

2.1. INTRODUCTION 67

2.2. GAME OF GO AND GO SOFTWARE 69

2.2.1. Game of Go: Basic Game Rules 69

2.2.2. Categories of Go Software 71

2.3. BIG PICTURE: PLAYER CONFIGURATIONS AND THE INTERFACE ISSUES 73

2.4. PROPOSED ARCHITECTURE FOR HUMAN-MACHINE -COMPUTER INTERFACE 76

2.4.1. Dilemma of a Contemporary Player: Traditional Board and Stones or Online Go Server? 76

2.4.2. Playing on a Go Server Inevitably Decays Amateur Club Houses 78

2.4.3. Proposed Architecture to Solve a Go Player's Dilemma 78

2.5. IMPLEMENTATION OF A GO ROBOT: CHEONSOO-I 79

2.5.1. Vision System and Image Processing 79

2.5.2. Stone Manipulation with Robotic Arm 82

2.6. CONCLUSIONS 84

2.7. REFERENCES 85

3. RECONFIGURABLE DISRUPTION TOLERANT ROUTING VIA REINFORCEMENT LEARNING 87

ABSTRACT 87

3.1. INTRODUCTION 87

3.2. MARKOV DECISION PROCESS AND REINFORCEMENT LEARNING 90

3.3. PROPOSED MARKOV DECISION PROCESS FORMULATION AND REINFORCEMENT LEARNING MODEL 92

3.3.1. System Architecture of the RL Model 92

3.3.2. Markov Decision Process Formulation 93

3.4. SIMULATION ENVIRONMENT 96

3.5. SIMULATION RESULTS 98

3.6. CONCLUSIONS 101

3.7. REFERENCES 103

SECTION

3. CONCLUSIONS 105

APPENDICES

A. GO GAME RULES 106

B. 2X2 GO GAME TREE 112

BIBLIOGRAPHY 131

VITA 144

LIST OF ILLUSTRATIONS

Page

Figure 1.1. Grading systems in Go 6

Figure 1.2. State-space and game tree complexities of board games 10

Figure 1.3. Tree search methods in depth and breadth dimensions 23

Figure 1.4. Various optimization methods 27

Figure 1.5. Four paradigms of CI and sub-domains of those paradigms according to Engelbrecht′s classification of CI 32

Figure 2.1. Sample 2x2 board status and graph notation 37

Figure 2.2. A game tree of a 2x2 board for a starting move on the left bottom play point 38

Figure 2.3. Selected game trees for optimal moves in territory rules 39

Figure 2.4. The CSRN structure 41

Figure 2.5. The big picture of the learning process over time 42

Figure 2.6. The proposed modified CSRN structure 43

Figure 2.7. Learning curves for the 2x2 CSRN 49

PAPER 1

Figure 1.1. Group size counting problem example 55

Figure 1.2. An example of the data structure for the proposed recursive scheme 57

Figure 1.3. System architecture and macroscopic view of its operation 59

Figure 1.4. An example of a 19x19 Go game 63

Figure 1.5. Board size vs. the excess waiting time ε in linear and log scale 64

PAPER 2

Figure 2.1. Final at the 27th KBS Baduk-wangJeon (Battle of the Baduk King) 70

Figure 2.2. Artificially placed stones to explain adjacency (top) and liberty (bottom) concepts 70

Figure 2.3. Go client program 72

Figure 2.4. Block diagram of unified structure for human–computer-network interface 73

Figure 2.5. Player configurations and the corresponding interface issues 74

Figure 2.6. Stone placement 77

Figure 2.7. The proposed Human-Machine-Computer-Network Interface architecture 78

Figure 2.8. Player configurations and interface issues 80

Figure 2.9. Cheonsoo-I 81

Figure 2.10. Example of an image processing sequence 82

Figure 2.11. Gripper tips for Cheonsoo-I 84

PAPER 3

Figure 3.1. Agent-environment interaction in an RL model 91

Figure 3.2. Simplified block diagrams of our RL model 92

Figure 3.3. An example of the discrete state space *S* of the knowledge bases 94

Figure 3.4. 5-by-5 mesh topology with wired links 95

Figure 3.5. Comparison of average packet delivery rates in a 5-by-5 mesh topology: Local learning, Link state and Gossip 99

Figure 3.6. Relative performance improvement by on-node local learning 101

LIST OF TABLES

Page

Table 2.1. Simulation configurations for the CSRN 47

Table 2.2. Configuration for cellular PSO 47

PAPER 2

Table 2.1. Summary of play configuration and interfaces 76

Table 2.2. GIS's counterpart block for different opponents 76

PAPER 3

Table 3.1. Simulation parameters 96

LIST OF ALGORITHMS

Page

PAPER 1

Algorithm 1.1. The proposed recursive algorithm..... 58

1. INTRODUCTION

1.1. OBJECTIVE

The main theme of this dissertation is spatial-temporal reasoning applications in computational intelligence (CI). Spatial-temporal reasoning is a process of drawing conclusions from facts or spatial images over time. CI is a field of study that uses intelligent algorithms, such as artificial neural networks (ANN), evolutionary computing (EC), swarm intelligence (SI), and fuzzy systems (FS) [1]. In this dissertation, spatial-temporal reasoning concepts are combined with various CI techniques. In the process, research was conducted regarding how to draw conclusions over time for problems in computer Go, robotics and computer networks that require a mental perception of the problem space.

1.2. GAME OF GO AND COMPUTER GO RESEARCH

1.2.1. Game of Go. Go, also called Baduk in Korea and Weiqi in China, is a two-player board game that is popular in East Asia and is gaining more popularity in other regions. Chess could be considered its Western counterpart. Go is as popular, or perhaps more popular, in East Asia as Chess is in the Western world. Professional players associations exist in South Korea, China, Japan and Taiwan as evidence of the game's popularity. A Weiqi tournament was even held at the 2010 Asian Games in Guanzhou, China.

The basic Go rules are so simple that a complete beginner can learn to play within an hour. One Go player possesses black stones; the other controls white stones. The two players alternately place a stone on the Go board. A typical Go board is a 19-by-19

square lattice, which represents an imaginary land under territory dispute. A player's objective is to obtain more territory on the board according to the Go rules [2]. Interested readers may refer to Appendix A for the game rules in further detail.

Unlike learning the basic rules, mastering the game is another story. Life time is required to master the art of playing Go. Go is so profound that experts believe no identical game has been played since the game was invented. In general, about a decade of intense and dedicated effort is a prerequisite to pass the professional test.

1.2.2. Game Rules and Norms. There are three major sets of Go rules (Chinese, Japanese and Korean), and some professional tournaments make minor modifications to the major rules after careful review. The three major rule sets fall into two categories: area rules and territory rules. Chinese rules follow the area rules, while Japanese and Korean rules are territory rules. These rules differ in how they dictate that the final score be counted, so they affect game play. On the other hand, the essential rules for both categories are identical and straightforward.

Rule 1 (Go board): A Go board is an *n-by-n* grid on whose intersections stones are placed. Currently, the norm for a typical game is a 19x19 board, but smaller board sizes of 5x5, 6x6, 9x9 and 13x13 can be used for educational and research purposes.

Rule 2 (Players): Two players, black and white, alternately place a stone on the board. Who places the first stone is determined by the type of game.

Rule 3 (Purpose of the game): Go is a territory game, so, according to territory rules, the purpose of the game is to gain more territories on the board than the opponent. A territory is an empty intersection seized by a player. According to area rules, on the

other hand, a player with more areas wins the game. An area is a territory with stones surrounding it.

Rule 4 (Types of games: even and handicapped): Black plays first in an even game, and white in a handicapped game. When both players, whether amateur or professional, are equally strong, they play an even game that starts from an empty board. When amateurs at different levels play a game, handicap stones are placed on the board before the game begins to allow two players with different ranks to play a competitive game.

Rule 5 (Compensation for a white player): A black player gains a significant advantage by playing the first move in a competitive game. In order to compensate for black's advantage, a white player receives a score compensation of 6.5, 7.5, or 8, depending on the rules. Though their accuracy remains under debate, these values, except for 8, are agreed-upon scores resulting from many game plays by professionals.

Rule 6 (Liberty, group and capture of stones and an eye): A group of stones refers to one or more stones of the same color that are connected vertically or in parallel. A group of stones is captured when the last liberty for that group is removed. A liberty is an empty, adjacent intersection of a stone connected by lines drawn on the board. Typically, a group of stones is captured when it is completely surrounded by the opponent's stones.

Rule 7 (Illegal moves: suicide and Ko rules): A suicide move is prohibited. Therefore, placing a stone in an eye surrounded by the opponent's stones is illegal unless this placement captures the opponent's group. Another illegal move is one that repeats the board's status at the player's previous move. This rule is called a Ko rule and was established to avoid infinite game play. In order to avoid the infinite loop of repeatedly

capturing the opponent's stone, each player must wait one move to capture an opponent's stone in the disputed territory.

1.2.3. The Grading System to Measure the Computer Go Program's Strength. A human player's skill level is indicated by his/her ranking within the player's community, as well as by a grade [3,4,5]. A professional player's ranking is his current score in competitions, while the grade reflects the player's history. In general, a computer Go engine's strength is measured with respect to other Go engines. Say a Go engine wins more games against the other; the former is considered to be stronger. Occasionally, matches between a computer and a human player are held to measure the strength of the top Go engines. As a result, the Go engine's strength can be assessed with respect to a human's grade system.

Several professional Go associations exist in East Asia, including the Korea Baduk Association[6], Chinese Weiqi Association [7], Nihon Ki-in (Japan) [8], Kansai Ki-in (Japan) [9], and Taiwan Chi Yuan [10]. Two professional associations exist in the Western world, American Professional Go Association and European Go Professionals [11], but the former is considered to be defunct, and the latter is less influential within the professional societies compared to the associations in East Asia.

As of July 2011, nine of the top ten nationally-ranked players in Korea are professional 9 Dan; one player (Jiseok Kim) is professional 7 Dan. Most top-ranked professional players become 9 Dan as the result of career achievements. A player is promoted to the next grade when he/she accumulates enough points during tournaments. Therefore, Kim will be promoted to 9 Dan eventually if he keeps playing well. There are 55 professional Go players at the professional 9 Dan (top grade) level among the 254

professionals in the Korea Baduk Association [6]. It is possible for a player at the top rank with a lower grade to defeat a player at a lower rank with a higher grade. Consequently, a time will come when computer Go programs will be stronger than they are now, and a player's ranking will need to be considered when discussing a computer Go's strength.

A huge gap in strength exists between professional and advanced amateur players in general. For a professional player, the ranking is more important than the grade as a measure of the player's current strength. This is especially true for international competitions because grading system standards vary among nations. In general, the national associations have a trainee system to prepare the pro exam and the degree of competition differs from nation to nation.

Typically, only the general structure of a grading system is explained in the computer Go literature (as in Figure 1.1.a). The grading system is largely divided into the advanced level and the remaining levels similar to those found in the martial arts, such as Taekwondo, Karate and Judo. An advanced player is granted Dan analogous to a black belt in martial arts; the grades for a weaker player are called Kyu in Japanese, Geup in Korean, and Ji in Chinese. Dan is divided further into a professional Dan and an amateur Dan if a large pool of professionals exists in the community. Kyu (Geup or Ji), amateur Dan, and professional Dan (when these distinctions exist) are abbreviated as k, d, and p in this dissertation, respectively. For example, professional 9 Dan is abbreviated to 9p.

However, the aforementioned grading system does not sufficiently capture the current stance in the Go world. No single grading system is internationally standardized. Nihon Ki-in in Japan set the contemporary grading system, and other organizations

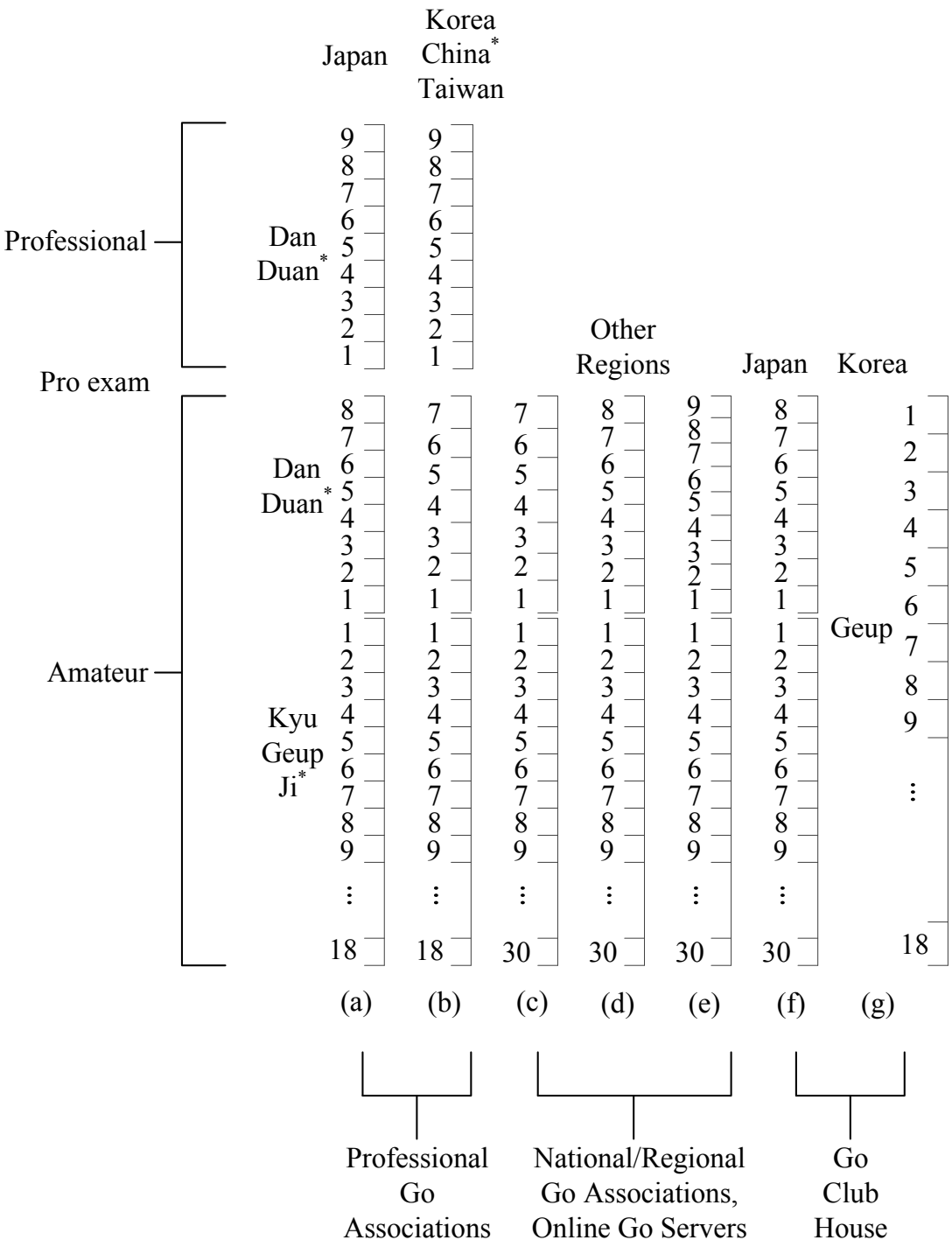


Figure 1.1. Grading systems in Go.
(a) certified grading system of a national association,
(b) uncertified grading system of an online Go server,
(c) Kiwon Geup-soo or Go club house grading system.

referred to it in general when devising their own systems, which remain diversified. Currently, the grading systems differ in professional associations, national/regional associations, online Go servers, and club houses. To name a few, the American Go Association [12], Canadian Go Association [13], Singapore Weiqi Association [14], and European Go Federation [15] are examples of national/regional associations. Examples of famous online Go servers include TYGEM [16], Cyberoro [17], HANGAME [18], KGS [19], and IGS [20]. Even given the existence of the International Go Federation [21], attempts at standardization have failed because this federation is less influential to the professional societies than the major professional associations. Traditions and localisms have not yet been broken to reinvent a single grading system.

Figure 1.1 summarizes the grading systems currently used in Go. The grading systems are categorized into three groups: professional Go associations, national/regional Go associations and online Go servers, and Go club houses. All players in professional Go associations certify as professional 1 Dan to 9 Dan and an honorary 10 Dan (not depicted in the figure). Japan's Nihon Ki-in and Kansai Ki-in certify amateur Dans up to 8 Dan, unlike other East Asian countries. The grading systems for Kyu level players are identical in all East Asian nations.

Other countries do not certify professional players even if professionals belong to the association. Only amateur Dan and Kyu level players are certified. Note that players from these associations start from 30 Kyu as opposed to 18 Kyu in professional Go associations. Online Go servers employ the same structure even if the criterion to determine a grade may differ. The top grade for an amateur Dan may vary from 7 to 9 depending on the association.

The last category is that of a Go club house, called Kiwon in Korean and Ki-in in Japanese, which serves as a focal point for local Go communities. In South Korea, the traditional grading system of using only Geup remains in place. Regardless of the efforts by the Korea Baduk Association, the newer amateur Dan system has not been implemented fully. The contemporary Japanese club house's grading system with Dan levels is depicted for comparison purposes.

1.2.4. Why Research Computer Go? Computer Go [22,23,24,25] is a field of study that aims to develop computer programs associated with the game of Go and developing a strong program that can defeat human champion is an unconquered challenge in the Artificial Intelligence and Computational Intelligence societies [26,27].

The history of computer Go research traces back to computer Chess, regarding which the research objective was to construct a chess-playing computer program. Wiener's book published in 1948 [28] describes how a computer Chess program could be developed with a depth-limited minimax tree search [29] with an evaluation function. Wiener's method is conceptually similar to how a human plays Chess. A computer program can play Chess if it can foresee the future possible moves and properly evaluate the goodness of each move. The minimax tree search searches for moves that maximize the player's chance to win while minimizing the opponent's chance to win. A tree of board statuses called a *game tree* is constructed in this process, and each node of the tree is evaluated by an evaluation function. In 1950, Shannon published a paper [30] dedicated to a computer Chess program with a tree search and evaluation function, and Turing discussed the first program capable of playing a full game in 1951 [31,32]. Within

half a century, the IBM supercomputer Deep Blue [33] defeated the human Chess champion Garry Kasparov in May 1997.

The primary focus of computer Go research is to develop an intelligent computer program called a *Go engine* that generates a sequence of moves to play the game. The first Go engine capable of playing a full Go game against a human was published by Zobrist in 1970 [34]. Computer Go's technical significance was rediscovered in the quest for a more complicated problem than Chess after Deep Blue's breakthrough in computer Chess. Currently, computer Go is considered one of the most challenging unsolved problems in the Artificial Intelligence and Computational Intelligence societies. Unlike Chess, the best Go engine cannot beat a human champion even with the fastest computer to date, making computer Go an excellent test bed for computer intelligence. Additionally, it serves as an excellent test bed for novel applications to practical problems because of its clear-cut objective and the vast amount of sub-problems that allow for tests of practicality.

1.2.5. Why is Go Difficult for a Computer? Go is more difficult than Chess because its tree searches are far more complex, and its game tree size is much larger. In addition, developing an accurate evaluation function and ending the game is difficult in Go. In other words, these three challenges complicate move generation in Go.

First of all, constructing a game tree in Go is more challenging than in Chess because of Go's high branching factor. Go has a much larger number of possible moves in each position, and the length of a game is longer than Chess [35]. The game tree size of a 19x19 Go board is estimated to be between 10^{575} to 10^{620}, while that of a Chess board is 10^{123} [36].

Figure 1.2 compares the complexity of Go to other board games. (Refer to [37] for more board games.) Using an analogy, the complexity of Go compared to Chess is greater than that of Chess to Tic-tac-toe. Secondly, the evaluation function in Go is more difficult to build than that in Chess. Unlike Chess pieces to which various values can be assigned [26], all Go stones are identical, and a stone's value should be evaluated in the context of the geometrical and strategic importance of its location. A single stone at one location can be more valuable than one at another location depending on the other stones on the board. Creating an accurate evaluation function is difficult [35] because scoring a stone is challenging.

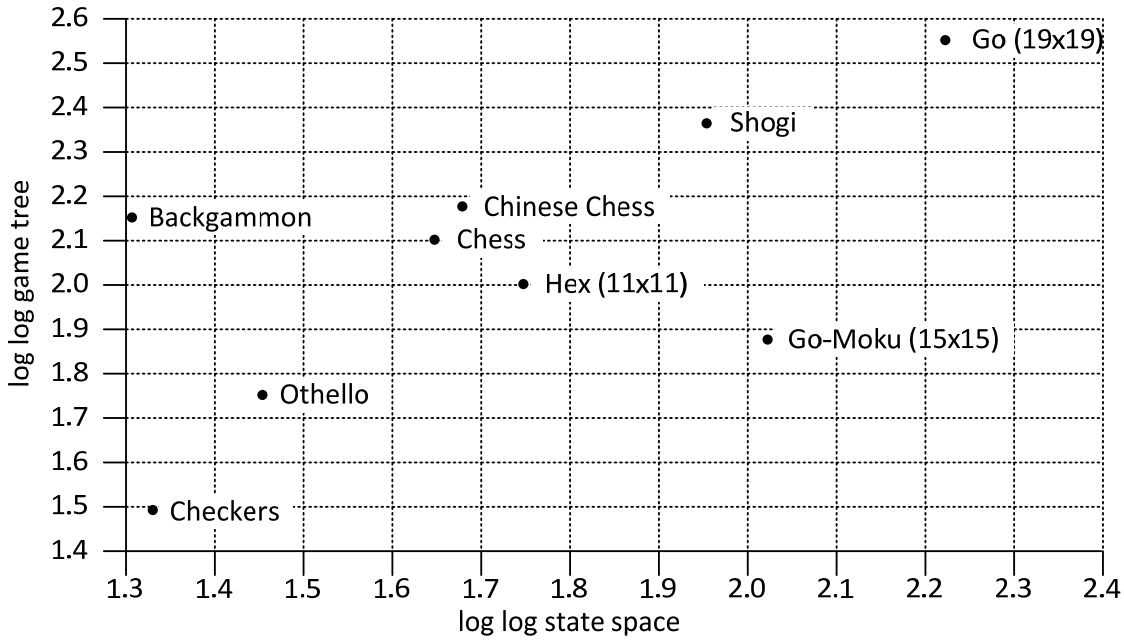


Figure 1.2. State-space and game tree complexities of board games.

Lastly, the endgame in Go is somewhat ambiguous because of three problems specific to Go [35]. Most importantly, judging whether a position is terminal is difficult, unlike in most board games [35,24,38]. Judging the terminal state is easy in Chess; the game is over when the king is taken (checkma te) or when any legal move would place the king in check (stalemate). In Go, two players must agree when to finish the game unless a player forfeits. The game also ends if both players make two consecutive passes. Computer Go uses the latter method because the former is difficult for computers to negotiate. Therefore, a pass move should be added to the tree search. When a player passes on a move that may end a game, the entire board status must be assessed for the correct decision. Correctly judging the terminal position is challenging because board status assessment is non-trivial. Even though passes are allowed, passing a move should be considered only toward the very end of the game, in general, because a pass may not be beneficial during the game. This is especially true for a close game. Additionally, the *Ko* rule that disallows a repeated board status further complicates the pass judgment. A full board position repetition is illegal in Go, but local position repetition may occur [35].

1.3. HOW A COMPUTER CRACKS GO

1.3.1. Computer Go Programs: Overview. The term *computer Go program* is polysemous. It could signify a program that has the intelligence to generate a move (Go engine, or engine for short), a program that displays the Go board status (Go GUI), a program that allows a human to connect to an online server and play online (Go client, or client for short), or a program that helps humans learn to play Go (Go Learning Tool). This research is interested in the first two types. The purpose of a Go engine is to

generate a computer's next move. A Go engine is also referred to as computer Go intelligence or a Go-playing program. Several famous Go engines include The Many Faces of Go, MoGo, GNU Go, Crazy Stone, Silver Star, and Fuego. Go GUI is the frontend to a human user in that the GUI (Graphic User Interface) displays the Go board and stone placements. Popular open-source Go GUIs include CGoban [39] and GoGui [40]. Go GUI is also called a Go editing program to emphasize that users can store and edit game records with it. Most Go clients incorporate Go GUI. Therefore, a well-designed Go client also provides the editing feature. For example, the Go GUI software CGoban serves as a Go client for a popular Kiseido Go Server (KGS) [19] and also as editing software.

A *Go communication protocol* is a program that allows Go programs to communicate with each other. The Go Modem Protocol (GMP) [41] and the newer Go Text Protocol (GTP) [42] represent the *de facto* standard for open source Go programs. The former was developed by Bruce Wilcox with input from other Go programmers to be used for computer Go competitions. The latter is a modern alternative to GMP. A common usage of a Go protocol is to pair up Go engines in a computer Go competition. Two Go engines with identical Go protocols can be paired up by a matching program. Another popular usage is to connect a Go GUI to a Go engine. While a Go engine can be combined with a Go GUI (along with a Go client), they typically are separate programs. For example, GoGui, which is written in Java, incorporates GTP. It can be linked to any Go engine with GTP written in any programming language. Fuego in C++ implements GTP, so a human can play against Fuego on GoGui.

1.3.2. Popular Approaches Used for Go Engines. Both Go and Chess are perfect information two-player board games. That is, no information is hidden from the players, and no forced random factor exists, such as dice rolling in Backgammon. According to Claude Shannon, a perfect information two-player board game can be solved if the game tree and the evaluation function are available, meaning that a player can choose the best moves if all the possible moves and their values are known. Shannon's prediction worked for Chess because Deep Blue, the computer Chess program, defeated a human champion, Garry Kasparov, using, in essence, a game tree search combined with board evaluation. For Go, both a game tree search and board evaluation are extremely challenging to perform because Go has a large branching factor, and the values of Go stones are hard to evaluate. Therefore, the methods successful for Chess have not been successful for Go.

To proceed in computer Go, new methods had to be developed. In general, a well-designed, contemporary computer Go program follows a multitude of design philosophies. In other words, contemporary computer Go researchers employ hybrid methods combining different methods and algorithms [27]. Most Go engines combine several approaches; using a single approach is rare.

Traditionally, the design philosophies are categorized into (1) influence function, (2) tree search, (3) pattern matching, (4) knowledge-based systems, (5) Monte-Carlo (MC) methods, and (6) machine learning. It is the computer Go researcher's decision to select which philosophies to use in order to create a stronger program. For example, Crazy Stone, one of the strongest Go engines, combines the MC evaluation technique, tree search, and a pattern learning algorithm[43].

In this dissertation, we categorize the conventional design philosophies into search-based approaches and non-search-based approaches. The former includes (2) tree search, (5) MC methods, and a part of (6) machine learning approaches, such as reinforcement learning and temporal difference learning. The latter includes (1) influence function, (3) pattern matching, (4) knowledge-based systems, and the remainder of (6) machine learning approaches. What distinguishes the former from the latter is the search mechanism, or an algorithm to explore the game tree.

1.3.2.1. The search-based approaches and advances in the reinforcement learning-based approach. A tree search is an algorithm used to search a game tree, which is a hierarchical tree structure with linked nodes that correspond to a snapshot of a Go board during a game. Mathematically, the tree is ordered and directed because the root is an empty Go board and an ordering is specified for the children of each vertex. Constructing the full game tree is impractical because of Go's large branching factor. Therefore, the game tree mentioned here is an abstraction in the mind to organize the progress of Go games.

Finding winning rather than optimal moves against the opponent is the objective of a game because truly optimal moves may not be known to human experts. In this context, the objective of a tree search is to find winning moves against the strongest opponent in existence. Humans' limited knowledge of truly optimal moves is explained through the discussion of a professional's game by other professionals. The selection of Fuseki and Joseki, i.e., the well-established theories for opening moves, is debated frequently. Often, the soundness of an opening move selected by a professional player is analyzed by other professionals. This fact implies that the optimal moves are not

understood fully even by professionals. The situation is even more ambiguous for the middle and end of a game because the theories regarding these portions are not as well-established as the opening.

Assuming optimal moves are unavailable, the question is how to find winning moves via a tree search. Inspired by success in Chess, exhaustive searches with limited depth have been applied to Go. Computer Chess uses minimax algorithms with alpha-beta pruning [24]. The minimax algorithm selects nodes that minimize the opponent's chance to win but maximize the player's chance to win. Alpha-beta pruning decreases the number of nodes evaluated by the minimax algorithm. Even after the exponential increase in computational power provided by Moore's law since 1997, the brute-force search successful for Chess has not brought success to Go thus far. It is a common belief in the computer Go society that an exhaustive game tree search using a pruning technique does not work for Go because Go has a higher branching factor than Chess, and an accurate node evaluation is unavailable. However, Feng-Hsiung Hsu, the principal designer of the IBM Deep Blue, predicted in [26] that the exhaustive search will play a pivotal role in computer Go.

In computer Go, three types of tree searches are commonly used: a full-board search, a single-goal localized search and a multiple-goal search [23]. A full-board search is challenging, as explained in Section 1.2.4. A single-goal localized search is used widely in current Go programs to achieve particular tactical goals, such as ladder [44], single group capture [44], life and death [45,46], and Semeai [47]. A multiple-goal search is used to achieve a combination of two or more goals. Even if this approach is more

desirable than a single-goal-based search, its implementation remains rudimentary because of its complexity [23].

The Monte Carlo (MC) approach [48,49,50] has brought the most success to computer Go. The advantage of the MC approach over the conventional tree search methods is that MC is a value-based reinforcement learning (RL) method [51,48]. Unlike the conventional tree search that requires a separate evaluation function, this RL approach plays many random games to compute the value of a node. More experience is gained as more games are played, leading to more accurate results. Ideally, an accurate approximation of a node's value can be computed if enough games are played. Conversely, insufficient number of sample games, e.g. random games played till the end, leads to false values of the candidate nodes.

The first MC tree search Go program, Gobble, was developed by Bruegmann and appeared in 1993 [52]. The architecture of Gobble is simple, but its drawback is that it does not make living groups or finish games properly [49]. Domain-dependent and OLEG also use domain-specific knowledge to define eyes to random games and also knowledge was added to define eyes[1] by Fotland [53] and Chen and Chen [54]. OLGA apply techniques such as progressive pruning, all moves initially being heuristic, the concept of temperature, simulated annealing and depth-two tree search within the MC framework [49,55].

A recent breakthrough by Go engines was made using a variant of the MC method, i.e., the Upper Confidence bound applied to Trees (UCT). The UCT was first

---

[1] An eye is an empty space of intersection where the opponent can neither play on nor force it to be filled [149] .

applied to Go by Kocsis and Szepesvari [56]. Theoretically, the finite-horizon discounted Markov Decision Process (MDP) [57] allows for the attainment of near-optimal solutions. Formulated by MDP, a multi-arm bandit approach [48] allows the program to look ahead at future moves. The Upper Confidence Bound (UCB) [58] is a type of bandit algorithm [56], and the UCT applies UBC1 to the tree search. The first successful implementation of the UCT was in MoGo, developed by Wang and Gelly for [59,60,51,61,62]. MoGo is considered one of the strongest computer Go programs to date. The executable version is available at [62] and the pseudocodes of the UCT approach are published in. However the details of the program are unknown because the source codes are not available to the public. Gelly's publications [59,60,51,61] discuss how the UCT is modified for Go[61] and improve the UCT with patterns and offline knowledge [51,60,59]. Chaslo et. al. also combined expert, offline, transient and online knowledge into an MC tree search [63].

MoGo has achieved master-level play on a 9x9 game board [64] and has won computer Go tournaments, such as KGS international tournaments [65] and the World 9x9 Computer Go Championship held at the National University of Tainan (NUTN) in Taiwan [66]. In [66], it is described that MoGo has reached the level of 3 Dan. Even if this description is correct, MoGo's strength remains controversial given the level differences in different countries, as presented in Section 1.2.3. and Figure 1.1. On the other hand, the computer Go community was excited about MoGo's achievement during the US Go Congress [67] organized by the American Go Association [12] in 2008. The MoGo Titan program using the Dutch supercomputer Hyugens played a 9-stone handicap game on a 19x19 board against Myungwan Kim, an 8 Dan professional from Hangook

Kiwon [6], on August 7th, 2008 [68]. This official match resulted in MoGo's triumph by 1.5 points. Interested readers may investigate the game record available at [69].

Even though this event is considered a breakthrough by the computer Go community, the correct assessment of MoGo's strength must be explored. Surprisingly, the computer Go community was not as impressed by this victory, claiming that MoGo did not reach a Dan level because a 9-stone handicap against a 9 Dan player is assessed as 1 Kyu. Considering that the opponent was a professional player, MoGo's win is encouraging, and other official matches have substantiated MoGo's strength against other professional players. On February 10-13 at the Taiwan Open 2009, MoGo TITAN ran on Huygens in Amsterdam, Netherlands, and won two 19x19 games, one with a 6-stone handicap against Li-Chen Chien, a 1 Dan professional Go player from Taiwan Qiyuan [10], and the other with a 7-stone handicap against Jun-Xun Zhou, a 9 Dan professional from Taiwan Qiyuan [70]. The latter event is impressive even with the 7-stone handicap because Zhou won an international title LG Cup in 2007.

The MC approach has become trendy owing to MoGo's success. Other engines, such as Orego [71], Crazy Stone [43], The Many Faces of Go [72], Fuego [73], Indigo [74,75], and Phantom Go [76,77] use this approach. An MC method that recursively extends nodes and plays effective moves has been implemented and has improved the performance of Crazy Stone [78].

The temporal difference (TD) learning method, which has been applied to various board games [37] such as Tic-Tac-Toe [79], Chess, Othello, and Backgammon, also has been applied to Go. This method proved most successful in Backgammon; TD-Gammon [79], developed in 1992 by Gerald Tesauro at IBM, uses a variant of TD learning called

TD-lambda [48,37]. The drawback is that table-based reinforcement learning suffers from the curse of dimensionality [48]. Backgammon's state space is about 10^{20} states [80], which is too large for table-based reinforcement learning. To tackle the curse of dimensionality, Tesauro used a backpropagation-based, three-layer neural network and implemented the outcome from a self-play game as the reinforcement signal [80,79]. As a result, TD-Gammon reached the level of human experts [48,37,79].

In [81], Schraudolph et. al. experimented with Tesauro's approach to apply TD learning to Go, but a fully-connected 82-40-1 backpropagation network by self-play on a 9x9 Go board was not very successful. Next, they tried using a network with less than 500 weights and TD learning so that Go positions on a 9x9 board could be evaluated. Three computer opponents were used to train the network: a random move generator, Wally, and The Many Faces of Go. Schraudolph et. al. tested training the architecture by self-play and playing against Wally, concluding that the latter method outperforms the former when the engine is tested against both Wally and The Many Faces of Go. First, they trained with the weaker Wally for about 2000 games and then switched the training partner to the much stronger The Many Faces of Go for another 1000 games. They achieved some success against The Many Faces of Go set to a medium level.

Another Go engine using neural networks trained with TD learning is NeuroGo [82], developed by Markus Enzenberger at the University of Alberta. NeuroGo uses a feedforward backpropagation architecture trained with TD learning and self-play [83,84]. In this architecture, the number of neurons and connections changes depending on both the board's size and status [85]. NeuroGo achieves a moderate level of strength compared to other famous Go engines, but it actually is a crossover program. It leans heavily

towards the soft AI technique of machine learning but also incorporates expert knowledge, which ultimately must come from human experience.

Other research has investigated using neural networks and TD learning. Chan et. al. also trained a feedforward neural network with TD-lambda from self-play [86]. They tested various lambda values and found that the non-zero value outperforms TD(0). However, this network has not been tested against an existing Go engine, so its strength is unclear. Additionally, Zaman and Wunsch developed a neural network trained with TD learning [80,87]. The board evaluation function was learned by three 2-layer feedforward neural network experts trained by a hybrid mixture of experts (HME) and Meta-Pi. This network can beat Wally on a 9x9 board. In [88], Zaman et. al. explored the possibility of using another type of neural network that can learn to play Go. A heuristic dynamic programming (HDP)-type adaptive critic design (ACD) was used to play games against Wally. The trained network did not perform strongly, but this actor-critic design learned to play Go from zero knowledge, showing that the principle of dynamic programming can be applied to Go. In [89], Wu and Baldi used another type of neural network with recursion, a type of Bayesian network called a directed acyclic graph recursive neural network (DAG-RNN) with six layers. They used game records on 9x9, 13x13, and 19x19 board settings. However, whether or not the trained networks learned to play Go well against existing Go engines remains unclear. Previous research with various neural networks has shown interesting results, and some networks have learned to play Go. Unfortunately, it is unclear whether or not neural networks alone can serve as a strong Go engine. However, as Ramon and Blockeel suggest in [90], CI techniques can be utilized

to learn aspects other than move generation, such as candidate move ordering, leaf node static evaluation, temperature or urgency of a position, and game opening techniques.

Mayer [91] studied the impact of three different board representations for neural networks and the optimality criteria for TD learning on a 5x5 board. Mayer used feedforward neural networks with TD learning, the Neural Go players learning from self-play, and a tournament of the players. The resulting Neural Go player won a computer Go program called Gojen. A new neural network architecture with TD learning also was used. Blair used a cellular neural network (CNN) trained with TD learning and self-play [92]. The CNN used an identical feedforward neural network in a cellular architecture trained with TD-lambda at λ=0.9. This network has been tested on an internet computer Go server, CGOS [93], and it achieved a rating of 500 on a 9x9 board and 1000 on 19x19 board. The network was parallelized on a GPGPU, NVIDIA GeForce 8800 graphics card to achieve a faster speed.

Runarsson and Lucas introduced another CI technique to the previous TD learning research. They compared two methods for acquiring position evaluation on a 5x5 board: coevolution learning (CEL) and a self-play feedforward neural network with TD learning (TDL) [85]. A neural network needed to be learned, for example, a single-layer perceptron with a linear output. Afterward, Krawiec and Szubert applied coevolutionary TD learning (CTDL) to a 5x5 board, thus combining a coevolutionary search and TD learning [94]. In this work, a board evaluation function was computed from a linear combination of the board's state and a weight matrix called a weighted piece counter (WPC). The weights were updated with CTDL. The strength of this architecture remains unclear because the opponent is a random player, a WPC, and the

performance is compared among the methods given in the paper, i.e. pure TD, pure coevolution, and CTDL.

Recent advances in computer Go can be explained by the reinforcement learning framework. In [48], Sutton and Barto explain the relationship among search methods in terms of the depth and breadth of the search. Figure 1.3 depicts the relationship between an exhaustive search, the MC approach, and temporal-difference learning as presented in [48]. All these algorithms find a node's value in the game tree but differ in the depth and width of their game tree search. For example, the exhaustive search covers both depth and breadth. If the search starts from depth, the method uses a depth-first search [95]; if all the child nodes are visited first, the method uses a breadth-first search [95]. On the other hand, the MC method searches the tree in depth with game samples. In other words, a sample of a random game is played until the end of the game. More samples are generated when more computational power is consumed, and the estimated value more closely approximates the true value when enough sample games are played.

1.3.2.2. The non-search-based approaches. Using an influence function is a legacy approach to generating a move. The first Go engine that played a full Go game against a human was reported by Zobrist in 1970 [34]. Zobrist used an *influence function* to evaluate a board's status as the sum of all the stones' influences on the board. The influence function is a way to quantify a stone's influence as it decreases with distance from the stone's location. The move with the highest influence is chosen for game play.

Pattern matching is an important topic in computer Go because patterns are a simple but effective way to incorporate a human expert's knowledge or offline knowledge into a Go engine [23]. If the entire Go board is seen as a graph, a

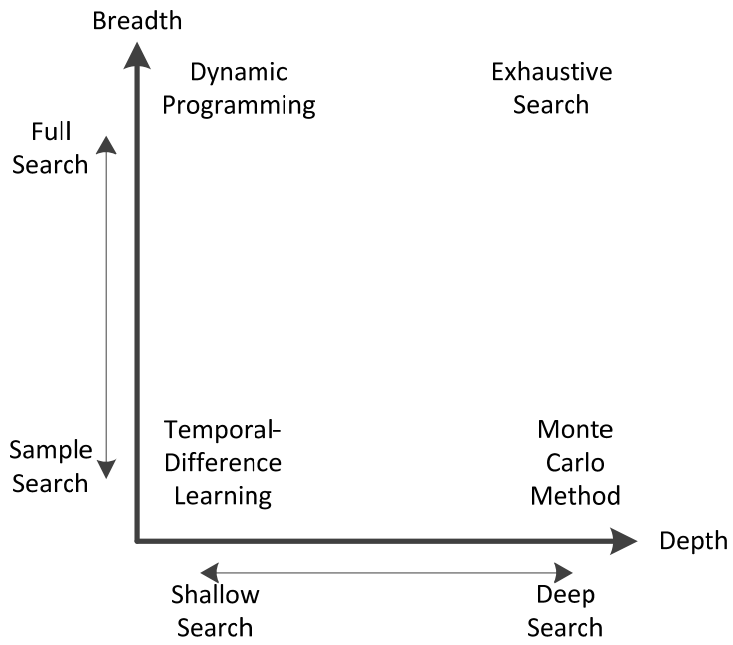


Figure 1.3. Tree search methods in depth and breadth dimensions.

pattern would be considered its sub-graph. Typically, the size of the sub-graph or pattern is much smaller than the full 19x19 board. A well-designed Go engine is equipped with a pattern database. In order to utilize the database, the patterns must be matched to the full board representation. Therefore, a pattern-matching subsystem is necessary. The advantage of using patterns in a Go engine is that human knowledge and experience can be incorporated into the computer program. On the other hand, the shortcoming to using patterns is the additional cost imposed for both the programmer and the computational resources. Constructing and maintaining an efficient pattern database consumes a programmer's time and requires knowledge about Go. Additionally, matching patterns to the Go board's status is computationally expensive.

Several approaches exist to mitigate the burdens of these drawbacks, some more useful than others. One approach is automatic pattern extractions. Several methods have been proposed to extract patterns automatically from readily available professional game records. In [96,97,98], Kojima used an evolutionary approach to build a system that learned to acquire the knowledge necessary to perform at an expert level. Patterns are obtained by statistically analyzing game records (Coulom) [99] or real-time game play (Sutton et. al.) [100].

Due to the grand scale of the computer Go problem, it is divided into many sub-problems [2]. Humans and computers understand endgames relatively well compared to game openings and mid-games. In [101], Berlekamp and Wolfe detailed the analysis of a late endgame from combinatorial game theory. The problems relating to their research are two-fold. The first problem arises out of an underlying assumption. The boundaries of all the blocks are clearly separated in their work, so the endgame is analyzed independently from one group to the other. In practice, the boundaries remain unclear until the very end of a game, leaving this approach impractical. Another problem is that their book is so esoteric that understanding it becomes difficult. In [102,103,104], Mueller et. al. introduce a method by which to analyze a Ko threat situation for endgames with thermography, i.e. a tool for analyzing combinatorial games. A thermograph graphically represents the value of a Go board's position at different temperatures. In this 2D graph, the horizontal axis represents the count, and the vertical line represents the temperature, which is a measure of the urgency of playing a move. In [105,106], Mueller further investigated Go endgames and relaxed the independent conditions of groups.

The life-death problem is one of the key problems for endgames, of which the complexity is PSPACE-hard [107]. The answer to the problem provides essential information for judging the life and death of the group [108,109]. In general, the life and death of a group significantly impacts the entire game. In [110], Benson introduces Benson's algorithm, which presents a graph-theoretic static analysis of the board arrangement for unconditional life. The notion of unconditional life for a group of stones is well-defined mathematically, and the conditions for unconditional life are mathematically established. In essence, this fine work is mathematically equivalent to a statement in Go, "Two true eyes are necessary for a group to survive unconditionally." In practice, Benson's algorithm is insufficient to perform well in an actual game, and it requires further research. In [111], Mueller addresses one of the important topics during an endgame via static rules and a local tree search, i.e., how to recognize secure territories in computer Go. Vila and Cazenave address more practical situations to determine the safety of a group when the group has large eye shapes or one eye [112]. In [113], Kishimoto and Mueller apply the depth-first proof-number (df-pn) search algorithm, which was applied to Othello in [114], and enhance the algorithm by comparing knowledge-based and tree-search-based approaches for the life-death problem in [115]. Additionally, Kishimoto and Mueller present a divide-and-conquer approach that dynamically decomposes a tree search to solve the one-eye problem [116]. Niu and Mueller improved the safety solver by developing a search-based region merging technique, efficiently strengthening weakly depending regions [117]. The series of work by Mueller et. al. attempted to improve their safety-of-territory solver. In [118], Niu et.

al. further improve this safety-of-territory solver by studying more complex situations in Go, that is, Seki, or the dual-life of a group.

While Mueller et. al. conducted a series of related research, other studies also should be mentioned. Chen and Chen performed a static analysis of the life-death problem in [54]. Fotland explained the static eye analyzer built into The Many Faces of Go in [53]. In [119], Lee extensively used various CI techniques to solve the life-death problem. An unsupervised learning method, more specifically, a two-layered Kohonen neural network (KNN), clusters similar life-death patterns and analyzes eye shapes to conclude the life-death of a group. To predict the life-death problem, Van der Wert et. al. trained feedforward neural networks with gradient descent with momentum and adaptive learning, as well as RPROP backpropagation from learning examples extracted from game records [120]. GoTools, developed by Thomas Wolf, undisputedly has been the strongest tsumego solver or life-death problem solver [121,122,123]. It incorporates many functionalities such as a table, rules, and a depth-first alpha beta search algorithm [115].

Other CI techniques, such as evolutionary computing and the support vector machine (SVM), also are used in Go. Richards et. al. used symbiotic adaptive neuro-evolution (SANE) to train a three-layer neural network [124]. Lubberts and Miikkulainen used a co-evolving architecture [148]. Churchill et. al attempted to combine soft methods, such as neural networks, and hard AI methods, such as the alpha-beta pruned minimax game tree search [125]. An SVM was used to detect the community structures or loosely connected stones [126].

1.4. PARTICLE SWARM OPTIMIZATION BACKGROUND

The goal of an optimization problem is to find the optima of the objective function, which represents the quality of each solution, with respect to given constraints. The optima can be either minima or the maxima of the objective function. Figure 1.4 shows the various optimization methods according to Rui Mendes [150] modified for this dissertation.

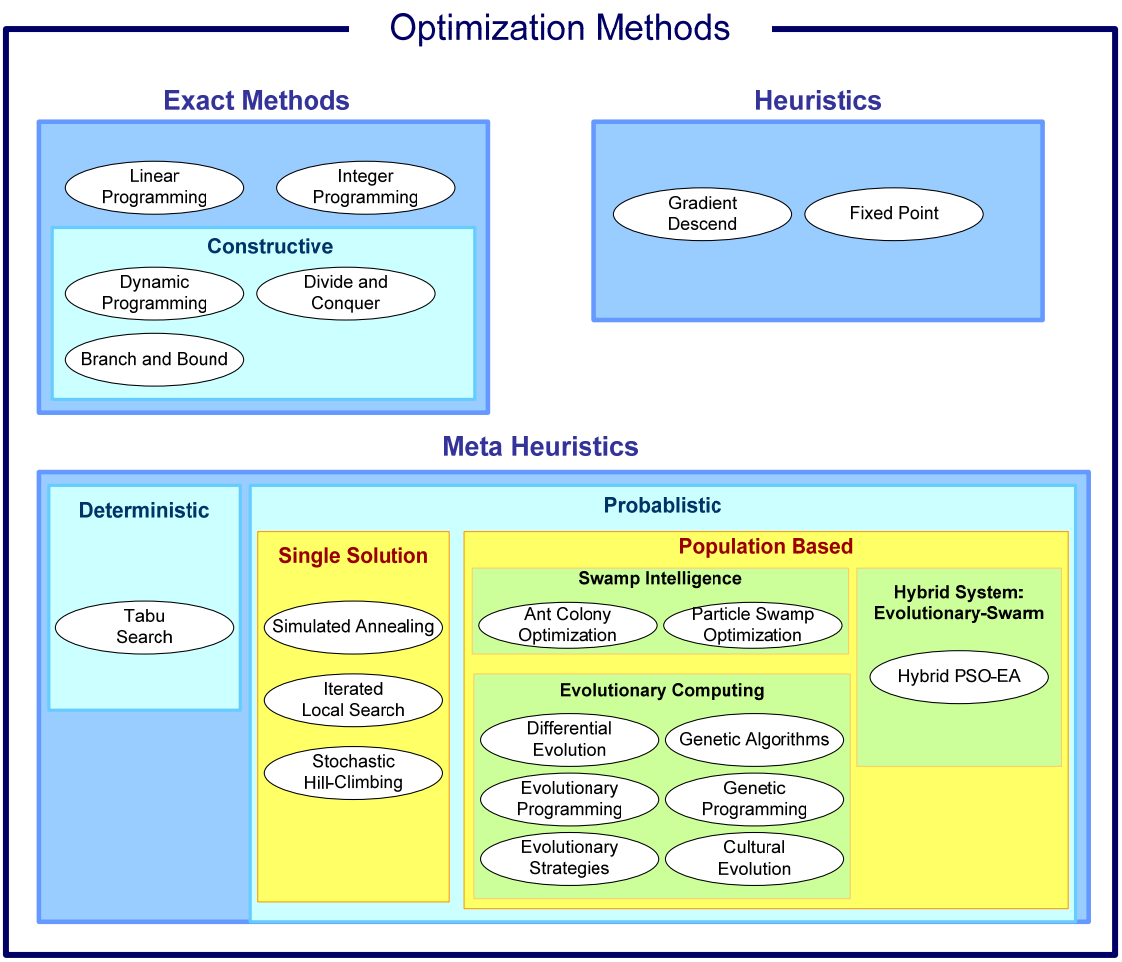


Figure 1.4. Various optimization methods.

Particle swarm optimization (PSO) is originally proposed by James Kennedy in 1995 [146] as a simulation of social behavior, and was introduced as a tool to solve the global optimization problem. We employ PSO because of its simplicity and the ability to escape a local minimum. PSO is one of the population-based probabilistic meta-heuristic optimization methods. In the CI literature, PSO belongs to the swarm intelligence (SI) inspired by the study of swarm organisms such as a school of fish, flock of birds, and colony of ants. Our literature review reveals that no one has used PSO to train the neural network architecture we use for the computer Go research.

Since PSO is a population-based algorithm, the number of whole population is set, say 20, and the location of each individual is updated according to equations 1 and 2. The subscript i indexes each individual or particle.

$$V_{i+1} = w \cdot V_i + c_1 \cdot rand_1 \cdot (P_i - X_i) + c_2 \cdot rand_2 \cdot (G_{best} - X_i) \quad (1)$$

$$X_{i+1} = X_i + V_i \quad (2)$$

These equations represent vectors in the problem space. The velocity update equation equation (1) updates the velocity V_i of particle i; the location update equation equation (2) renews the location X_i of particle i. The velocity update equation consists of 3 parts: the momentum, the cognitive acceleration, and the social acceleration parts. Influence of each part to the current velocity is controlled by two memories P_i, G_{best} and three parameters w, c_1, c_2. This velocity is moving towards both the local best of the given particle and the global best positions of the whole population with certain inertia.

The first part of the velocity update equation, i.e. momentum part $w \cdot V$, determines the ratio of the previous velocity reflected to the current velocity where the ratio is determined by inertia constant w. This concept of inertia was not in the initial PSO equations. It was developed by Shi and Eberhart in 1998 [147] from the desire to reduce or even eliminate the importance of $V_{\max}$ because the choice of $V_{\max}$ seemed too problem specific and has no rule of thumb.

The second and third parts are the cognitive acceleration term $c_1 \cdot rand_1 \cdot (P_i - X_i)$, which reflects the particle's own experience, and the social acceleration term $c_2 \cdot rand_2 \cdot (G_{best} - X_i)$, which forwards the population toward the best location found so far. These two terms are balanced by the acceleration constants c_1, c_2. The cognitive term utilizes information on the best location of the particle in its history, i.e. the local best P_i. On the other hand, the social term is governed by the global best G_{best}, the best location of the entire population.

The global best is the best of the local best. Therefore, we compare the fitness of the best of the local best to the fitness of the global best. If the current global best is worse, the global best is updated.

1.5. DISSERTATION PROBLEM STATEMENT

In this dissertation, the following four problems are tackled:

1. How to learn a Go game tree autonomously under constraints of myopic scope and information exchange limited to the immediate neighbors when the elements are connected only vertically and horizontally (computer Go).

2. How to estimate the group size of elements under the same constraints as in the previous work (computer Go).

3. How to innovate the traditional interface with a novel human-machine interface (Go robotics).

4. How to deliver more packets in highly disrupted computer networks (computer networks).

The above topics are related to the main theme as follows.

1. A Go player decides the next move from the current board status, which is spatial information. To select the best possible move, the player must possess the ability to foresee a sequence of future board statuses when a sequence of moves is chosen. A game tree in Go is a collection of board statuses. Learning the game tree means learning about the spatial information that is used to select the best possible moves over time.

2. An autonomous computing device perceives the system from the limited information about the space provided by the immediate neighbors. The device's perception of the system changes over time. The goal of each device is to estimate the size of the group to which it belongs.

3. The proposed human-machine interface is a system that perceives the current board status and places/removes a Go stone or stones on the Go board over time. In other words, actions must be taken based on the perceived spatial image over time.

4. In order to route a packet to the destination, each node or router in a computer network perceives the network status and selects the best possible route (or routing scheme, in this case) over time.

1.6. TECHNICAL APPROACHES AND CONTRIBUTIONS

1.6.1. Technical Approaches. According to Engelbrecht [1], CI is categorized into four paradigms: artificial neural networks (ANN), evolutionary computing (EC), swarm intelligence (SI), and fuzzy systems (FS). These four paradigms are classified further into sub-domains, which are depicted in Figure 1.5. On the other hand, artificial intelligence (AI) can be seen as a combination of several research disciplines, such as computer science, physiology, and philosophy [1]. The CI algorithms in Figure 1.5 also interact with machine learning, which is a category in AI. In other words, these CI algorithms also can be categorized as machine learning techniques. How an intelligent algorithm is grouped may depend on the scientist/engineer's background.

In this dissertation, two paradigms of CI techniques are used: ANN and SI. ANN has three major training principles and we use all of them: supervised learning, unsupervised learning, and reinforcement learning. The method used by the first experiment employs cellular simultaneous recurrent network (CSRN) trained with a variant of PSO called cellular PSO (CPSO). CSRN is a supervised learning neural network (SLNN). The proposed algorithm for the second experiment is called collectively cooperative learning (CCL) and it falls into the category of the supervised learning. The last experiment employs Q-learning, which is a reinforcement learning approach.

On the other hand, the third experiment combines robotics, network programming, and image processing techniques. In order to reduce the development time of the prototype Go robot, we used an off-the-shelf robot arm and an image processing library.

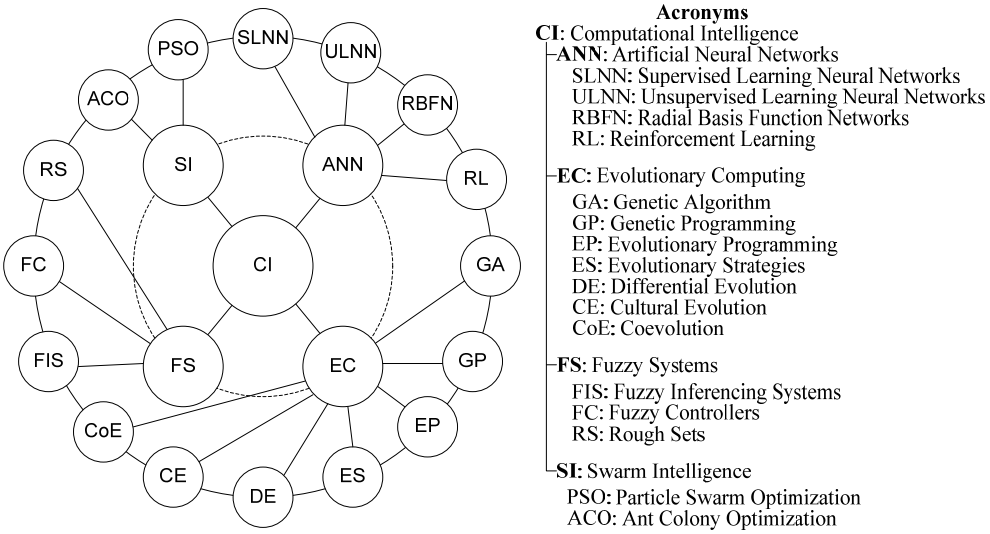


Figure 1.5. Four paradigms of CI and sub-domains of those paradigms according to Engelbrecht's classification of CI.

1.6.2. Contributions. The first experiment represents the first training of a modified cellular simultaneous recurrent network (CSRN) trained with cellular particle swarm optimization (CPSO). Another contribution is the development of a comprehensive theoretical study of a 2x2 Go research platform with a certified 5 dan Go expert. The proposed architecture successfully trains a 2x2 game tree.

The contribution of the second experiment is to develop a computational intelligence algorithm called *collective cooperative learning* (CCL). CCL learns the group size of Go stones on a Go board with zero knowledge by communicating only with the immediate neighbors. An analysis determines the lower bound of a design parameter that guarantees a solution.

The contribution of the third experiment is that we are the first, to our knowledge, to contribute research on Go robots and to articulate the human-computer interface in the existing literature. Another contribution is the proposed unified system architecture

interfacing Go Text Protocol (GTP) [9], which is the *de facto* standard for open-source computer Go engines to interface with a Go GUI program.

The last experiment tackles a disruption-tolerant routing problem for a network suffering from link disruption. Our contribution is to model this problem with MDP and to propose a solution to make the popular link state routing scheme more robust to link disruptions. The packet delivery rate has been improved under a range of link disruption levels.

1.7. ORGANIZATION

This dissertation is organized as follows. The four problems described in Section 1.5 are presented in each chapter followed by conclusions in Section 3. The chapter titles of the four experiments are listed here for convenience:

1. Modified Cellular Simultaneous Recurrent Networks with Cellular Particle Swarm Optimization (Section 2).

2. Recursive and Nonrecursive Algorithms for the Group Size Counting Problem in Computer Go (Paper 1).

3. Robotic Go: Exploring a Different Perspective on Human-Computer Interaction with the Game of Go (Paper 2).

4. Reconfigurable Disruption Tolerant Routing via Reinforcement Learning (Paper 3).

2. MODIFIED CELLULAR SIMULTANEOUS RECURRENT NETWORKS WITH CELLULAR PARTICLE SWARM OPTIMIZATION

2.1. INTRODUCTION

The CSRN possesses the potential to outperform the popular MLP [135,136] in that it can solve the connectedness problem that the MLP architecture has yet to solve. On the other hand, the added complexity of the CSRN architecture is a two-edged sword in that training the network itself proves challenging. Along with this complexity, the inherent structure interconnecting adjacent cells further complicates the training process and may be a limiting factor in using CSRNs for new problems. Regardless of these complexities in both the architecture and training of the network, studying CSRNs to solve novel problems is an interesting, scholarly endeavor because of their potential to surpass the popular MLP. The CSRN was proposed by Werbos and Pang [128] to solve the maze navigation problem and has been improved by Ilin et. al. [130,131,132,133] and Wunsch, who proposed the closed solution for CSRNs in [129]. CSRNs have been applied to image processing [137,138,139] and power applications [140]. Iftekharuddin et. al. applied clustering to solve the maze problem in [141].

Computer Go is considered more than just an unsolved problem; it also serves as an excellent test bed for novel applications to practical problems because of its clear-cut objective and the vast amount of sub-problems that allow for tests of practicality. For the purpose of this study, using the full Go board size is inappropriate because CSRNs are at their infancy. Instead, the smallest meaningful board size of 2x2 is used with the hope that the CSRN's operation can be analyzed in full. The 2x2 game tree is much more involved than expected. For example, even in cooperation with a Go expert, constructing

the game tree required weeks of work, including the time used to modify the full board rules for the 2x2 case.

Various types of neural networks are applied to computer Go problems. Chan et. al. [86] and Zaman and Wunsch [87] used feed forward networks with temporal difference (TD) methods for position evaluation. Mayer [142] applied a similar approach to learn board representation. Cai and Wunsch used the heuristic dynamic programming (HDP) method [143], and evolutionary algorithms were used by Kendall et. al. [144] and Mayer and Maier [145]. Wu and Baldi [89] applied recurrent neural networks to learn the evaluation function, and they are the only researchers who the current authors found in reviews of the literature who have applied a recurrent neural network structure in computer Go.

The authors make several contributions through this chapter. We are the first, to our knowledge, to use a CSRN to solve a problem in computer Go. We propose a CSRN trained with cellular PSO. We also developed an accurate and comprehensive mathematical foundation and game tree for the 2x2 board.

This chapter is organized as follows. Section 2.2 describes the problem by providing an overview of Go, developing a mathematical foundation for the problem under investigation, and presenting the game tree of a 2x2 Go board. Sections 2.3 and 2.4 propose the neural network architecture and training algorithm used to solve the problem, explaining CSRNs and cellular PSO as a training algorithm for CSRNs. The simulation results, conclusions and future research follow.

2.2. GAME OF GO

2.2.1. The 2x2 Game Rules. Even if the norm for playing contemporary Go is to use a19x19 board for a regular game, several examples show that this norm can be broken for educational and historical purposes. 5x5, 6x6, 9x9, and 13x13 boards have been used for educational and research purposes. Historical records show that Go's board size for typical games progressively enlarged from its original 9x9 size. An example of a larger board is the 25x25 board at the museum in the Department of Baduk Studies at Myongji University. The norm for a Go board can be broken not only for size, but also for shape. A non-square Go board with 361 play points is being exhibited at the Go Hall of Fame and Museum of Nihon Ki-in.

What the above facts signify is that the board size may be changed to suit one's purpose. For this study, a 2x2 Go board was chosen. Before discussing the 2x2 board, it is necessary to settle the white player's compensation. Even if determining the accurate compensation for a white player is a prerequisite to properly evaluating a game, no comprehensive theoretical study has been conducted for this board size. This work may have been neglected because computer Go researchers rely on the Go literature for this kind of information, but the literature focuses on the 19x19 board. A 2x2 board looks trivial from a Go expert's perspective, as well as for many computer Go researchers. However, it is non-trivial from a CSRN perspective. The regular board size is too large to perform a thorough analysis and must be scaled down to a level at which analysis is possible. No comprehensive study of the 2x2 board was found in the literature, so this tedious study had to be conducted as part of the current study in close cooperation with a Go expert. The results show that most of the rules remain identical. The tricky and

critical component is the compensation for white, which is zero according to Theorem 1 in Appendix A.

2.2.2. 2x2 Go Game Tree. A complete game tree of a 2x2 board for a starting move on the left bottom play point (Figure 2.1.c.1) is depicted in Figure 2.2. This figure is only one-fourth of the game tree when both players play randomly, and all the bottom states in the figure are recurring states that return to one of the states in the tree. This figure is presented to emphasize the fact that an accurate analysis of the 2x2 game tree is non-trivial. The remainder of the game tree is omitted because of space and time limitations. See Appendix B to view the actual game tree.

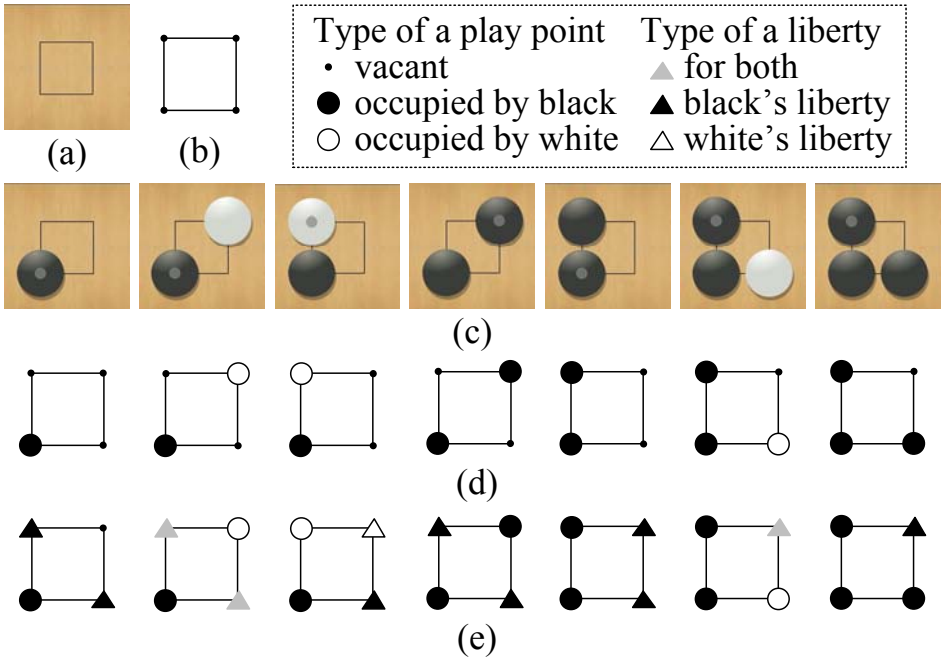


Figure 2.1. Sample 2x2 board status and graph notation.
(a) 2x2 Go board, (b) G_2 graph representation of (a), (c) representative board statuses (labeled 1 from the left). A representative board status can be rotated and flipped to match other possible board statues. The stone color can also be switched,
(d) a graph representation of (c), (e) liberties of (c).

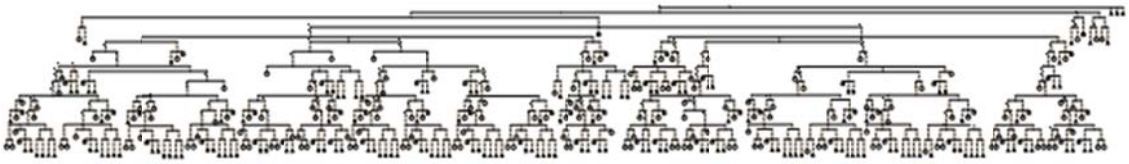


Figure 2.2. A game tree of a 2x2 board for a starting move on the left bottom play point. Refer to Appendix B for the game tree in a readable size.

Figure 2.3 extracts meaningful moves from the full game tree. Figure 2.3.a. illustrates both players playing optimally, which results in a draw regardless of territory or area rules. This substantiates Corollary 6. Figures 2.3.b and 3.c, respectively, are the cases in which black and white play optimally according to territory rules while the opponent plays randomly. Territory rules were chosen for this study in consideration of the amount of work necessary to examine both rules.

In Figure 2.3.b, black wins by forming two eyes when white's first move is not optimal. The game ends because no suicide move is allowed, according to Definition 14. If white moves optimally, the game either is a draw or returns to its beginning status. The recurrent state marked by a circle must not be confused with the repeated board status, which is prohibited. A repeated board status may result in mutual life (*Big* in Korean, *Gonghuo* in Chinese, and *Seki* in Japanese). This status is caused because any other move results in losing the game, so both players cannot help repeating the board status infinitely. Unlike the mutual life status, the recurring state has a chance for other moves. Therefore, the game continues after returning to the top of the game tree.

In Figure 2.3.c, the terminal states for white are either to win or draw a game. It is also possible to return to the top of the game tree (marked with a circle). Black's first move includes a pass. In this case, the sub-tree is identical to Figure 2.3.b when the stone color

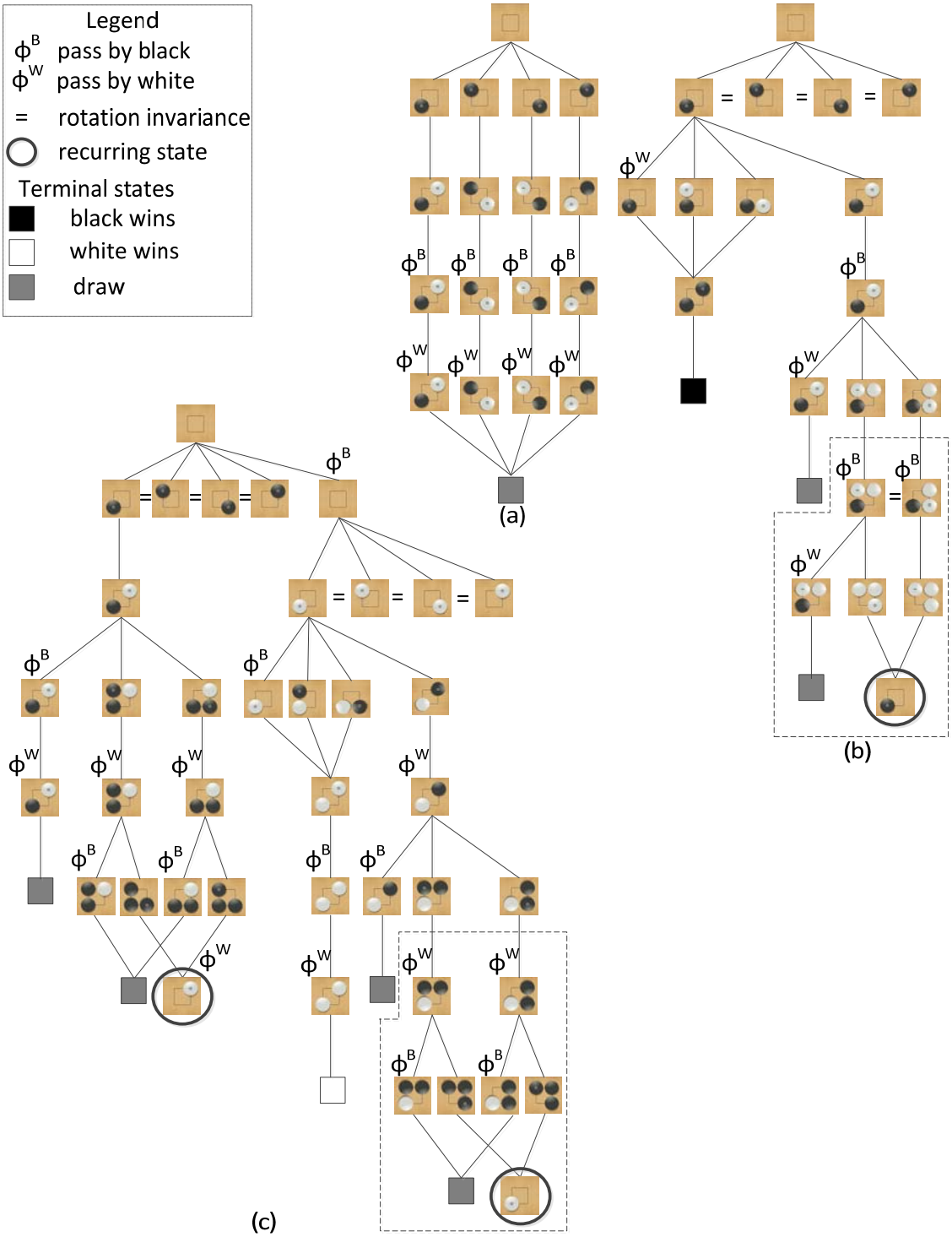


Figure 2.3. Selected game trees for optimal moves in territory rules.
(a) both players play optimally, (b) black plays optimally and white plays randomly,
(c) black plays randomly and white plays optimally.

switches from black to white and vice versa. One difference is that white's pass results in a draw, but this is not white's optimal move because white has a chance to win by continuing the game. The other difference is that the branches on the right bottom equal sign or rotation invariance in Figure 2.3.b is explicitly illustrated in Figure 2.3.c. Rotation invariance is a property of a board status that rotates and flips the status results in an identical end state.

2.3. CELLULAR SIMULTANEOUS RECURRENT NETWORK

2.3.1. Conventional Cellular Simultaneous Recurrent Network. A CSRN is a neural network architecture that consists of SRNs in the cellular structure. Each SRN or cell is interconnected with the neighboring cells. This internal structure provides a framework that allows SRNs in the system to share their outputs throughout the system. Figure 2.4 illustrates the general architecture of a CSRN system, and Figure 2.4.a shows the cellular architecture. Cells are connected vertically and horizontally. As illustrated in Figure 2.4.d, each cell has a neural network whose output is fed back to its input. The popular MLP is chosen as the neural network structure. The timing of the neural network output being sent out to the cell determines the cell's overall structure. If the feedback is iterated only once, the cell is a recurrent network. Multiple iterations make the cell a simultaneous recurrent network. Eight inputs, excluding the bias, are presented to the neural network because the four neighboring cells each contribute a pair of data. The input flow from the neighboring cells is depicted in Figure 2.4.b. The raw input to a cell n is x_n, and the cell's output is y_n. The center cell is labeled zero, and labeling continues clockwise from the top. Note that a pair (x_n, y_n) is an element of the system's input and

output, too. Figure 2.4.b shows the input flows from neighboring cells, but the cell receiving the flows also passes a pair of its raw input and cell output to the neighboring cells. Figure 2.4.c shows that the center cell receives pairs of data from neighboring cells and sends its own pair of data to them.

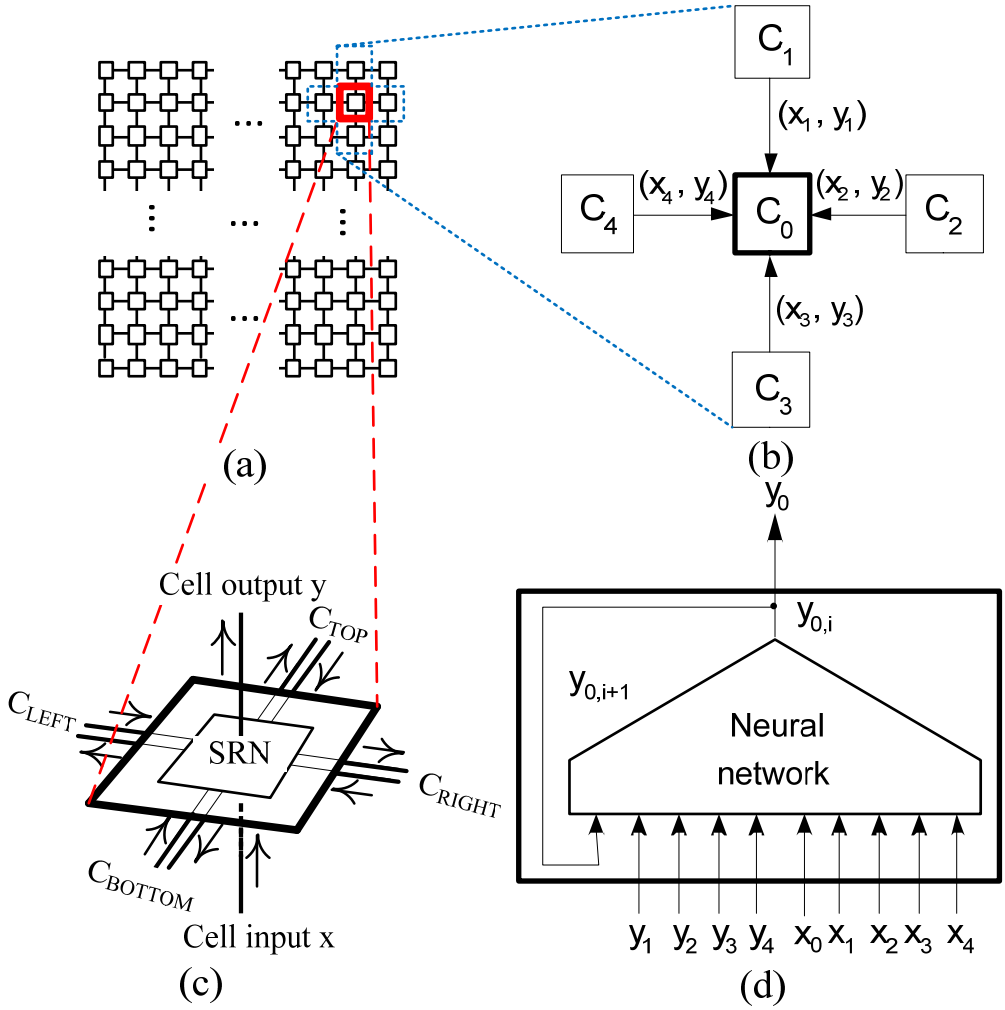


Figure 2.4. The CSRN structure.
(a) The general cellular architecture of a CSRN system,
(b) The input data from neighboring cells to a single cell,
(c) The structure of a single cell with emphasis on the input-output flows,
(d) The structure of a single cell from the neural network's perspective.

Visualizing the data flow in 3D helps to clarify the overall data flow. Consider Figure 2.4.c and Figure 2.5 together in order to understand how a CSRN operates. In Figure 2.5, a batch of training input planes is provided to the system. The system is illustrated as a grid of cells connected vertically and horizontally. A matrix of inputs to called an *input plane* to emphasize the fact that the raw input data forms a layer in the entire training data set. Given the batch of input planes, the corresponding batch of output planes are given out at each epoch of the training process. In this figure, only a batch of output planes is presented for simplicity. With this training process in mind, turn again to Figure 2.4.c. Imagine that the raw input to the cell comes from the bottom, and the

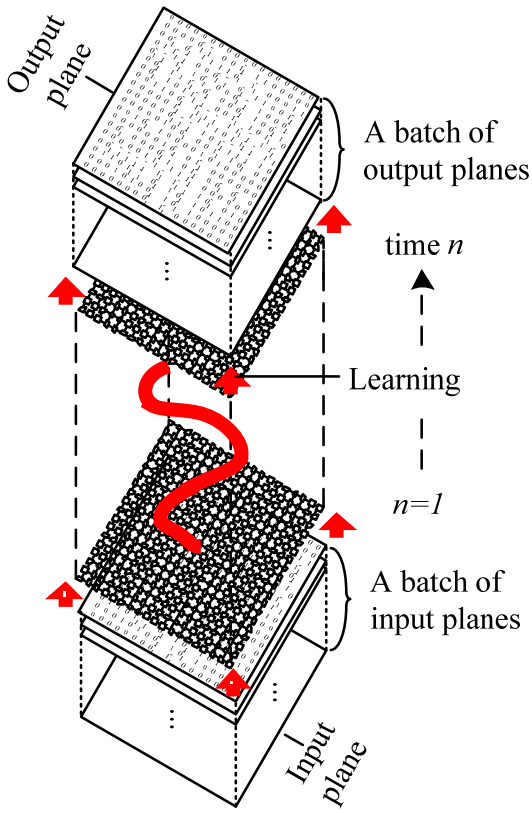


Figure 2.5. The big picture of the learning process over time.

neighboring bottom, and the neighboring cells pass their raw inputs and the current cell outputs to the side. This data exchange occurs at the cell level at each epoch. The neural network in Figure 2.4.d computes all the received data, iterates its output several times, and then outputs the last neural network input outside the cell, which is the cell output. This process occurs in all the cells of the system simultaneously, so the CSRN architecture is quite complex.

2.3.2. Modified Cellular Simultaneous Recurrent Network. One problem under investigation is to learn the optimal moves in the 2x2 game tree. Therefore, the general CSRN architecture is reduced to a 2x2 cell architecture, as given in Figure 2.6.a. Each cell has two neighboring cells; thus, there are six neural network inputs, excluding the bias (Figure 2.6.b): the cell's raw input x, the neighbors' raw input x_{n1} and x_{n2}, the cell's recurrent output y_{i+1}, and the neighbors' feedback outputs y_{n1} and y_{n2}. An index i is added to emphasize the internal loop within the cell. The conventional CSRN does not utilize the normalized input dimension reduction (NIDR) in Figure 2.6.b, and all six inputs are fed directly to the neural network.

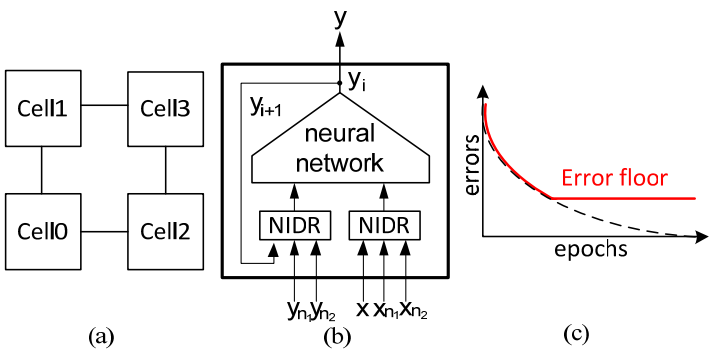


Figure 2.6. The proposed modified CSRN structure.
(a,b) Proposed modified CSRN for the 2x2 Go board,
(c) An error floor in the learning curve caused by the multi-valued function problem.

Applying the conventional CSRN to this problem fails to produce a solution. Rather, the learning curve hits an error floor, as illustrated in Figure 2.6.c. In other words, it does not converge to a target error, which is very small; rather, it converges to a much higher error than the target. An analysis of this phenomenon reveals that the conventional architecture suffers from the *multi-valued function problem*. A neural network is a function approximator. A function demonstrates one-to-one correspondence between the input and the output. The CSRN's training data is a function in that an input plane is mapped to an output plane. In other words, there exists one-to-one correspondence between a batch of input planes and a batch of output planes. However, one-to-one correspondence between a batch of neural network inputs in a cell and a batch of outputs is impeded because of the CSRN's internal structure. Some data exhibit one-to-many correspondence, which is a multi-valued function. A multi-valued function is not a function in a strict sense, so this name is a misnomer. Therefore, a neural network is prone to more errors than the samples with one-to-many correspondence.

Two techniques are introduced to overcome this problem: ternary coding of the Go board's representation and a *normalized input dimension reduction*. The former technique changes the numerical mapping of the Go board's stones or symbols to a ternary number. A popular Go board representation is to map black, empty, and white stones to a combination of (-1,0,1). For example, black, empty, and white are mapped to (1,0,-1), while the mapping in this study is (2,1,0). Converting a board's status to a ternary number enables the employment of other techniques used in the ternary world. This perspective change introduces the next. The normalized input dimension reduction converts a ternary number to a decimal number normalized to the range of [0,1]. This

technique reduces the complexity of the CSRN structure. Without this technique, a neural network in a cell receives seven inputs, including the bias; with the technique, it receives only three inputs, resulting in fewer synaptic weights. This savings is seen in the number of total cells, which is four in the current case because there are four cells in the CSRN.

2.4. CELLULAR PARTICLE SWARM OPTIMIZATION

The PSO algorithm should be modified to fit into the CSRN framework. A cellular PSO is proposed in Equations (3)-(8). It takes a divide-and-conquer approach to discover the desired weights for all the cells distributed throughout the topology. The total number of particles in the system N_p is the sum of the number of particles in all cells.

$$N_p = \sum_{c=0}^{N_c-1} N_{p,c} \tag{3}$$

where $N_{p,c}$ is the number of particles in cell c, and N_c is the total number of cells. The particles are allocated equally to the cells for simplicity. That is,

$$N_{p,c} = \frac{N_p}{N_c} \tag{4}$$

The domain size d of each PSO in all cells is equal to the number of weights $N_{w,c}$ in the corresponding cell.

$$d = N_{w,c} \qquad \forall c \in \mathrm{C} \tag{5}$$

where c is the cell index, and C is the set of cells. In all the cells, each PSO is trained by Equations (6)-(8). An *activation delta function* δ_a is introduced in Equation (8). The cell is activated or awake when it needs to be trained; otherwise, it hibernates.

$$x_{c,p} = x_{c,p} + v_{c,p} \tag{6}$$

$$\begin{aligned} v_{c,p} = \delta_a \{ & w_c v_{c,p} + c_{c,1} rand()(pbest_{c,p} - x_{c,p}) \\ & + c_{c,2} rand()(gbest_c - x_{c,p}) \} \end{aligned} \tag{7}$$

$$\delta_a = \begin{cases} 1, & \text{if the cell is active.} \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

where $x_{c,p}$ and $v_{c,p}$ are the location and the velocity of particle p in cell c; w_c, $c_{c,1}$, and $c_{c,2}$ are the design parameters; *rand()* is a uniform random variable distributed over [0,1]; and $pbest_{c,p}$ and $gbest_c$ are the particle best for particle p and the global best for all the particles in cell c, respectively. Equations (6) and (7) are the velocity update equations based on [147,148].

Hibernation may save a significant amount of computational resources when a cell reaches its target. However, the hibernating cell must awaken when a neighboring cell's output changes.

2.5. SIMULATION CONFIGURATION, RESULTS AND DISCUSSIONS

2.5.1. Configuration. The modified CSRN trained with cellular PSO learns the optimal moves for black on a 2x2 Go board. Tables 2.1 and 2.2 summarize the simulation configurations. Managing the clocks needed for simulation, the internal and external clocks, is tricky. The internal clock is the time clock used to iterate the feedback output

within a cell. This clock is similar to the time stamp used in training conventional neural networks, such as MLP. While the internal clock exists at the cell level, the external clock exists at the system level. All the cells output and exchange pairs of data with neighbors in reference to the external clock. Conceptually, all the cells compute simultaneously, so their data exchange must be synchronized appropriately in order to emulate this simultaneous operation. The training data is carefully chosen. The current board status is an input plane to the CSRN, and the desired board status is the output plane. For black, a batch of training data is taken from Figure 2.3.b. The initial neural network weights follow a uniform distribution between [0,1].

Table 2.1. Simulation configurations for the CSRN.

Parameters	Values
Number of cells Nc	4
Data representation/data offset value	Ternary/1.0
Activation function (most layers/output layer)	Tansig/modulo-3
Tansig slope parameter/initial cell output value	1.0/1.0
Number of neurons in hidden layer/output layer	7/1
Number of weights Nw,c	56

Table 2.2. Configuration for cellular PSO.

Parameters	Values
Target error (system/cell)	0.0/0.0
Maximum iterations for the outer/inner loop	100/5
Particle size $N_p/N_{p,c}$	40/10
Design parameters $w_c/c_{c,1}/c_{c,2}$	0.20/2.0/2.0
Range of $x_{c,p}$ / Initial value for $v_{c,p}$	[-100.0,100.0]/10^{-15}

2.5.2. Performance Metric. A cell error $e_{n,c}$ of cell c at time n is the overall error between the target cell output $\mathbf{T}_{n,c}^{p}$ and the estimated cell output $\hat{\mathbf{T}}_{n,c}^{p}$ for all the output planes. An absolute error is chosen because it is easier to assess the amount of cell errors than the mean squared error.

$$e_{n,c} = \sum_{p=1}^{N_p} \left| \mathbf{T}_{n,c}^{p} - \hat{\mathbf{T}}_{n,c}^{p} \right| \tag{9}$$

where p is an index for an output plane, and N_p is the total number of output planes. The system error e_n at time n is the sum of all cell errors for the system. That is,

$$e_n = \sum_{c=0}^{N_c-1} e_{n,c} = \sum_{c=0}^{N_c-1} \sum_{p=1}^{N_p} \left| \mathbf{T}_{n,c}^{p} - \hat{\mathbf{T}}_{n,c}^{p} \right| \tag{10}$$

2.5.3. Selected Result. Due to this chapter's space limitations, only one of the simulation results is presented, which explains the CSRN's properties well. Figure 2.7.a shows a successful training of black's optimal game tree. The CSRN is trained so that the optimal game tree for black can be learned with a system error of zero. In Go, accuracy is important because a single mistake may lead to losing a game. Training a CSRN to achieve a system error of zero is the most challenging part of this process. Both the neural network structure and the PSO design parameters should match in order to achieve this accuracy.

Figure 2.7.a presents the system-level training result. The cell-level training result is presented in Figure 2.7.b, which also shows two types of learning curves. The bigger learning curve is a cell error for the external loop, and the smaller one is for the loop

within a cell. The latter is presented to demonstrate the simultaneous part of the SRN, which facilitates learning. The first three samples in the former learning curve correspond to the samples at the internal clock settings 0, 5, and 10 of the latter curve. For example, Cell 1's initial error is 8 in both learning curves. After five iterations within a cell (smaller curve), the cell's error reduces to one, which is shown at $n_o=1$ in the bigger curve and at $n_i=5$ in the smaller curve. The cell errors at n_o in Figure 2.7.b add up to the system error at n_o in Figure 2.7.a.

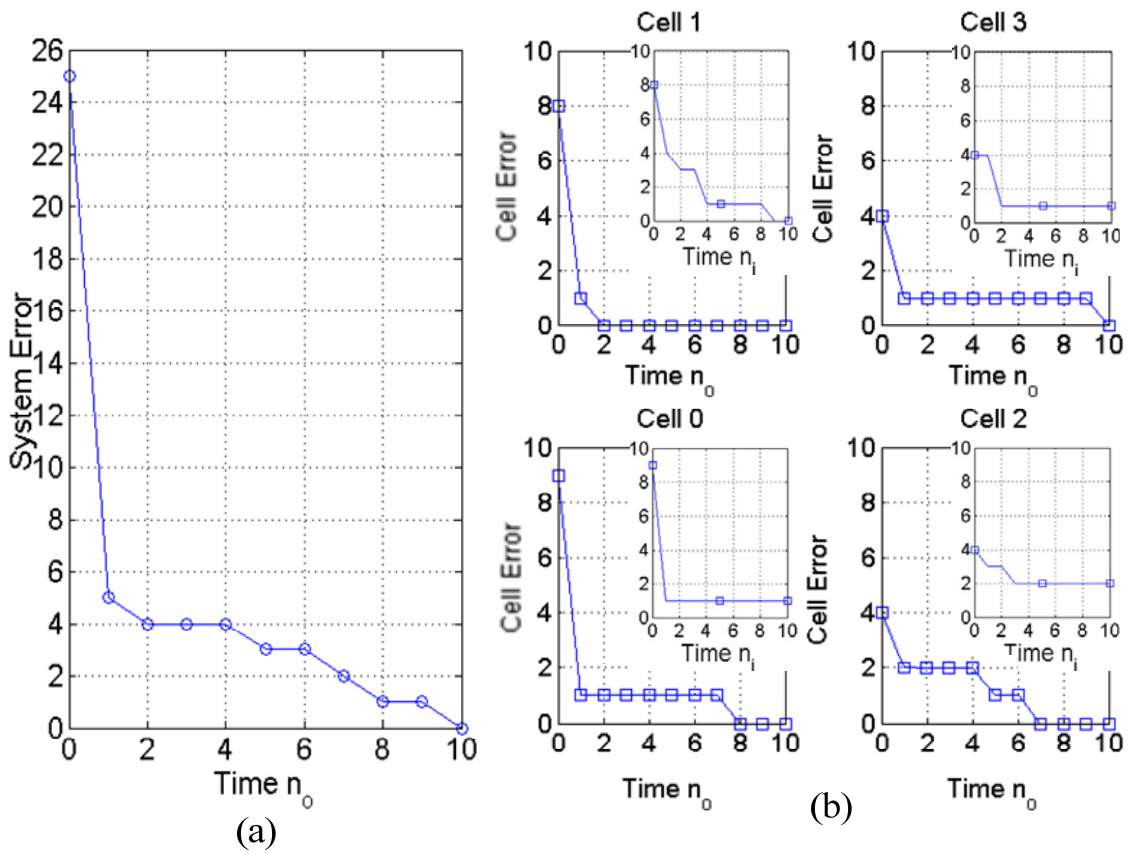


Figure 2.7. Learning curves for the 2x2 CSRN
(a) The system error with respect to the external loop n_o,
(b) Cell errors for the external and internal loops n_o and n_i, respectively.

2.6. CONCLUSIONS

In this chapter, a thorough theoretical study of a 2x2 Go case was performed. Based on the outcome, a modified CSRN trained with cellular PSO that learns the optimal moves of the 2x2 game tree was proposed. The CSRN's supervisory signal was constructed from the game tree.

The conventional CSRN suffers from a multi-valued function problem. A neural network learns a function whose input and output lie in one-to-one correspondence. The CSRN architecture takes the neighboring cells' output as a feedback input, along with raw input data. This structure converts the input and output blocks of a CSRN that exhibits one-to-one correspondence into one that exhibits one-to-many correspondence from the internal cell's (or neural network's) perspective. The term multi-valued function is a misnomer as it is not a function in a strict sense.

The problem is overcome by introducing two techniques: ternary coding of the Go board representation and a normalized input dimension reduction. The former changes the numerical mapping of a Go board's stone or symbol into a ternary number. One perspective change introduces the next. The normalized input dimension reduction converts a ternary number to a decimal number normalized to the range of [0,1]. This technique reduces the complexity of the CSRN structure. Without this technique, a neural network in a cell receives seven inputs, including the bias; with the technique, it receives only three inputs, resulting in fewer synaptic weights. This savings is reflected in the number of total cells, which is four in this case because there are four cells in the CSRN.

The simulation results show that the optimal game tree of a 2x2 Go board is trained successfully with cellular PSO. Each cell's internal loops facilitate learning the

training data, and the outer loop of the CSRN is synchronized at each epoch in order to feed cells' outputs back to cells' inputs at the next epoch. The number of total epochs required to finish the CSRN training varies because of the random nature in which PSO explores the problem space.

2.7. FUTURE RESEARCH

Even if the results presented in this chapter are thrilling, this work can be extended further. A number of future research directions include comparing the cellular PSO to other population-based training algorithms, such as the evolutionary algorithm, to implement this work in general purpose graphics processing units, and replacing the supervisory signal with a temporal difference signal in order to learn a larger board size.

PAPER

1. RECURSIVE AND NON-RECURSIVE ALGORITHMS FOR THE GROUP SIZE COUNTING PROBLEM IN COMPUTER GO

ABSTRACT

Computer Go has been recognized as an unconquered challenge to the Artificial Intelligence and Computational Intelligence societies since 1997 when Garry Kasparov, the human Chess champion, was defeated by the IBM supercomputer Deep Blue. Computer Go is considered more than just an unsolved problem; it also serves as an excellent test bed for novel applications to practical problems because of its clear-cut objective and the vast amount of sub-problems that allow for tests of practicality. We propose novel recursive and non-recursive approaches to count the size of groups of connected stones on a Go board. The simple and elegant recursive approach always answers the correct group size. Inspired by the former's tree traversal mechanism, the non-recursive approach guarantees to return the correct answers in an autonomous and scalable manner.

1.1. INTRODUCTION

Computer Go [2,8] is a field of study aiming to develop a computer intelligence that can generate a sequence of moves that successfully plays a game of Go. Go, also called Baduk in Korea and Weiqi in China, is a two-player board game of which the Western counterpart is Chess; it is as popular, or perhaps more popular, in East Asia as Chess is in the Western world and is gaining more popularity in other regions. Computer Go's technical significance was rediscovered in the quest for a more complicated problem than Chess after the IBM supercomputer Deep Blue defeated the human Chess

Master Garry Kasparov in May 1997. It is far more complex than computer Chess; the game tree size of a 19x19 Go board is estimated to be between 10^{575} to 10^{620}, while that of a Chess board is 10^{123} [7]. The implication is that computer Go cannot beat a human champion with the fastest computer available to date, making computer Go a novel test bed for computer intelligence. Currently, computer Go is considered one of the most challenging unsolved problems in the Artificial Intelligence and Computational Intelligence societies [3,6].

Due to the grand scale of the computer Go problem, it is divided into numerous sub-problems [5]. One of the sub-problems of interest in the current study is to count the size of connected stones or a group of stones. In other words, the target in this chapter is to develop an algorithm that can solve the problem in an autonomous and scalable manner. The answer to the problem provides essential information for judging the life and death of the group [10,13]. In general, the life and death of a group significantly impacts the entire game, of which the purpose is to gain more territories than the opponent. The game's full rules [5] may confuse a reader without experience with the game, possibly concealing the game's technical depth. Thus, only the relevant rules are covered in Section 1.2.

This chapter proposes both recursive and non-recursive algorithms guaranteed to solve the group size counting problem. Theoretically, the non-recursive approach with minor tweaks works for wired networks in a mesh topology and a wireless sensor network with limited connectivity, but the proposed solution does not claim to solve these cases or to avoid the fallacy of over-generalization.

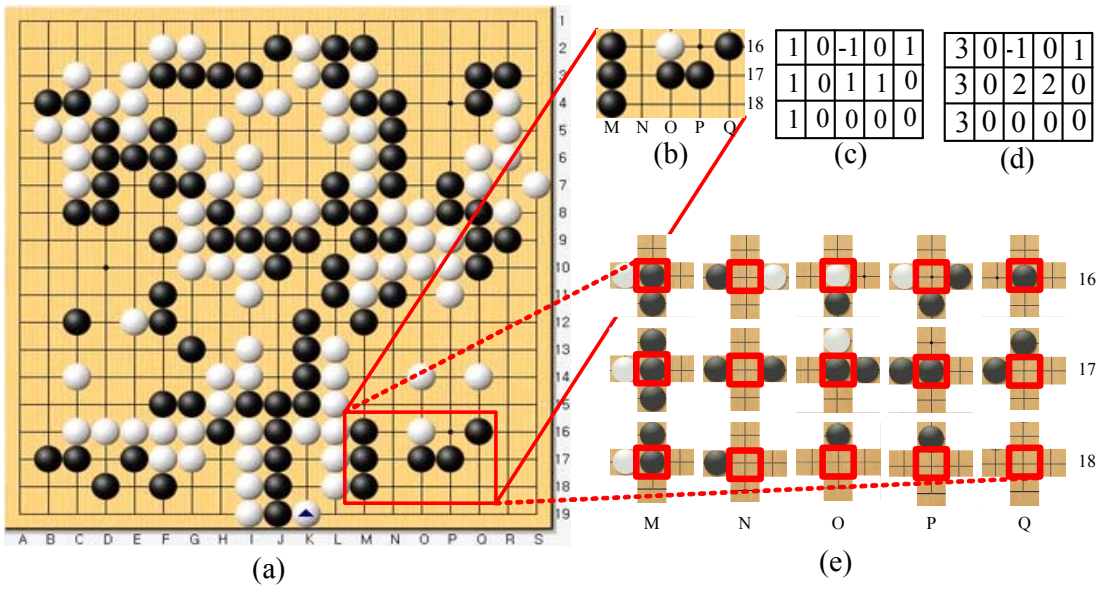
This chapter is organized as follows: Section 1.2 describes the group size counting problem in computer Go, and Section 1.3 explains the proposed algorithms. The following section discusses the simulation settings and results, respectively. Conclusions, acknowledgements and references conclude the chapter.

1.2. GROUP SIZE COUNTING PROBLEM IN COMPUTER GO

The group size counting problem in computer Go is a simpler sub-problem than the full computer Go problem, and its solution provides the essential information for determining the life-death of a group of stones. Figure 1.1 shows a snapshot of a match between two top professional Go players [1] used to describe the problem. Figure 1.1.b shows a fraction of the raw board status depicted in Figure 1.1.a. A popular board representation in computer Go is to assign -1 for white stones, 0 for empty intersections, and 1 for black stones, so Figure 1.1.c. provides such a data representation of Figure 1.1.b. The target answer for Figure 1.1.b is depicted in Figure 1.1.d. Each square cross $F_{location}$ in Figure 1.1.e is a board segment of Figure 1.1.a centered around a location. For instance, the top left square cross F_{16M} is a board segment centered around 16M and its adjacent locations (15M, 17M, 16L, 16N).

The goal of the group size counting problem is to obtain the size of a group or connected stones of the same color. According to Go rules, adjacency is defined vertically and horizontally, not diagonally. Therefore, there are three groups of connected black stones and one group consisting of a single white stone in Figure 1.1.b. The former group includes the vertical straight line of three stones (left), the horizontal straight line of two stones (middle), and a single black stone (top right). The sign of the white stone in

Figure 1.1.c. and the group size in Figure 1.1.d. are negated to distinguish them from those of black. The final correct answer is the group sizes for the entire board, but Figure 1.1.d. is shown here to explain the problem.



1	0	-1	0	1
1	0	1	1	0
1	0	0	0	0

3	0	-1	0	1
3	0	2	2	0
3	0	0	0	0

Figure 1.1. Group size counting problem example.
(a) A snapshot of a Go game, (b) a fraction of (a), (c) data representation of (b),
(d) the target answer or group size for (b),
(e) Each square cross $F_{location}$ is a board segment of (a) centered around a location.

Three reasons exist for tackling the group size counting problem. First, a Go board's status forms a 2D grid world with vertices; it is a special case of a grid world that can be related to graph theory. The differences are: (1) a standard Go board is square with a size of 5, 6, 9, 13, or 19; (2) only two different types of stones are used in Go, so black and white stones are placed alternately on Go board intersections; (3) the maximum number of edges for a vertex is limited to four in Go. The second reason is historical because the property of the group size counting problem was discovered while

performing research to combine cellular simultaneous recurrent networks [4,11,12] with computer Go. Lastly, it is convenient to test unexpected shapes formed during game play.

1.3. THE PROPOSED RECURSIVE AND NON-RECURSIVE ALGORITHMS

Both recursive and non-recursive solutions for the group size counting problem are proposed in this study. The perspective of the former is that of an arbiter overlooking a Go board, as given in Figure 1.1.a. In contrast, the visibility of the latter is limited locally to the immediate neighbors, as given in Figure 1.1.e.

1.3.1. The Recursive Approach. The proposed recursive algorithm counts the group size like a human, and this mechanism is implemented elegantly with function recursion. The idea is to traverse a board as a human moves his eyes and counts the group size. Assume one starts from the top left corner in Figure 1.2.a.; the top row is traversed quickly because no stone is placed. The attention, or one's eyes, stops on the black stone marked with the square. The attention then moves down to the black stone marked with the triangle, then back to the stone with the square, and then on to the black stone marked with the circle. As the human's attention travels from stone to stone, the group size is counted in the back of one's mind. Take as another example the group with nine white stones. For this group, the attention moves along the path "1→2→3→2→4→5→6→7→8→9," where each stone's number is indicated in Figure 1.2.a. Notice that stone 2 is visited twice. It should not be counted on the second visit, but returning to stone 2 retains the *forward path* of attention. In this manner, a human can count the group of nine white stones. This process is called the *counting phase*.

On the other hand, each element of the estimated group size matrix ***G*** in Figure 1.2.c knows the group size of the corresponding location. After the counting phase, the estimated group size *g*, or *g=9* in this example, is delivered to all stones in the group. The attention path traces back to Stone 1. The *backward path* is "9→8→7→6→5→4→2→3→2→1". This process is called the *group size propagation phase* or *propagation phase* for short.

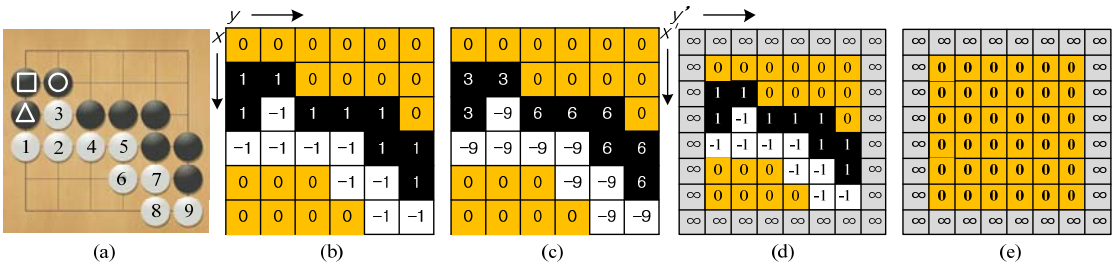


Figure 1.2. An example of the data structure for the proposed recursive scheme. (a) A raw board status, (b) Board status matrix ***B***, (c) Estimated group size matrix ***G***, (d) Dummy status matrix ***D***, (e) Visit mark matrix ***V***.

Algorithm 1.1 is the pseudocode of the proposed scheme. The current estimate of the group size *g* at location *(x',y')* is incremented by the *player color index p* (line 2). The visit is marked to avoid repeated counting (line 3). One must look around at the neighbors (lines 4-7) to recursively count the group size only if the *consanguinity visit test* (line 1) is satisfied. The consanguinity visit test checks if the neighbor is from the same kind/blood and, if so, whether or not it was visited earlier.

Notice that both black (*p=1*) and white (*p=-1*) are counted by a single equation (line 2), and child functions are spawned recursively (lines 4-7). Whether or not a child

function spawns its descendants depends on the consanguinity visit test. This solution is gracefully simple because the genealogy of a function is automatically managed by stack. Also note that the algorithm belongs to the depth-first search algorithm category [9].

Algorithm 1.1. The proposed recursive algorithm.

```
COUNT_GROUP_SIZE_RECURSIVELY(p, x′, y′, D, V, g)
 1 if p=D(x′, y′) and V(x′, y′)≠1
 2   g←g+p                // p is the player color index. p=1 for black; p=-1 for white
 3   V(x′, y′)←1
 4   [g,V]=COUNT_GROUP_SIZE_RECURSIVELY(p, x′-1, y′, D, V, g)     // Up
 5   [g,V]=COUNT_GROUP_SIZE_RECURSIVELY(p, x′+1, y′, D, V, g)     // Down
 6   [g,V]=COUNT_GROUP_SIZE_RECURSIVELY(p,  x′, y′-1, D, V, g)     // Left
 7   [g,V]=COUNT_GROUP_SIZE_RECURSIVELY(p,  x′, y′+1, D, V, g)     // Right
 8 return [g, V]
```

1.3.2. The Proposed Non-recursive Approach with Cellular Structure.

Cellular structure and the concept of baton passing play key roles in the proposed non-recursive scheme. The cellular architecture of the system is depicted in Figure 1.3.a. The system, in this example, consists of 361 cells interconnected only vertically and horizontally according to the adjacency definition in Go. Each (square-shaped) cell is an autonomous processing unit (APU) mounted with the proposed non-recursive algorithm. Therefore, all the cells can process the input simultaneously at each time clock.

Figure 1.3.b. illustrates the cell's structure. A cell exchanges information with the neighboring cells in the four cardinal directions, i.e., C_{TOP}, C_{BOTTOM}, C_{RIGHT}, C_{LEFT}, limiting the visibility and communication range of a cell accordingly. When an input is given to a cell, that cell communicates with the neighboring cells and then outputs the

computed value. As the input to the system is a matrix, it is computed in parallel with all the other cells in the system. In essence, a batch of system input is mapped to cells in the system at $n=1$. The system learns about the group size from zero knowledge and outputs a batch of estimated group size at $n=N_{end}$. In this figure, it is assumed that the system learns about a 19x19 Go board's status. The system input is a 19x19 board status matrix ***B***; hence, the cellular architecture is 19x19, as is the system output.

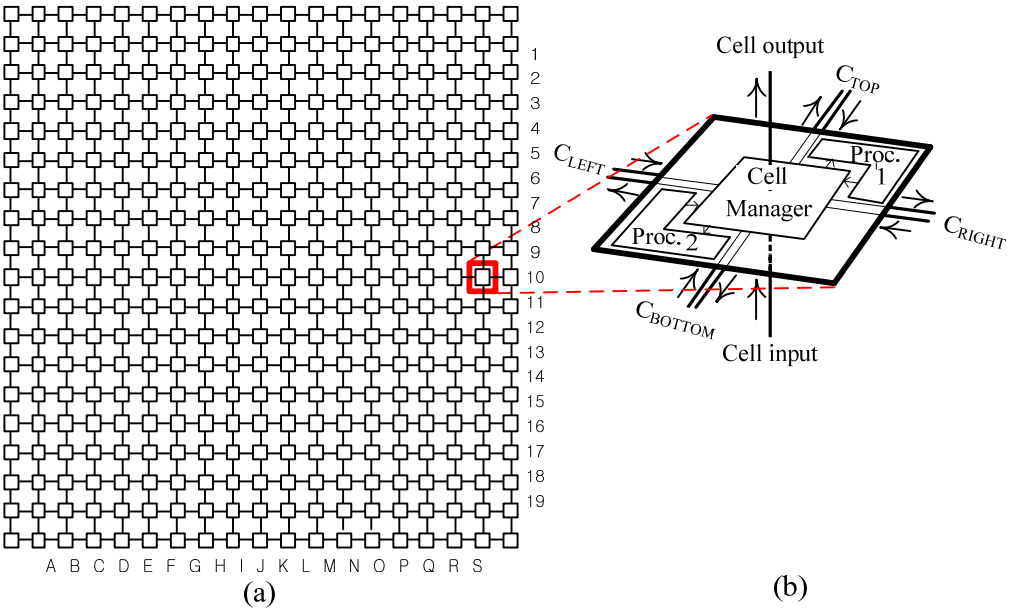


Figure 1.3. System architecture and macroscopic view of its operation.
(a) The cellular system architecture, (b) The structure of a single cell.

The baton-passing concept should be explained first to clarify the counting and propagation phases (Phases 2 and 3) of the non-recursive algorithm. The concept is explained best with an analogy. In a team of relay runners, only one runner with a *baton* runs for the team even if all the team members have the ability to run. When *n* teams are

in a stadium, *n* runners with batons are running, and the remainders are resting. The baton is passed to another runner in order to change the turn. This concept is borrowed to explain passing control from cell to cell in the same group. All the cells in a group except for the cell with the baton are hibernating. Only one cell per group is activated. A *passer* who passes a baton enters hibernation as soon as the baton is passed to a *receiver* who receives the baton. Obviously, the receiver is activated as soon as the baton is passed. It should be stressed that the system activates *n* cells when *n* groups exist in the input board status.

Two states are defined: the non-steady-state (NSS) and the steady state (SS). SS means that the leader (root cell) of a group is decided. In SS, only one cell per group is active. Conversely, all the cells are awake in NSS. In other words, parallel processing occurs blindly in NSS but in a controlled manner through baton passing in SS. In this study's implementation, processing block 1 in Figure 1.3.b. processes the root cell discovery phase (Phase 1) during NSS; processing block 2 is responsible for the phases (Phases 2 and 3) occurring during SS. A *cell manager* judges the current phase and selects the appropriate processing block. It not only allocates tasks to the processing blocks, but also controls the information flow. This information includes a cell's input and output, the baton, the current estimate of the root cell and group size exchanged with neighbors, and the decision regarding whether or not to release the cell output. Each phase has a purpose that can be omitted only if the phase is unnecessary.

The three phases are explained here in further detail. The goal of Phase 1 is to discover a group's root cell because of the differences between the recursive and non-recursive solutions. Recursive solutions know where to start (the root node) by iteratively

assigning a location on a board; in contrast, non-recursive solutions require that a group of cells agree on where to start. The leader of a group is called a root cell because a root cell corresponds to the root node in the tree constructed by the recursive algorithm. A cell exchanges its own estimate of the root cell (ERC), initialized to its cell index, with neighbors of the same kind or color. If this cell sees a smaller ERC than the current estimate within the acceptable scope (the cell's neighbors and the cell itself), it updates the current estimate to the smallest. This procedure spreads the smallest cell index to the entire group, as when water is poured at the entry point of a dried canal and spreads along the canal's waterway. Using the largest ERC works the same way but with a minor tweak. Once a cell is sure that all the cells in the group share the same ERC, it switches its internal state from NSS to SS. In steady state, all cells within the group share the same ERC. In other words, different groups can be identified by their ERC.

The idea behind Phases 2 and 3 is similar to the counting and propagation phases of the recursive solution, respectively. However, the implementation differs because the tree traversal mechanism in the recursive solution must be performed autonomously. In Phase 2, the estimated group size is incremented only when the consanguinity visit test is satisfied. In some shapes, this is not too complicated; all the cells in the same group may have the correct group size at the end of Phase 2. The root cell always has the correct group size as long as the cell traversal mechanism does not terminate prematurely. Phase 3 ensures that the correct answer is shared with all the group members via the cell traversal mechanism identical to that in Phase 2.

1.4. SIMULATIONS AND DISCUSSIONS

Both recursive and non-recursive schemes are tested on various Go board sizes, returning the correct answers for all possible configurations on 2x2 through 6x6 boards. For 13x13 and 19x19 boards, thirty game records are used to verify the correctness of the algorithms. The reason that all possible board configurations are not tested on 13x13 and 19x19 boards is two-fold. First, the proposed algorithms scale well for 13x13 and 19x19 boards because the logic regarding how to count the group size remains the same even for larger board sizes. Secondly, it is impossible to do so because of Go's large branching factor [2,8].

As an example, Figure 1.4 shows an input of a 19x19 Go board and the corresponding output, or the estimated board size. The proposed recursive algorithm correctly counts all the groups on the Go board; the proposed non-recursive approach learns the correct group sizes for all groups in an unsupervised manner. The non-recursive solution is correct only if the design parameter *excess waiting time* (EWT) ε is larger than the lower bound of *guaranteed excess waiting time* (GEWT) ε_g.

$$\varepsilon \geq \varepsilon_g \tag{1}$$

Each cell internally counts its own waiting time and moves to the steady state when the waiting time exceeds the EWT. This parameter is introduced as a measure by which to ensure that all the cells learn the root cell correctly.

Figure 1.5 shows the relationship between the board size b and the EWT for fully connected boards. In other words, all the intersections on a board are occupied by the same color stones. The solid line with asterisks is GEWT ε_g. Note that EWT ε above this

line always results in the correct group size; ε below it does not always lead to the correct answer. Carefully chosen, the correctness and the execution time are tradeoffs.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	-1	-1	0	0	1	-1	1	1	0	0	0	0	0	0
0	0	-1	0	-1	1	1	1	1	0	-1	1	-1	0	0	0	1	1	0
0	1	1	-1	-1	0	0	0	-1	-1	0	-1	1	1	0	0	1	-1	0
0	-1	-1	1	-1	1	0	-1	0	0	0	-1	-1	1	0	0	0	-1	0
0	0	-1	1	1	1	-1	0	-1	0	0	0	-1	1	0	0	-1	-1	0
0	0	-1	1	0	1	1	-1	-1	0	0	1	-1	1	0	1	-1	0	-1
0	0	1	1	0	0	-1	1	-1	-1	-1	1	1	-1	-1	1	1	-1	0
0	0	0	0	0	1	-1	1	1	1	1	0	1	1	-1	-1	1	1	0
0	0	0	0	0	0	-1	-1	-1	1	0	1	-1	-1	-1	-1	1	0	0
0	0	0	0	0	1	0	0	-1	0	0	1	-1	1	0	-1	0	0	0
0	0	1	0	-1	1	0	0	0	0	1	0	1	0	0	0	0	0	0
0	0	0	0	0	0	1	0	-1	0	1	-1	0	0	0	0	0	0	0
0	0	-1	0	0	0	0	-1	-1	0	1	-1	0	0	-1	0	-1	0	0
0	0	0	0	0	1	1	-1	1	1	1	-1	0	0	0	0	0	0	0
0	0	-1	-1	-1	-1	-1	1	-1	1	-1	-1	1	0	-1	0	1	0	0
0	1	1	0	1	-1	-1	0	-1	1	0	-1	1	0	1	1	0	0	0
0	0	0	1	0	1	0	0	-1	1	0	-1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	-1	1	-1	0	0	0	0	0	0	0	0

(a)

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	-2	-2	0	0	1	-2	3	3	0	0	0	0	0	0
0	0	-1	0	-4	4	4	4	4	0	-2	3	-1	0	0	0	3	3	0
0	2	2	-4	-4	0	0	0	-2	-2	0	-5	5	5	0	0	3	-5	0
0	-4	-4	5	-4	4	0	-1	0	0	0	-5	-5	5	0	0	0	-5	0
0	0	-4	5	4	4	-1	0	-6	0	0	0	-5	5	0	0	-5	-5	0
0	0	-4	5	0	4	1	-6	-6	0	0	5	-5	5	0	6	-5	0	-1
0	0	5	5	0	0	-6	1	-6	-6	-6	5	5	-10	-10	6	6	-1	0
0	0	0	0	0	1	-6	5	5	5	5	0	5	5	-10	-10	6	6	0
0	0	0	0	0	0	-6	-6	-6	5	0	2	-10	-10	-10	-10	6	0	0
0	0	0	0	0	2	0	0	-6	0	0	2	-10	1	0	-10	0	0	0
0	0	1	0	-1	2	0	0	0	0	10	0	1	0	0	0	0	0	0
0	0	0	0	0	0	1	0	-4	0	10	-7	0	0	0	0	0	0	0
0	0	-1	0	0	0	0	-4	-4	0	10	-7	0	0	-1	0	-1	0	0
0	0	0	0	0	2	2	-4	10	10	10	-7	0	0	0	0	0	0	0
0	0	-7	-7	-7	-7	-7	1	-4	10	-7	-7	3	0	-1	0	1	0	0
0	2	2	0	1	-7	-7	0	-4	10	0	-7	3	0	2	2	0	0	0
0	0	0	1	0	1	0	0	-4	10	0	-7	3	0	0	0	0	0	0
0	0	0	0	0	0	0	0	-4	10	-1	0	0	0	0	0	0	0	0

(b)

Figure 1.4. An example of a 19x19 Go game.
(a) The board status matrix *B* for Figure 1.1.a. as an input for both proposed schemes,
(b) The estimated board size matrix *G* for both schemes.

An equation to choose ε within the given boundary is suggested in Equations (2) and (3), shown as dashed lines in Figure 1.5. The suggested Equation (2) is simple because it is in a quadratic form. If EWT ε is selected according to these equations, the chosen ε is close enough to, but larger than, GEWT, which guarantees a correct counting of the board's size. The lines with asterisks in Figure 1.5 illustrate GEWT with respect to a board size *b*. GEWT is determined experimentally.

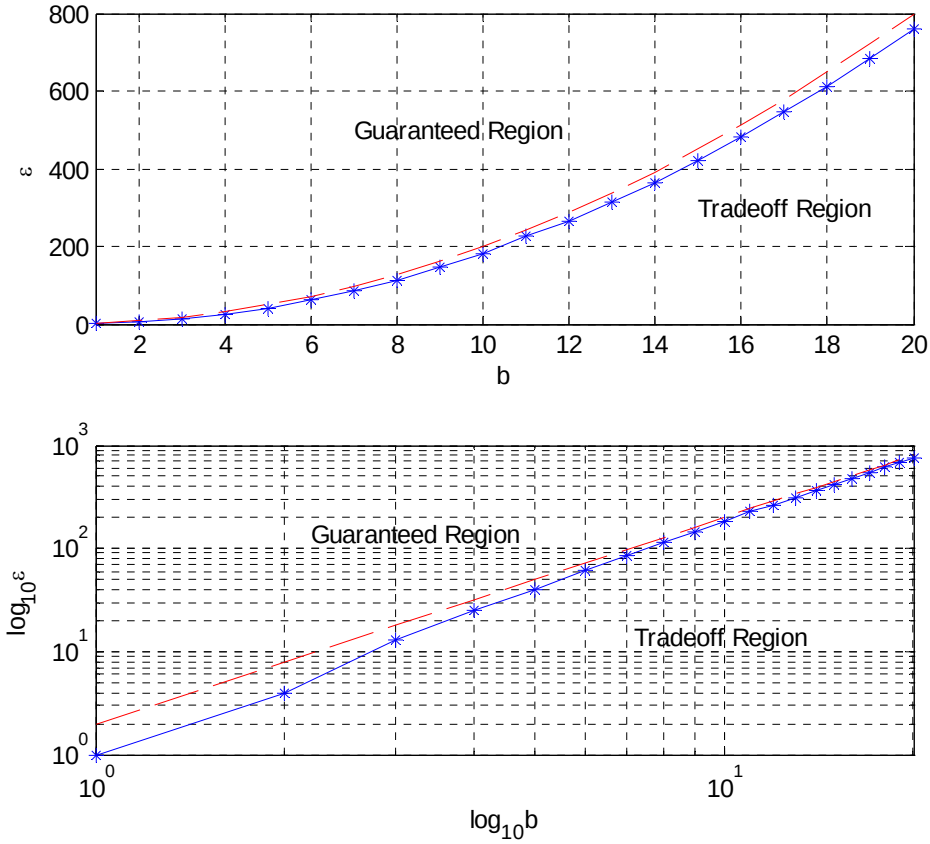


Figure 1.5. Board size vs. the excess waiting time ε in linear and log scale.

$$\varepsilon = 2b^2 \tag{2}$$

$$\log_{10}\varepsilon = \log_{10} 2 + 2\log_{10} b \tag{3}$$

1.5. CONCLUSIONS

The proposed recursive and non-recursive approaches for the group size counting problem are successfully developed and tested. The recursive scheme always returns the correct answer to the given inputs. The non-recursive scheme gives the correct answer when the design parameter excess waiting time ε is larger than the guaranteed excess

waiting time ε_g . Various sizes of Go boards are used, and both of the proposed schemes scale well and work for all the tested sizes.

1.6. REFERENCES

[1] A charity match for Haiti victims Sedol Lee vs. Changho Lee. (2010, Feb.) [Online]. http://open.cyberoro.com/gibo/201002/100211-it-Lee.C.H.sgf.

[2] Xindi Cai and Donald C. Wunsch II, "Computer Go: A Grand Challenge to AI," Challenges for Computational Intelligence, Heidelberg: Springer Berlin, vol. 63, pp. 443-465, May 2007.

[3] Feng-Hsiung Hsu, "Cracking Go," IEEE Spectrum, vol. 44, no. 10, pp. 50-55, September 2007. [Online]. http://spectrum.ieee.org/computing/software/cracking-go/0.

[4] Roman Ilin, Robert Kozma, and Paul J. Werbos, "Beyond Feedforward Models Trained by Backpropagation: A Practical Training Tool for a More Efficient Universal Approximator," IEEE Trans. on Neural Networks, vol. 19, no. 6, Jun. 2008.

[5] Janice Kim and Jeong Soo-Hyun, Learn to Play Go, I-V, Second printing ed. Denver: Good Move Press, 2005.

[6] Jan Krikke, "The Challenge of Go," IEEE Intelligent Systems, vol. 22, no. 6, pp. 11-18, Nov./Dec. 2007.

[7] Byung-Doo Lee and Soo-Hyun Jeong, Computer Go. Seoul: ITMedia, 2008.

[8] Martin Müller, "Computer Go," Artificial Intelligence, vol. 134, no. 1-2, pp. 145-179, Jan. 2002.

[9] Bruno R. Preiss, Data Structures and Algorithms with Object-Oriented Design Patterns in C++: John Wiley & Sons, Aug. 1998.

[10] Erik C.D. van der Werf, Mark H.M.Winands, H. Jaap van den Herik, and Jos W.H.M. Uiterwijk, "Learning to predict life and death from Go game records," Information Sciences: An International Journal, vol. 175, no. 4, pp. 258-272, Nov. 2005.

[11] Paul J. Werbos and Xiaozhong Pang, "Generalized maze navigation: SRN critics solve what feedforward or Hebbian nets cannot," in Proc. of IEEE International Conference on Systems, Man, and Cybernetics, Beijing, China, Oct. 14-17, 1996, pp. 1764-1769.

[12] Donald C. Wunsch II, "The Cellular Simultaneous Recurrent Network Adaptive Critic Design for the Generalized Maze Problem Has a Simple Closed-Form Solution," in Proc. of the IEEE-INNS-ENNS International Joint Conference on Neural Networks IJCNN 2000, Como, Italy, July 24-27, 2000, pp. 79-82.

[13] Changhyuk You, Introduction to life-death problem for beginners 1-3. Seoul: Dasan Books, Mar. 2003, (in Korean).

2. ROBOTIC GO: EXPLORING A DIFFERENT PERSPECTIVE ON HUMAN-COMPUTER INTERACTION WITH THE GAME OF GO

ABSTRACT

The advent of computers and the World Wide Web diversified the way in which the game of Go is played. While traditional human-to-human play still remains an important form of game play, amateur players, along with some professional players, have shifted the play domain from “off-line” club houses to “on-line” Go servers. Computer Go is an important field of study to develop a software to play Go or a Go engine. In addition to human-to-human play, a Go engine or computer intelligence to play Go adds another axis to play configuration: human-to-computer play and computer-to-computer play. These revolutions in the game of Go happened in an extremely short period of time compared to the history of the game, which is more than 4,000 years. We summarize this unavoidable change for the first time in the literature, to our knowledge, and propose a novel way to interact with the current technological advances. We present the new Human-Machine-Computer-Network Interface concept and our implementation of the machine interface with a robot arm. This Lynxmotion robotic arm named Cheonsoo-I successfully places stones on a board under the proposed architecture.

2.1. INTRODUCTION

The entertainment industry is an established market, and the entertainment robot is at its dawn and growing quickly. The day will come when robots play an inevitably important role in the entertainment industry. However, only about two decades have passed since the traditional way of playing a board game face-to-face has been

diversified. The advent of computers and the World Wide Web (WWW) revolutionized the conventional human-to-human player pattern.

The game of Go [1] is a two-player game especially popular in East Asia and is gaining more popularity in other regions. The game is called Baduk in South Korea and Weiqi in China; its history spans more than 4,000 years. The popularity of Go in the East exceeds or is comparable to that of Chess in the West. In the meantime, the Artificial/Computational Intelligence society has been paying increasingly more attention to Go. Now, computer Go [2, 3] is considered to be a new, unconquered challenge in these fields similar to when IBM's computer Chess program Deep Blue defeated human Chess master Garry Kasparov in 1997.

While Go has received a great deal of attention, an extensive literature search reveals no paper publications about Go robotics. Therefore, the literature survey should be stated in the context of entertainment robotics. Dautenhahn et. al. in [4] provide a solution to an intellectual question regarding what constitutes a robot companion. Their survey results show that a majority of people favor the idea of a robot companion. The current work was performed based on the hypothesis of a favored robot companion. In [5], Goodrich and Schultz review human-robot interaction and discuss it as a growing field of research and application requiring interdisciplinary understanding. They explore human-computer interaction, artificial intelligence and cybernetics as important fields of study. The experiences of the current authors support their view on the multi-disciplinary nature of human-robot interaction. While unique challenges exist in Go robotics, one of the technical challenges for the authors was gathering knowledge and experience across multiple fields of study. That is, a background regarding computer Go, socket

programming, image processing, and robotics was prerequisite. Very few papers exist in the literature about board game robots. They mostly are limited to Chess-playing robots [6,7,8], probably because of the game's popularity.

The authors make several contributions through this chapter. The first and most important is that they are the first, to their knowledge, to contribute research on Go robots and to articulate the human-computer interface in the existing literature. Another contribution is the proposed architecture interfacing Go Text Protocol (GTP) [9]. GTP is the *de facto* standard for open-source computer Go engines to interface with a Go GUI program that displays a Go board and stones.

This chapter is organized as follows. Section 2.2 overviews Go and computer Go. Section 2.3 presents the high-level view of the human-computer-network interface for Go, and then the architecture is proposed in Section 2.4. Section 2.5 summarizes the current study's implementation of a Go robot based on the proposed architecture. Section 2.6 concludes the chapter, and the following sections present acknowledgments and references.

2.2. GAME OF GO AND GO SOFTWARE

2.2.1. Game of Go: Basic Game Rules. The basic rules of Go are simple. Players place stones to capture territories, which are intersections that remain empty at the end of the game. Figure 2.1 shows two top professional players discussing the territories on the board. The placed stones form a group with adjacent stones, and they remain on the board unless they are captured. Adjacency is defined only in parallel or vertically along the lines, not diagonally. In Figure 2.2, the stones on the top row form groups. On the right

corner, eleven stones construct a group. Conversely, none of the stones on the second row are grouped because all of them are located diagonally.

Figure 2.1. Final at the 27th KBS Baduk-wangJeon (Battle of the Baduk King), Changho Lee vs. Sedol Lee, Mar. 16, 2009, South Korea [14].

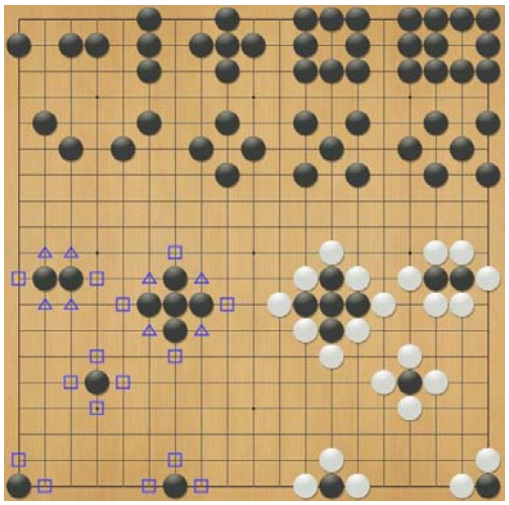


Figure 2.2. Artificially placed stones to explain adjacency (top) and liberty (bottom) concepts.

A group of stones (including a group of one stone) is captured when the last *liberty* of a group is removed, i.e., surrounded fully by the opponent's stones. A liberty is

defined as an empty adjacent intersection of an occupied stone position. On the bottom left of Figure 2.2, a liberty is marked with a square or a triangle. A triangle is used to emphasize that the corresponding liberty is from an occupied stone located either vertically or in parallel, but not diagonally. The black stones on the bottom right should be removed as captured stones because all the liberties are taken by the opponent, or white stones. Another basic rule is the Ko rule to avoid an infinite loop of recurring board status. One cannot place a stone that causes the board's status from the previous play to be repeated.

2.2.2. Categories of Go Software. The term *Go software* [10] is so polysemous that the accurate meaning should be clarified before further discussion. Firstly, computer Go is a field of study that aims to create a computer program that plays Go. Its ultimate goal to date is to defeat the human champion. Such a program is referred to as a Go engine, computer Go program, or computer Go intelligence. Some famous Go engines are The Many Faces of Go, MoGo, GNU Go, Silver Star, and Fuego.

Other categories of Go software of interest are the Go client program and the Go editing program. The Go client program connects a Go server with a pool of Go players. It follows the client-server model and allows one player to play against another from the pool, as illustrated in Figure 2.3. The Go editing program, more specifically the Go GUI (Graphic User Interface) program, provides a GUI that displays the board's status. One can also store and edit game records with it. For example, Figure 2.2 shows a screen capture of the editing program GoGui. Typically, a well-designed Go client also provides the editing feature.

Figure 2.3. Go client program: (a) the client-server concept that overcomes spatial constraints to connect Go players around the world, (b) list of connected players at Cyberoro, a server recognized by the Korea Baduk Association.

Here, two new terminologies are defined: *Go Interface Software (GIS)* and *Go Interface Protocol (GIP)*. GIS refers to a computer program that serves as a front end to a human player. For example, both the Go client program and editing program are GIS with different features. GIP is a protocol used by GIS to communicate with a Go Engine. GIP sets a group of standard commands/procedures so that GIS can exchange information about game plays with a Go engine. The *de facto* standards are the Go Modem Protocol (GMP) and the newer GTP.

2.3. BIG PICTURE: PLAYER CONFIGURATIONS AND THE INTERFACE ISSUES

GIS plays a central role in interconnecting human and computer players over the network. The interface issues can be best explained with block diagrams. Figure 2.5 summarizes the existing player configurations and the corresponding interface issues. Additionally, Figure 2.4 illustrates the unified block diagram that explains how humans, computers, and networks interconnect. Note that both figures are coherent; Figure 2.4 unifies the existing structures in Figure 2.5.

In general, a human player can choose to play against another human player or a computer player (Go Engine); the opponent's location can be local or global. A global opponent is one who is located across the network. Given that the opponent is chosen to be a local human player, the interface between them can be a traditional board and stones, represented in Figure 2.5.a, or a computer, represented in Figure 2.5.b. Typical input and output devices for the latter are a mouse and computer monitor.

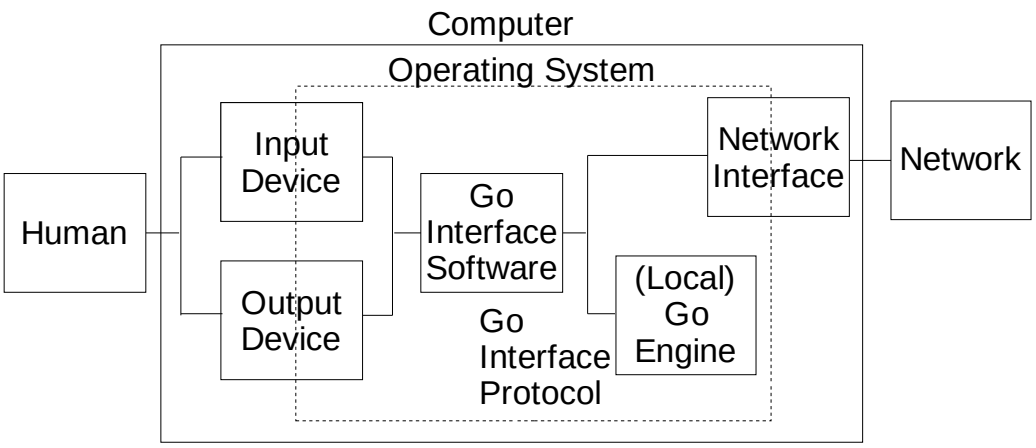


Figure 2.4. Block diagram of unified structure for human–computer-network interface.

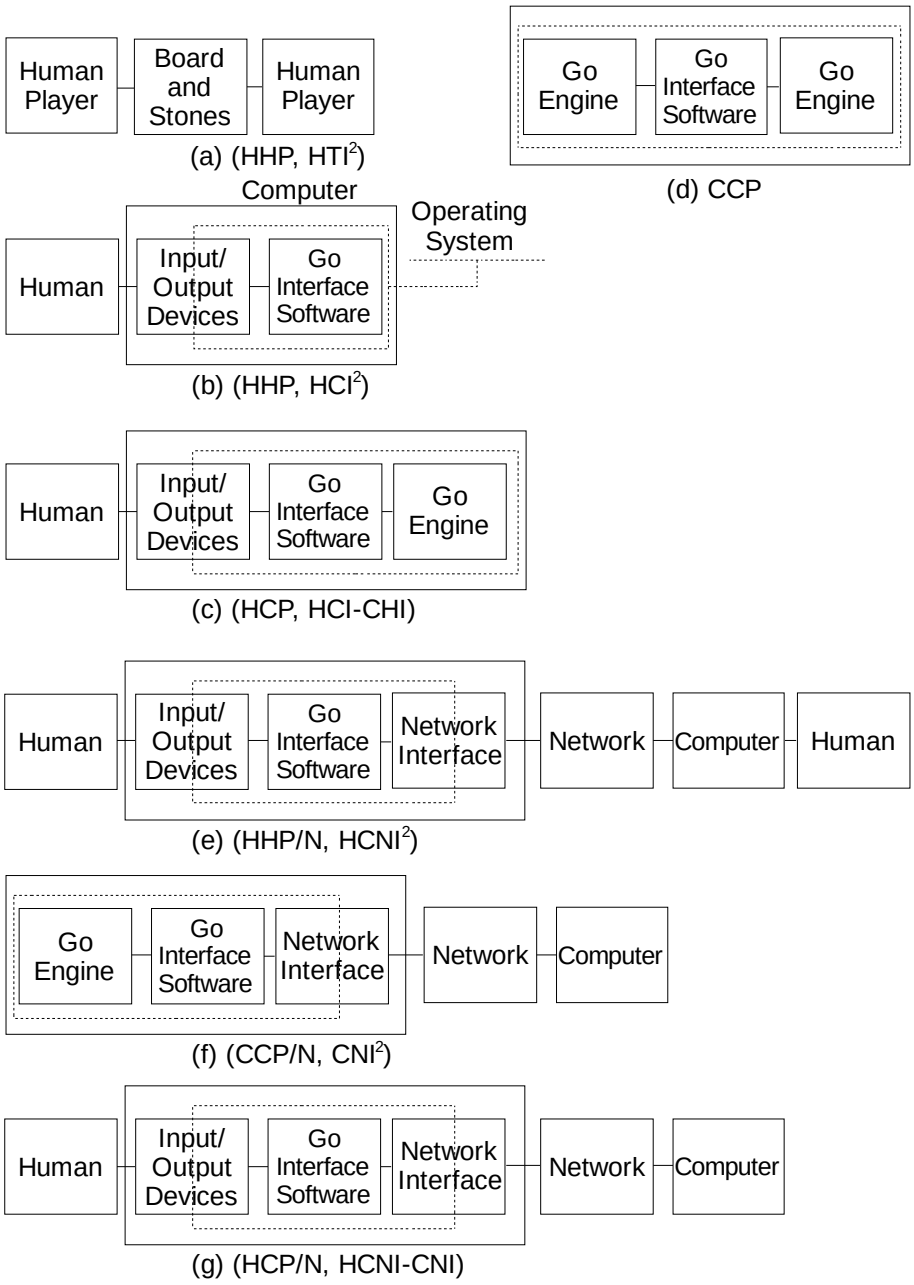


Figure 2.5. Player configurations and the corresponding interface issues: (a) HHP (Human-to-Human Player) with HTI (Human-Traditional equipment Interface), (b) HHP with HCI (Human-Computer Interface), (c) HCP (Human-to-Computer Player) with HCI-CHI (Computer-Human Interface), (d) CCP (Computer-to-Computer Player), (e) HHP/N (HHP over the Network) with HCNI (Human-Computer-Network Interface)-HCNI, abbreviated as $HCNI^2$, (f) CCP/N (CCP over the Network) with CNI (Computer-Network Interface)-CNI, abbreviated as CNI^2, (g) HCP/N (HCP over the Network) with HCNI-CNI. The notation is (Player configuration, Interface of a player [left] – Interface of a player [right]). If interfaces for both players are identical, a square abbreviates the notation, e.g., HCI^2 instead of HCI-HCI.

Human-human physical interaction with a board and stones will be discussed further in the following section. Another popular configuration is to play against other human players over the network, denoted as (HHP/N, $HCNI^2$). Typically, the counterpart computer is a Go server, so GIS serves as a Go client program. In a Go program, an Internet socket serves as the universal gateway to the network interface, as well.

The structure of (HCP, HCI-CHI) evolved from the monolithic structure in [11], in which GIS and the Go engine are not modularized. While some programs still maintain the monolithic structure, it is recommended that GIS and Go engine be modularized separately because there is a *de facto* standard protocol to communicate between them. Another HCP configuration is depicted in Figure 2.5.g, (HCP/N, HCNI-CNI). This is a setting to play against a global computer player. On the other hand, CCP (Computer-to-Computer Player) is becoming more important as the development of a strong Go engine becomes an important research topic. Figures 2.5.d and 2.5.f present the two existing settings. The former is a local test that sets matches between two Go engines. A tool such as *twogtp* that interfaces GTPs of two Go engines corresponds to GIS. The latter is the structure for a global test to a computer Go server (CGOS), which automatically organizes matches between Go engines. CGOS mandates using a custom client program and implementing GTP as the GIP.

In summary, Figure 2.4 and Figure 2.5 are tabulated in Table 2.1 and Table 2.2, respectively. The former summarizes the existing interface issues under different player configurations, and the latter provides a framework to unify Figure. All the player configurations and interfaces are consolidated nicely in Figure. Go Interface Software selects the input among the Go engine, network interface, and input/output devices.

Table 2.1. Summary of play configuration and interfaces.

	Human	Computer
Human	(HHP, HTI^2), (HHP, HCI^2) (HHP/N, $HCNI^2$)	(HCP, HCI-CHI) (HCP/N, HCNI-CNI)
Computer	Reciprocal to HCP	CCP (CCP/N, CNI^2)

Table 2.2. GIS's counterpart block for different opponents.

	Player	
	Human	Computer
Local	Input/Output Devices	Local Go Engine
Global	Network Interface	Network Interface

2.4. PROPOSED ARCHITECTURE FOR HUMAN-MACHINE-COMPUTER INTERFACE

2.4.1. Dilemma of a Contemporary Player: Traditional Board and Stones or Online Go Server? Contemporary players suffer from a dilemma regarding how best to enjoy a Go game. One can choose HHP or HHP/N. Choosing between a human or computer opponent currently is not a dilemma because the computer is much weaker than a human. An intermediate player can easily defeat, generally speaking, most Go engines available on Go servers.

A unique feature of this game is that the sound of placing a stone plays an important role in the enjoyment of Go. A beginner learns how to hold a stone properly in order to make a pleasant sound when it is placed on the board. Figure 2.6.a shows the proper technique for holding a stone between the index and middle fingers. The stone should be smashed down by the middle finger. The impact of the sound is considered to

reflect the player's level and experience. Figure 2.6.b shows a world top-level professional, Sedol Lee (9 Dan), placing a stone with curvy fingers and wrist to magnify the stone placement sound. This small gesture reflects his experience and skill in this game.

The dilemma occurs because of interface issues. One advantage of HHP is the stone placement sound and the physical feel of the stone, which increase the pleasure of playing Go. The disadvantage is the limited range of opponents caused by space-time constraints. On the other hand, players on the HHP/N setting typically use a mouse and computer monitor as the input/output devices. Its obvious advantage is a large pool of opponents that overcome the space-time constraints of HHP. The drawback is the absence of the pleasant tactile and auditory sensations. Most players prefer (HHP, HTI^2) over (HHP, HCI^2) due to the unstimulating nature of a mouse and computer monitor.

(a) (b)

Figure 2.6. Stone placement.
(a) proper way to hold a stone [15],
(b) Sedol Lee's stone placement exposing his experience in the game.

2.4.2. Playing on a Go Server Inevitably Decays Amateur Club Houses. A club house, which serves as a pool of players, is a part of Go culture. As its domain is shifted online, the amateur club house culture decays. A traditional club house compensates somewhat for the space-time constraints of HHP. It is referred to as kiwon in Korean, ki-in in Japanese, and qiyuan in Chinese, which also may refer to a nationwide association. National associations such as HangukKiwon (Korea Baduk Association), Nihon Ki-in (Japan Go Association), and ZhongguoQiyuan (Chinese Weiqi Association) operate club houses for professional players and supporting organizations.

In the arena of amateurs, the club house culture is declining. The popularity of Go servers directly impacts the industry. For example, once-popular club houses in South Korea are disappearing. Playing Go online is now an unavoidable social phenomenon.

2.4.3. Proposed Architecture to Solve a Go Player's Dilemma. Figure 2.7 depicts the proposed architecture to solve a Go player's dilemma of choosing between traditional equipment and a Go server. The key idea is to use a robot or machine as a frontend to play a game for the opponent. Two additional blocks are added to the

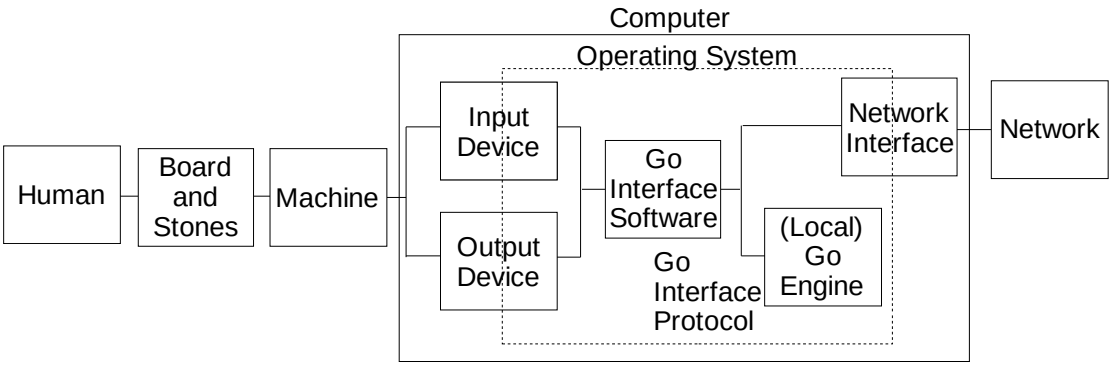


Figure 2.7. The proposed Human-Machine-Computer-Network Interface architecture.

structure shown in Figure 2.4 to create the structure shown in Figure 2.7. The traditional equipment consisting of a board and stones is placed between the human user on the left and the machine. Figure 2.8 details the unified structure in Figure 2.7, illustrating the possible player configurations and the corresponding interface issues. They all correspond nicely to the existing structures explained in Figure 2.7.

2.5. IMPLEMENTATION OF A GO ROBOT: CHEONSOO-I

The implementation of a Go robot in this study successfully places stones on the custom-made 9x9 board. The stones have a diameter of 19mm and are 6.5mm thick. The computer's interaction with the physical board is separated into two independent functions: identifying where the pieces are located on the board and when the human player has made a move, and manipulating stones during the robot's turn. The first task is passive and is accomplished by using a standard webcam, Logitech QuickCam for Notebooks, positioned above the board as shown in Figure 2.9.b. The images are processed with Roborealm software, RoboRealm v.1.8.22.1 [11], in order to determine both the position of the playing board in the image and which intersections contain stones. The second task is accomplished using the Lynxmotion robotic arm, Lynx 6 Robotic Arm Combo Kit for PC [12], with a modified gripper attachment to move stones.

2.5.1. Vision System and Image Processing. Roborealm is used to sample and process images from the webcam in order to determine the board's state. Red markers located at the corners of the board are used to crop and transform the image so that the board's intersections occur at known locations. Then, the image is processed using basic threshold and blob filtering tools, as shown in Figure 2.10, to isolate each stone. After the image is filtered, each blob's center of gravity is compared with the intersection locations

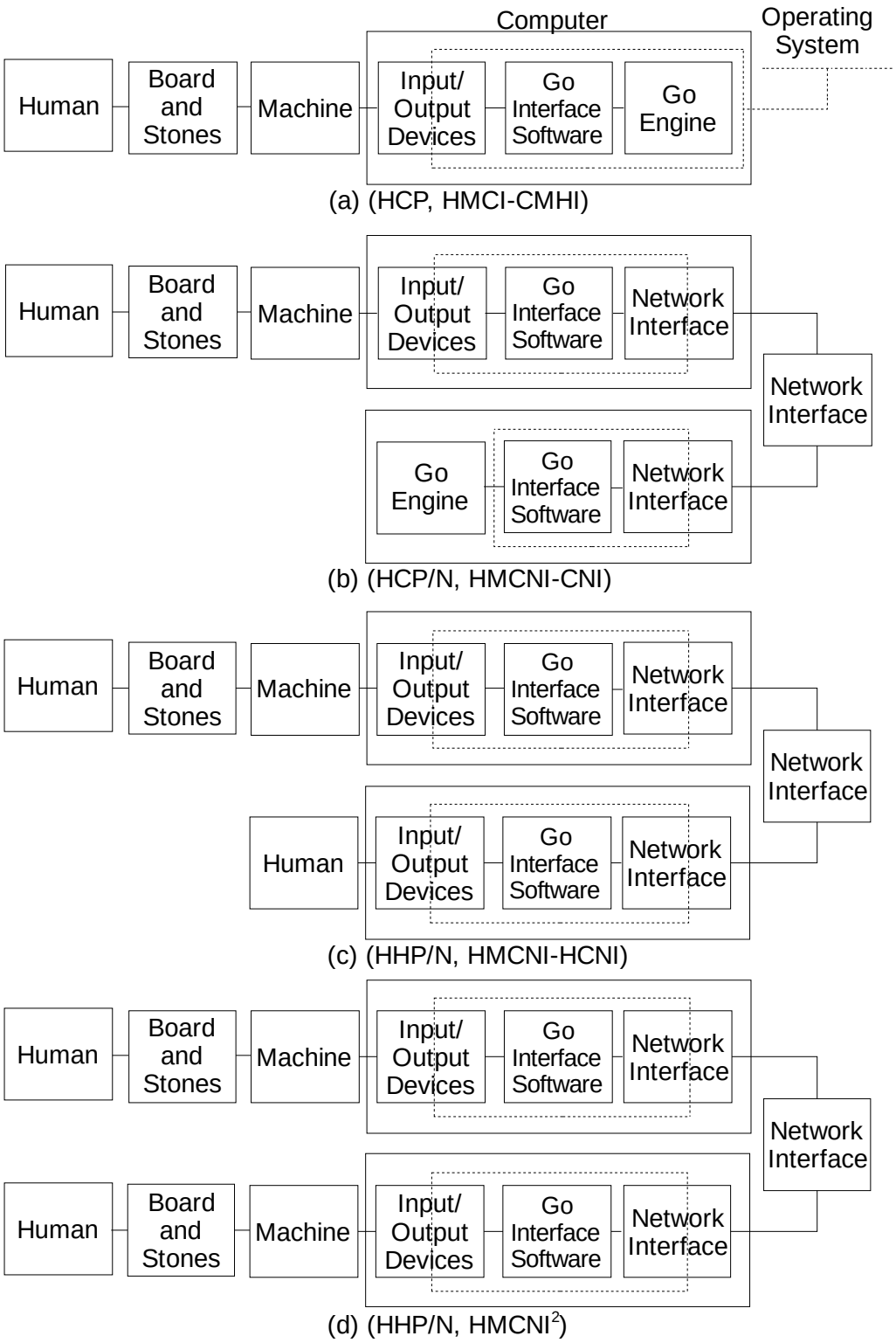
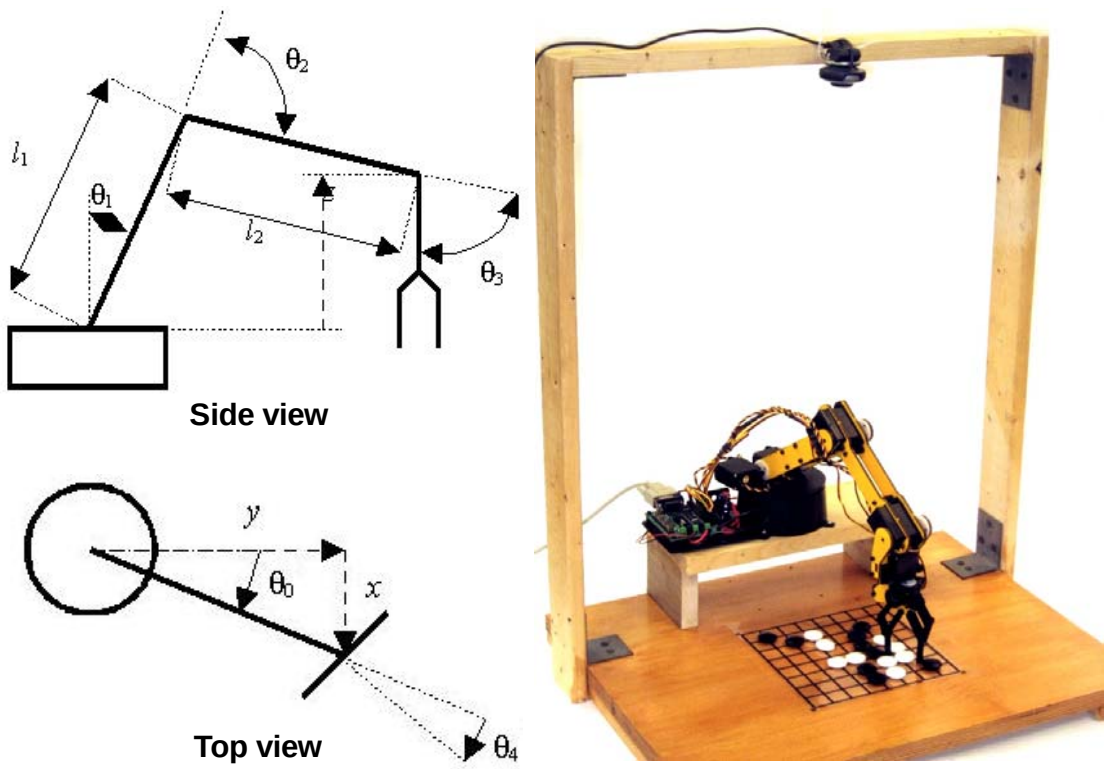


Figure 2.8. Player configurations and interface issues.

and is marked with the closest intersection. If the stone is not within the area of the playing board, it is marked as being off the board, and its coordinates are stored. If motion is detected in the images (by comparing several consecutive frames), the image processing sequence is suspended until the image stabilizes. The board's state then can be compared to the state held in the memory to determine if the human player has made a move.



(a) (b)

Figure 2.9. Cheonsoo-I.
(a) Ideal model of robotic arm,
(b) Hardware implementation of the machine input/output interface.

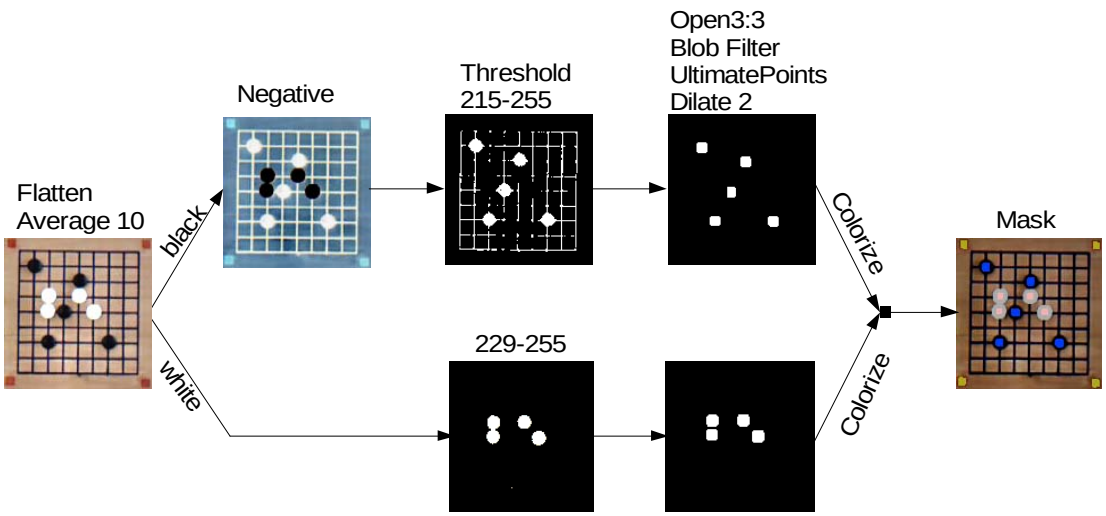


Figure 2.10. Example of an image processing sequence.

2.5.2. Stone Manipulation with Robotic Arm. The Lynxmotion robotic arm has six degrees of freedom and is controlled by the Roborealm software through the Lynxmotion SSC-32 servo controller board. If the robotic arm is modeled as an ideal mechanism, as represented in Figure 2.9.a, the Cartesian coordinates of the "wrist," along with the angle of the hand from a vertical position, can be transformed into the necessary angles for each servo using the following equations:

$$\theta_0 = \tan^{-1}\left(\frac{x}{y}\right) \tag{1}$$

$$\theta_1 = \frac{\pi}{2} - \tan^{-1}\left(\frac{z}{\sqrt{x^2+y^2}}\right) - \cos^{-1}\left(\frac{l_1^2 + x^2 + y^2 + z^2 - l_2^2}{2l_1\sqrt{x^2+y^2+z^2}}\right) \tag{2}$$

$$\theta_2 = \cos^{-1}\left(\frac{x^2+y^2+z^2-l_1^2-l_2^2}{2l_1l_2}\right) \tag{3}$$

$$\theta_3 = \pi - \theta_1 - \theta_2 \tag{4}$$

$$\theta_4 = \theta_1 + \frac{\pi}{4} \tag{5}$$

In order to compensate for the non-ideal characteristics of the actual arm caused by backlash in the servo gears and gravity-induced sag, the Cartesian coordinate input necessary for each intersection was found manually. Roborealm accepts input specifying the board positions at which a stone should be placed, and the necessary coordinates are looked up and used to calculate the servo angles.

The stones used for playing Go are difficult for a two fingered machine to manipulate due to their rounded shape, smooth surface and low profile. The standard rubber tips of the gripper mechanism were not able to pick up the stones reliably. Through experimentation, a successful design was achieved through a wire attachment to the tips of the gripper mechanism.

Figure 2.11 shows the configurations that were tested during the process of designing the modified tips. The original robotic grippers with rubber tips, shown in Figure 2.11.a, were not able to pick up the stones due to the stones' low profile. When the rubber tips were removed, as shown in Figure 2.11.b, the stones could be picked up; however, a very slight error in positioning the tips would cause the stone to slide out from between the grippers, which could catastrophically wreck the arrangement of the stones while playing an actual game. Figure 2.11.c shows a four-pronged wire attachment to the grippers that proved somewhat successful, but it still required fairly precise positioning. The last design, shown in Figure 2.11.d, proved to be successful in consistently picking up the stones, even under positioning errors of up to one quarter of the distance between grid intersections.

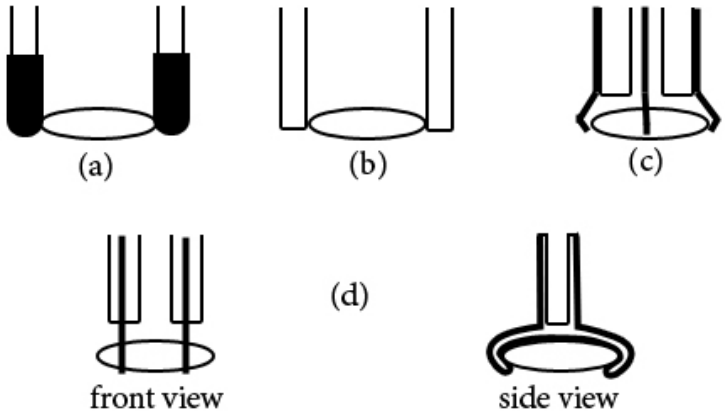


Figure 2.11. Gripper tips for Cheonsoo-I.
(a) unmodified with standard rubber tips,
(b) unmodified without rubber tips,
(c) four-pronged wire attachment,
(d) final design.

2.6. CONCLUSIONS

This chapter presented two major contributions. First, it introduced a unified architecture that explains the existing player configurations and interface issues. Secondly and more importantly, it proposed a unified Human-Machine-Computer-Network Interface implemented with a Lynxmotion robot arm.

Regarding the first contribution, the existing player configurations and interface issues were summarized, and a unified structure was introduced to explain the current settings. In this process, two new terminologies were defined: Go Interface Software and Go Interface Protocol. The former refers to any software that interfaces with the existing components. For example, Go software, such as the Go client program, Go editing program, and twogtp, fall nicely into the category of Go Interface Software. The latter, Go Interface Protocol, is defined as a protocol used by Go Interface Software to communicate with a Go engine. It is a group of standard commands/procedures that

allows Go Interface Software to exchange information about game plays with a Go engine, and the current *de facto* standard is Go Text Protocol.

The second contribution can be reinterpreted as a solution to “a Go player’s dilemma.” This dilemma has existed since a computer was first used to play a Go game. Modern players must choose between the pleasantness of the traditional human interface, i.e., a Go board and stones, and ease of access to a pool of other Go players or a Go server by using a computer’s “unstimulating” input/output devices. The proposed Human-Machine-Computer-Network interface concept, along with a “Go-playing robot,” addresses this issue.

The proposed prototype robot arm, Cheonsoo-I, successfully places stones on a 9x9 board. The implementation requires a multi-disciplinary background in computer Go, socket programming, image processing, and robotics, as well as an extensive amount of time to experiment with the proper shape of a robot arm’s tip to stably pick up and place stones of slightly varied shapes.

2.7. REFERENCES

[1] Soo-Hyun Jeong, Understanding the Contemporary Baduk. Seoul, South Korea: Nanampublisher, 2004 [in Korean].

[2] Martin Müller, "Computer Go," Artificial Intelligence, vol. 134, no. 1-2, pp. 145-179, Jan. 2002.

[3] Bruno Bouzy and Tristan Cazenave, "Computer Go: An AI Oriented Survey," Artificial Intelligence, vol. 132, pp. 39-103, 2001.

[4] Kerstin Dautenhahn et al., "What is a Robot Companion–Friend, Assistant or Butler?," in Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems, Aug.2-6, 2005, pp. 1192-1197.

[5] Michael A. Goodrich and Alan C. Schultz, "Human-Robot interaction: A Survey," Journal Foundations and Trends in Human-Computer Interaction, vol. 1, no. 3, Feb. 2007.

[6] G.A. der Boer, A. van Inge, R. Stam F.C.A. Groen, "A chess playing robot: Lab course in robot sensor integration," IEEE Transactions on Instrumentation and Measurement, vol. 41, no. 6, pp. 911-914, Dec. 1992.

[7] Shuying Zhao, Chao Chen, Chunjiang Liu, and Meng Liu, "Algorithm of location of chess-robot system based on computer vision," in Proc. of Control and Decision Conference, Yantai, China, Jul.2-4, 2008, pp. 5215-5218.

[8] Emir Sokic and Melita Ahic-Djokic, "Simple Computer Vision System for Chess Playing Robot Manipulator as a Project-based Learning Example," in Proc. of IEEE International Symposium on Signal Processing and Information Technology, Sarajevo, Dec.16-19, 2008, pp. 75-79.

[9] GTP - Go Text Protocol. [Online]. http://www.lysator.liu.se/~gunnar/gtp/.

[10] Sensei's library. [Online]. http://senseis.xmp.net/.

[11] Albert L. Zobrist, "Feature Extraction and Representation for Pattern Recognition and the Game of Go," University of Wisconsin, Ph.D. dissertation 1970.

[12] Roborealm–Robotic Machine Vision Software. [Online]. http://www.roborealm.com/.

[13] Lynxmotion Robot Kit. [Online]. http://www.lynxmotion.com/.

[14] NEWSis. [Online] http://www.newsis.com.
Photo provided by Tygem. [Online]. http://www.tygem.com.

[15] The Nihon Ki-in. [Online] http://www.nihonkiin.or.jp.

3. RECONFIGURABLE DISRUPTION TOLERANT ROUTING VIA REINFORCEMENT LEARNING

ABSTRACT

This paper shows packet delivery rate can be improved by adopting learning-based hybrid routing strategies when a wired network suffers from severe link disruption. The dynamics of the link disruptions complicate the routing problem; successful and stable routing operations of conventional routing approaches are hindered as the level of disruption increases. The target is to develop a robust and efficient routing approach in a single structure. A robust routing approach means a packet should be delivered to a destination even under severe disruptions. Efficient routing should deliver a packet with the shortest path at no disruption. These goals should be achieved with the maximum utilization of preexisting network components and with the minimal human intervention once installed. Therefore, we chose a popular conventional routing scheme, Link State, and add-ons that can learn changing network environment. Our approach is to add a learning agent and a simple routing scheme to Link State in order to automatically select a better routing scheme at an arbitrary level of disruption. Markov Decision Process is employed to model this problem. The simulation results show robustness and packet delivery rate are increased up to 35% at acceptable cost of computational and architectural complexity even when Link State approach is close to be collapsed.

3.1. INTRODUCTION

The Internet is an essential part of society. With its increased significance, unstable operations of (wired) networks have increasingly large socioeconomic impacts, and unexpected network failures could even lead to catastrophe. For example, a 7.1

magnitude earthquake off the coast of Taiwan on December 27, 2006, exposed the Internet's vulnerability to disruption. Most South-East Asian network traffic was compromised, and the backbone network suffered from low Internet bandwidth availability [1]. Lessons from this "choke" point, or single loop, encourage the creation of more robust "mesh" networks.

A literature review shows that no research has been conducted thus far, to the best of the authors' knowledge, directly related to the proposed approach. A few papers in the arena of fault-tolerant networking share some common elements. Bolding and Yost [2] emphasize the importance of a fault-tolerant design for multicomputer networks. Doley et. al. [3] model a fault-tolerant network as a graph in which a node represents a processor and an edge represents a communication link. They provide a mathematical background relevant to the current work in that a network can be modeled as a graph. However, their simplified assumptions about the link disruption behavior do not sufficiently explain the complicated disruptions in the current work. On the other hand, the literature on disruption/delay tolerant networks [4] is irrelevant to the results of this study even if the goals may share common points. The disruptions in disruption/delay tolerant networking are caused by long delays that the TCP/IP architecture cannot tolerate. Conversely, it is assumed that the TCP/IP architecture and disruptions are limited to link disruptions.

Reinforcement Learning (RL) [5] is employed as an optimization technique to select between two standalone routing schemes. A Markov Decision Process (MDP) [6] is used to formulate an RL problem. RL techniques can be classified as associative learning or non-associative learning. The associative property in RL means that actions

are associated with particular situations. More specifically, the action that should be selected is associated with the current state. The assumption is that the selected action will affect the next state in a possibly non-deterministic but predictable way. The approach used here lies somewhere between associative and non-associative learning. The basic philosophy of this approach is to select the routing scheme (action) that maximizes the expected packet delivery rate (value) for the given network status (state) in this sequential decision problem. However, due to the actions of other agents in the network, the true state of the environment is only partially observable, and the next state is not predictable.

The Link State protocol family is a popular routing scheme on the Internet. It finds the shortest path between a source and a destination with Dijkstra's algorithm [7]. Inherently, Link State is susceptible to frequent link disruptions that are widespread throughout the network. A throughput drop can be caused by these “soft” disruptions, and the entire network can be paralyzed because Link State breaks down at a certain level of disruptions.

Another protocol, Gossip [8], is a probabilistic flooding routing protocol. It spreads a received packet to all of the neighboring nodes except the original sender of the packet. The decision regarding transmission is determined probabilistically. This protocol is light-weight but inefficient under preferable network conditions because it “floods” the network with duplicated packets.

The authors are the first, to their knowledge, to model this problem with MDP and to propose a solution to make the popular routing scheme more robust to link disruptions at a high frequency.

This chapter is organized as follows. MDP and RL are overviewed briefly in Section 3.2. The proposed MDP formulation and RL model are presented in Section 3.3. The simulation environment and results are presented in Sections 3.4 and 3.5, respectively. The conclusions follow in Section 3.6.

3.2. MARKOV DECISION PROCESS AND REINFORCEMENT LEARNING

MDP provides a mathematical background to model a sequential decision problem. An MDP is formulated as a collection of objects: the decision epoch *T*, the action space *A*, the state space *S*, the transition probability function $p_t(s_{t+1} \mid s_t, a_t)$, and the reward function $r_t(s_t, a_t)$, where s_t and a_t represent an element of state and action space at time *t*, respectively. An MDP's goal is to find the best policy under the given optimality criteria by solving the Bellman equation in (1).

$$\begin{aligned} V^{\pi}(s) &= V^{\pi}(s_t) = E_{\pi}\{R_t \mid s_t = s\} \\ &= \sum_{a_t \in A} q_t(s_t, a_t)\left[\sum_{s_{t+1} \in S} p_t(s_{t+1} \mid s_t, a_t)\{r_t(s_t, a_t) + \gamma V^{\pi}(s_{t+1})\}\right] \end{aligned} \tag{1}$$

where $q_t(s_t, a_t)$ is the probability of taking action a_t when the current state is s_t. The value function $V^{\pi}(s)$ is defined as the expected total reward (or return) under the expected total reward optimality criterion [6]. Note that π is a policy, *s* is a state, R_t is a reward as a random variable at time *t*, $r_t(s_t, a_t)$ is a scalar reward for s_t and a_t, and Υ is the discount rate between 0 and 1.

The optimal policy π^* can be defined in terms of the value function $V^{\pi}(s)$, i.e.,

$$V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s), \forall s \in S \tag{2}$$

In other words, π^* is any policy π that maximizes the value function $V^{\pi}(s)$ for all states s.

The drawback of MDP is that it requires a complete model of the environment; therefore, it suffers from the "curse of dimensionality." RL often is used to solve MDP problems as it can mitigate the curse of dimensionality. It is a machine learning scheme that learns from interactions with the environment through trial and error. RL approximates the optimal solution to an MDP problem. RL's strength is that it does not require the complete model of the problem. However, developing an RL model can be a complex task if the number of states is large.

Figure 3.1 depicts the interaction between the agent and the environment. The agent learns about the environment by evaluative feedback, rewards r_t, and maintains estimates of the expected total reward, the value function $V^{\pi}(s)$. A policy of action choice is determined in order to maximize the value function, in a statistical sense.

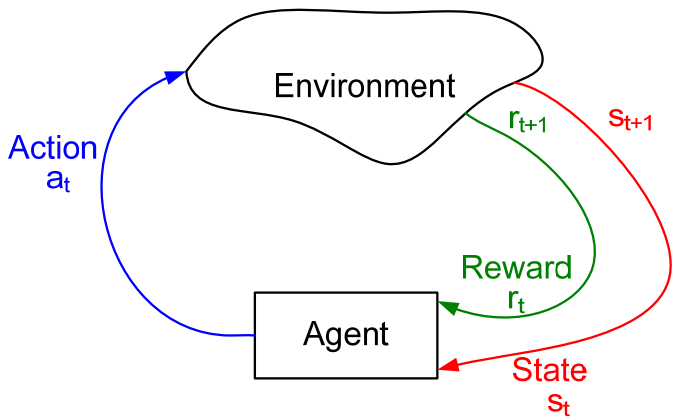


Figure 3.1. Agent-environment interaction in an RL model.

3.3. PROPOSED MARKOV DECISION PROCESS FORMULATION AND REINFORCEMENT LEARNING MODEL

3.3.1. System Architecture of the RL Model. Figure 3.2 is a simplified block diagram depicting the proposed RL model, which is consistent with Figure 3.1. An RL agent resides in a network node, and each node makes an independent decision. A routing agent on the left of Figure 3.2 receives state and reward information, s_t and r_t, and outputs an action a_t. Parts other than the routing agent are considered to be elements of the environment. Nodes contain components other than communication networks, and each node in the network has the same perspective. Namely, each node in a mesh topology (Figure 3.4) possesses its own routing agent, routing strategy selector, knowledge bases (KBs), and reward converter.

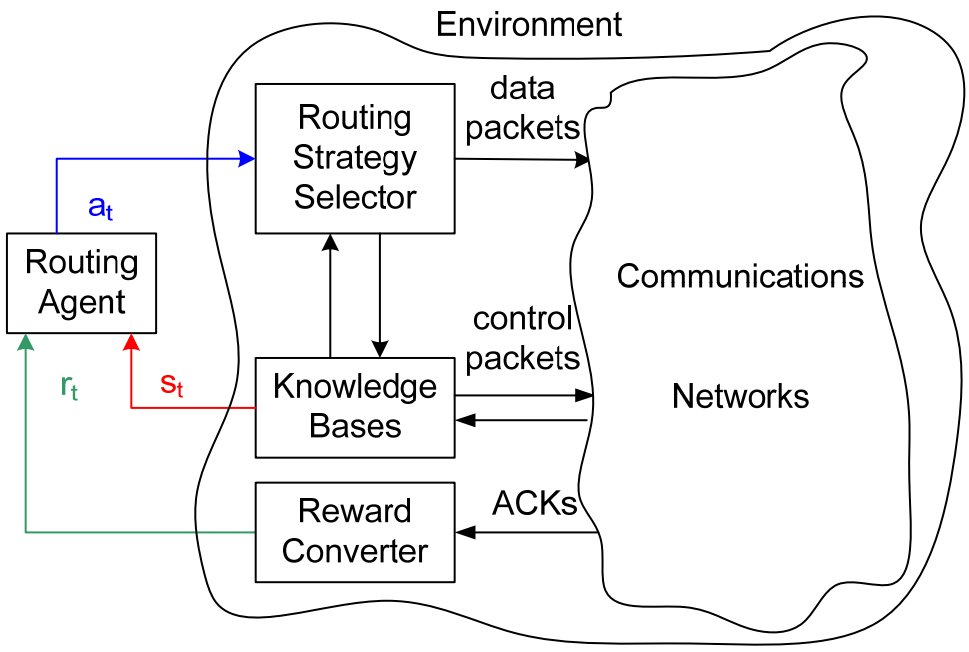


Figure 3.2. Simplified block diagrams of our RL model.

The roles of each component in Figure 3.2 are explained below. The routing agent is an RL agent that makes routing decisions. It maintains the action value function table $Q(s,a)$ and selects a routing strategy based on the value function. The routing strategy selector is a bank of routing schemes, including the Link State (LS) and Gossip (GS) schemes. The KBs (Knowledge Bases) block is a module used to estimate the current network status s_t maintained in the KBs by disseminating control packets. The routing strategy selector has a pool of candidate routing strategies and selects one for each packet based on the action selected by the routing agent. The reward converter converts an acknowledgement packet (ACK) into a reward for the routing agent. It generates a reward of 1 if an acknowledgement is received and 0 otherwise.

3.3.2. Markov Decision Process Formulation. An MDP formulation of this problem is challenging in that the next state does not depend on the choice of current action. In other words, the Markov property is broken. Sutton and Barto [5] note that the Markov property does not have to be maintained in order to apply RL to some real-world problems. Therefore, this problem can be formulated with MDP.

The proposed MDP formulation is explained below. Above all, the state space S, which is a representation of the network, consists of the state space of the Adjacency Knowledge Base (AKB) and the Reachability Knowledge Base (RKB), denoted as S_A and S_R, respectively.

$$S = S_A \cup S_R \tag{3}$$

While the representation's resolution may vary, the state space S is discretized into five binary digits, four bits to the AKB and one bit to the RKB, so that the action-

state value function is not too large. With this in mind, the number of states N_s is set to $2^5 = 64$. The least significant bit signifies S_R whether or not the destination is currently reachable, and the remaining bits reflect the current network reliability of S_A, as derived from information in the AKB.

The adjacency in the AKB is the average link availability $\overline{l_a}$ ranging from 0 to 1.

$$S_A = \{\overline{l_a} : 0 \le \overline{l_a} \le 1\} \tag{4}$$

Reachability in the RKB is a Boolean value indicating whether a path to the destination exists or not. The path to the destination is found by recursively searching the AKB.

$$S_R = \{0, 1\} \tag{5}$$

Figure 3.3 shows an example of a discrete state space S. The average link availability is $0.375 \le \overline{l_a} < 0.4375$, and this node has an end-to-end route to the destination at this particular time.

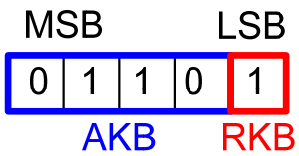


Figure 3.3. An example of the discrete state space *S* of the knowledge bases.

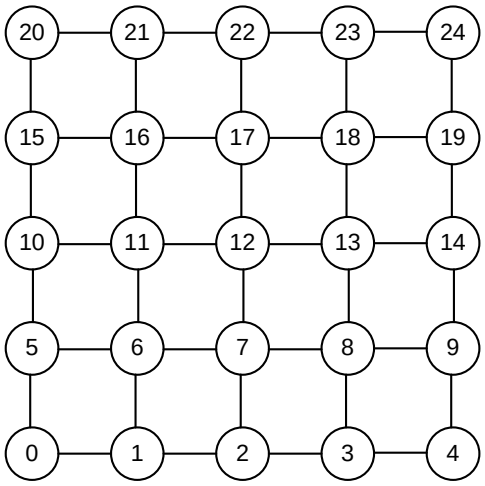


Figure 3.4. 5-by-5 mesh topology with wired links.

An average link's availability is the moving average of its available up-time over an observation time interval. The AKB disseminates control packets called AKB update packets in order to share information about the network. On each node, the AKB and RKB update their neighborhood information by using the information in the AKB update packets and provide an estimate of the current state to the routing agent.

The action space A is simple in terms of MDP formulation. The number of actions N_a is two because two candidate routing strategies are considered, GS and LS.

$$A = \{a_1, a_2\} \text{ where } a_1 = \{Gossip\}, a_2 = \{\text{Link State}\} \tag{6}$$

Both LS and GS are full routing schemes, so their implementation is fairly large.

The reward r_t is based on the ACK packets. This reward structure is consistent with the goal of the routing agent because the state-action value $Q(s,a)$ is an estimate of the probability that the routing protocol can deliver a packet to the destination. This is the

expected percentage of delivered packets. In the current implementation, $Q(s,a)$ is a $2^5 \times 2$ table because the size of the table $Q(s,a)$ is $N_s \times N_a$ .This design is reasonable because more value is attributed to the action that results in a higher number of delivered packets.

3.4. SIMULATION ENVIRONMENT

The major simulation parameters are summarized in Table 3.1. Given the simulation run time, the time required to generate the data packets is 370 seconds. The first 30 seconds are used to exchange control packets, and traffic is not generated during the last 10 seconds.

Table 3.1. Simulation parameters.

Parameters	Values	Remarks
Simulation run time	410 seconds	
Simulation warm-up time	30 seconds	beginning
Simulation idle time	10 seconds	end
Traffic generation rate	240 kbps	CBR
Traffic source-sink	100 pairs	
Packet size	2000 bytes	
Packet Time-To-Live	100 seconds	TTL
Link bandwidth	1 Mbps	
Link delay	10 ms	
Link queue size	100 packets	
Link change interval	10 seconds	In average
Minimum number of hops	4	Source - Sink
Q-learning discount/learning rate	0.9/0.85	

Mesh topology is employed to experiment with disruptions on backbone networks. The number of source-sink (destination) pairs is larger than the number of nodes in the network. A mesh topology was chosen that models backbone networks because the mesh topology has multiple links at the intermediate nodes, and packet congestion can be caused by multiple traffic sources. A single node can host numerous sources and sinks because a backbone network is assumed. Traffic source-sink pairs are uniformly distributed over the network.

Constant-Bit-Rate (CBR) traffic sources generate data packets of 2000 bytes at 240 kbps. This is equivalent to 15 packets per second. A packet's time-to-live (TTL) is set to 100 seconds in order to ensure that no packet is dropped because of its TTL. One-hundred pairs of traffic source-sinks are assigned randomly to nodes with the restriction that the source-sink pairs are separated by the minimum number of hops N_h. N_h is important because it affects the packet delivery performance. In an extreme case in which $N_h = 1$, Link State and Gossip perform similarly because both schemes can deliver packets to the destination with about the same performance. The probability of a link being down between the source and the destination is proportional to N_h because there is a greater chance of packet delivery failure. For simulations, each source is set to be separated at least 4 hops from its corresponding sink, i.e., $N_h=4$. UDP is used because of its small overhead.

The links go up and down dynamically to model disruptions in the network. The uniform link disruption model [10] is used. The ratio between a link's up interval and down interval is defined as network availability. These up/down intervals specify the

mean of the random distribution, defining the time that the links go up and down, respectively.

A custom network simulator, ADPNetsim [9], was developed, which is modeled after a popular network simulator, NS-2 [10]. ADPNetSim resembles NS-2 in its Links State implementation, job scheduler, trace file format, etc. The key difference is that ADPNetsim implements an RL algorithm, Q-learning [5], and can choose between two routing schemes on a per-packet basis.

3.5. SIMULATION RESULTS

The simulation results in this section show that the “on-node” local learning agent improves the packet delivery rate at all levels of disruption at an acceptable cost of computational and architectural complexity presented in previous sections. Additionally, the proposed approach adds more disruption resistance over Link State. Gossip chosen by the learning agent at a severe disruption level is more robust in nature than Link State because it sends out duplications of the received packet to all the neighboring nodes.

Figure 3.5 compares the performance of local learning to that of standalone Link State and Gossip under the given simulation environment, showing the average packet delivery ratio with respect to network availability. Network availability on the x-axis indicates the degree of disruption. Network availability 1 implies no disruption, while 0 implies that all the links are down continuously during simulation runs. The packet delivery rate on the y-axis is a ratio between the number of delivered packets and the total number of sent packets. Thirty-two simulation runs at a given network availability are averaged to obtain the resulting graph. Note that conventional network performance metrics such as delay, jitter, and packet drop with quality of service in mind are less

meaningful as disruption becomes more severe. Rather, the packet delivery rate gains importance as the chance for delivery failure increases.

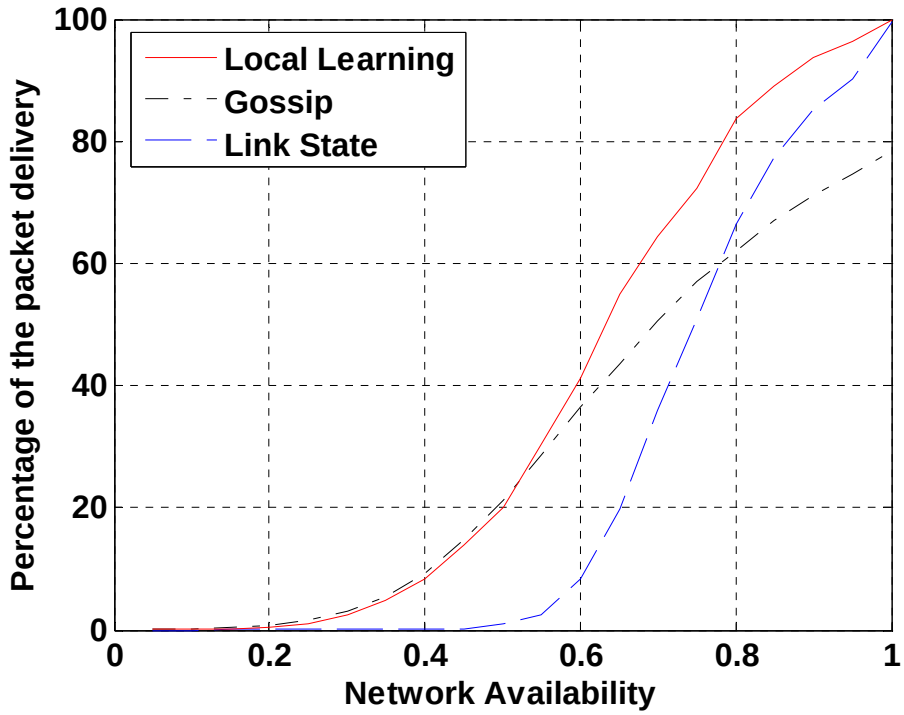


Figure 3.5. Comparison of average packet delivery rates in a 5-by-5 mesh topology: Local learning, Link state and Gossip.

When there is no disruption (network availability 1.0), all learning agents learn to select Link State. One can visualize that all the nodes in the mesh topology are marked as Link State at all times during simulation runs. When there is severe disruption (network availability between 0.2 and 0.55), Gossip is selected. Therefore, all the nodes are now marked as Gossip at all times.

Performance under mild disruption (network availability between 0.75 and 1.0) is worth noting. The delivery rate of the local learning case is higher than that of either standalone scheme. It was expected that the delivery rate graph would overlap that of Link State under no/mild disruption and that of Gossip under severe disruption. A closer look at the intermediate behaviors reveals that the learning agents learn to mix Gossip with Link State. The agents select Link State when the neighboring links are up most of the time and switch to Gossip when they experience more disruptions in terms of the moving average. At a specific time, nodes in favorable conditions tend to choose Link State, while those in more disrupted regions tend to select Gossip, which helps to increase the packet delivery rate compared to the case with a standalone Link State. One can visualize this desirable feature as some nodes in the mesh topology being marked as Link State and the others as Gossip at one time point, and then these marks changing dynamically as the "contaminated regions" vary over time. In effect, the distributed/on-node learning agent is reconfigurable not only to network availability in the general sense, but also spatially within the network.

The use of the learning agent makes routing under disruption more robust than in the case in which only Link State is used. Link State breaks down at a network availability of 0.5 as opposed to Gossip at 0.2. The advantage of using the learning agent is that it learns to select a more robust scheme (Gossip) under severe link disruption and a more efficient scheme (Link State) under mild link disruption.

Figure 3.6 depicts the learning agent's relative improvement over Link State and Gossip. It shows that using a learning agent improves the packet delivery rate by up to 35% over Link State at a network availability of 0.65.

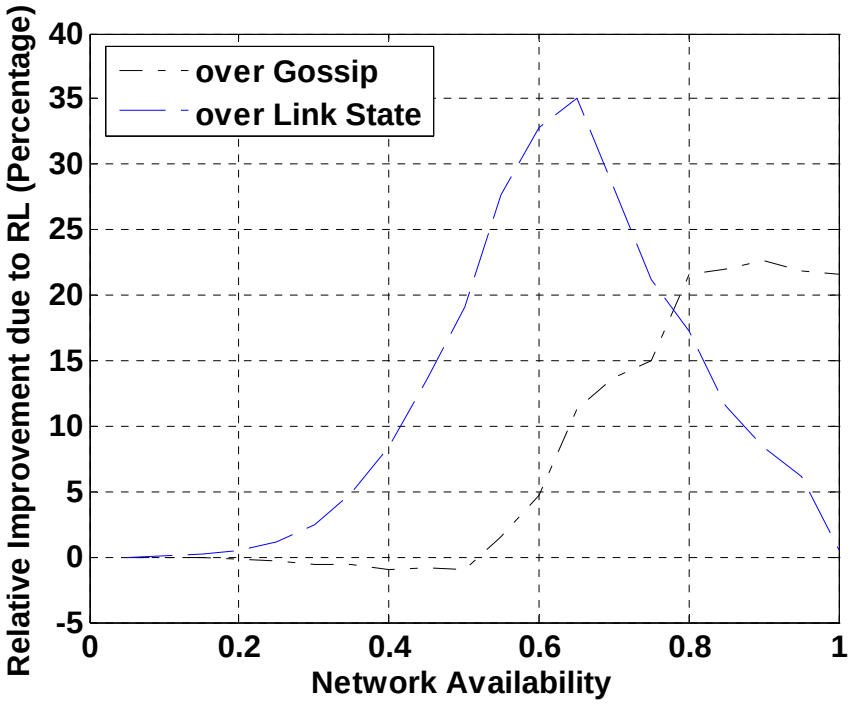


Figure 3.6. Relative performance improvement by on-node local learning.

3.6. CONCLUSIONS

The contribution of this chapter is that the authors are the first, to the best of their knowledge, to model this problem with MDP and to suggest a relatively light-weight solution to provide robustness to existing wired networks against a challenging and important defect in the existing popular routing scheme. The inherent defect in Link State is the packet delivery rate; throughput deteriorates as the level of disruption increases, and, more seriously, the scheme collapses under severe disruptions. One solution is to employ an additional routing scheme that can tolerate a level of disruption that Link State cannot. A flooding-based routing scheme satisfies this purpose, but the shortcoming of this approach is lower performance under no/mild disruption. The proposed solution is to

mount both schemes and a learning agent on a network node in order to configure the node to choose a better scheme by learning from its surroundings. The selection is reconfigurable depending on different levels of disruption and node locations in the network.

The proposed custom network simulator, ADPNetsim, modeled after a popular network simulator, NS-2, simulates a 5-by-5 mesh topology with wired links. The minimum number of hops between the traffic source and sink is set to 4 in order to prevent cases that can disguise increases in the packet delivery rate. An extreme example is a 1-hop traffic source-sink pair that a packet can traverse whenever a single link is available. The link disruption model is uniformly distributed over time given a duty cycle. Network availability is inversely proportional to the level of disruption. The MDP formulation is as follows. States represent the moving average of link availability and ability to reach the destination. Available actions are Gossip, Link State, or none. The reward is 1 if a packet is delivered and 0 otherwise. Learning is performed in a distributed manner on each node in the network.

The simulation results show that the local learning agent on each node improves the packet delivery rate by selecting a better scheme. Under no disruption (network availability of 1.0), all learning agents learn to select Link State. Under severe disruption (network availability between 0.2 and 0.55), Gossip is selected. Packet delivery rates under mild disruption (network availability between 0.75 and 1) warrant attention. The delivery rate is higher than in either standalone scheme. The delivery rate graph was expected to overlap the Link State graph under no/mild disruption and the Gossip graph under severe disruption. A closer look at the internal behaviors reveals that the learning

agents learn to mix Gossip with Link State. The agents select Link State when the neighboring links are up most of the time and switches to Gossip when they experience more disruptions in terms of the moving average. At a specific time point, nodes in favorable conditions tend to choose Link State, while those in more disrupted regions tend to select Gossip, which helps to increase the packet delivery rate compared to the case with a standalone Link Sate. This is a desirable feature that makes the proposed "distributed" local learning approach reconfigurable. In effect, it is reconfigurable not only to network availability in the general sense, but also spatially within the network.

Additionally, the proposed approach resists disruptions better than Link State. Gossip, naturally, is more robust than Link State under severe disruption. Gossip breaks down at a network availability of 0.2 in the current experiments, as opposed to 0.5 for Link State. Learning agents learn to switch to a more robust scheme, i.e., Gossip.

The proposed approach to amalgamate a light-weight and robust Gossip to the popular Link State achieved success by employing an ADP learning agent. The packet delivery rate and robustness can be enhanced under various levels of link disruptions at an acceptable cost of computational and architectural complexity to mask the Link State's inherent defect.

3.7. REFERENCES

[1] United Nations Economic And Social Council, "Consideration of Information, Communication and Space Technology Issues, Globalization and Information, Communication and Space Technology: Current Issues and Future Directions in the Asian and Pacific Region," Economic and Social Commission for Asia and the Pacific, Committee on Managing Globalization, Fourth session, Part II, Bangkok, Thailand, November 19-21, 2007.

[2] K. Bolding and W. Yost, “Design of a Router for Fault-Tolerant Networks,” In Proceedings of the 1994 Parallel Computer Routing and Communication Workshop, May 1994, pp. 226-240.

[3] D. Dolev, J. Y. Halpern, B. Simons, H. R. Strong, “A NEW LOOK AT FAULT TOLERANT NETWORK ROUTING,” Information and Computation, Vol. 72, Issue 3, pp. 180-196, March 1987.

[4] The Delay Tolerant Network Research Group, Available: http://www.dtnrg.org.

[5] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction", MIT Press, 1998.

[6] M. L Puterman, “Markov Decision Processes: Discrete Stochastic Dynamic Programming,“ John Wiley & Sons, Inc, New York, 1994.

[7] A. S. Tanenbaum, “Computer Networks,” fourth edition, Prentice Hall, 2002.

[8] A. Allavena, A. Demers, J. E. Hopcroft, "Correctness of a Gossip-based Membership Protocol," Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing, pp. 292-301, Las Vegas, NV, USA, 2005.

[9] ADP Network Simulator. [Online]. Available: http://web.mst.edu/~tk424/code_release/ADPNetSim/.

[10] The network simulator. [Online]. Available: http://www.isi.edu/nsnam/ns.

3. CONCLUSIONS

This dissertation presented a study of spatial-temporal reasoning applications in computational intelligence. Three problems concerning computer Go and one problem concerning the computer network were solved. More specifically, the group size counting problem, a sub-problem in computer Go, was solved with both recursive and non-recursive solutions. The non-recursive solutions were solved under the same constraints with CSRN. This is called collective cooperative learning. The second problem was to learn a game tree of a 2x2 Go board. The CSRN structure was modified and trained with cellular PSO. The game tree for black's optimal move was learned with zero system error. The third problem concerned Go robotics. A unified system structure for a novel human-machine interface was proposed and successfully implemented with a prototype robot arm, Cheonsoo-I. The last problem concerned a routing problem in a computer network. A hypothetical case was considered in which the links in a computer network were disrupted. This problem was formulated with MDP, and the proposed reinforcement learning scheme improved the overall packet delivery rate.

APPENDIX A.
GO GAME RULES

Hypothesis (Optimal moves in Go are unknown): Optimal moves in Go are unknown or have not been solved yet. Even though some computer Go researchers claim that they have solved Go for 6x6 and 9x9 boards, even human experts do not know the optimal moves for the regular Go board size. Therefore, the optimal solution for Go at any board size has not been solved yet.

Definition 1 (Play point as a vertex): A space on which a Go stone can be placed is called a play point. In graph theory, a play point in Go is a vertex v in a finite set V.

Definition 2 (Go board as an unordered graph): A Go board is a grid made of vertical and horizontal lines. An intersection of a vertical line and a horizontal line forms a play point. Therefore, a Go board consists of a set of play points. In graph theory, a line connecting two play points or two vertices (u,v), where u,v in V and $u \neq v$, forms an edge. Therefore, a Go board is an undirected graph G with the edge set E consisting of unordered pairs of vertices V.

Corollary 1 (2x2 board): Let two vertical and horizontal lines on a 2x2 board form four intersections or play spaces. Then, a 2x2 board is a graph G_2 with four vertices and four edges, i.e., $|V|=4$ and $|E|=4$.

Definition 3 (Occupied play point vs. vacant play point): Only one stone can occupy a play point at a time. We call a play point with a stone an occupied play point, while a point without a stone is a vacant play point. From graph theory's perspective, we name the former an occupied vertex and the latter a vacant vertex.

Definition 4 (Adjacency of a stone): An adjacency of a stone is a play point, regardless of its occupancy, located either vertically or horizontally from the occupied play point of the stone. A play point in the diagonal location is not adjacent.

Definition 5 (Liberty of a stone): A liberty is a vacant play point adjacent to a stone.

Proposition 1 (Liberty removal): When the opponent's stone is placed on the vacant play point in Definition 5, a liberty of a stone is removed

Proposition 2 (Addition of a liberty): When the player's stone is placed at an adjacency (Definition 4) of an existing stone, a liberty or liberties are added to the existing stone on the board.

Definition 6 (Group of stones): Stones of the same color adjacent to each other form a group. In graph theory, a group on a Go board is a sub-graph G' of a graph G.

Corollary 2: A group is a superset of a stone. By convention, a group may consist of one or multiple stones.

Corollary 3 (Liberties of a 2x2 board): Assuming a 2x2 board (Corollary 1), a group (Definition 6) can have one or two liberties. Then, the maximum edge size is two, i.e., $max\{E(G_2)\}=2$.

Proof. Figure 2.1.d. shows the graph representation of the representative board statuses in Figure 2.1.c. These representatives include all possible board statuses that can be rotated and flipped. Additionally, stone colors can be switched from black to white and vice versa. Therefore, Figure 2.1.d.1 (the leftmost board status with a single black stone) represents eight different board statuses. Figure 2.1.e illustrates the possible liberties. No stone can have more than two liberties. Each stone is a vertex in the graph. Thus, the maximum edge size in G_2 is two.

Definition 7 (A vacant play point as a territory): The significance of a play point lies in its potential to become a territory at the end of a game. A vacant play point secured

by a player is counted as a territory at the end of the game. According to territory rules, e.g. Japanese and Korean rules, the game's score is determined by territories.

Definition 8 (A play point as an area): According to area rules, e.g., Chinese rules, an area includes a territory (Definition 7) and a stone used to secure the territory. In other words, a play point secured by a player is counted as an area at the end of the game. Using area rules, the game's score is determined by the total size of the areas.

Corollary 4: An area is a superset of a territory.

Definition 9 (Eye): An eye is a play point or play points that the opponent can neither play on nor force to be filled.

Definition 10 (Death of a group): A group is dead when it is not alive [25]. A group is alive if it has at least one liberty or two (true) eyes.

Corollary 5: The only representative board status that is always alive on the 2x2 board occurs when two stones of the same color are located diagonally and no other stone exists on the board.

Proof. Refer to Definition 9 and Figure 2.1.d.4. The figure is self-explanatory.

Definition 11 (Neutral point): A neutral point is a play point that is neither player's territory nor has prospects of becoming a territory. At the end of a game, the vacant play points in Figure 2.1.d.2 become neutral points when a game ends in this board status.

Definition 12 (Pass): A pass is a move by a player that does not place a stone on a play point. It causes the opponent to gain the right to play the next move.

Definition 13 (Two consecutive passes): A game is over when both players pass consecutively or when both players agree to finish the game.

Lemma 1: A player cannot make an eye on a 2x2 board when both players play optimally because of the rule to play alternately. Two eyes can be made if and only if the opponent does not play optimally.

Lemma 2: When both players make optimal moves and each player takes each corner, there is no reason to continue the game with two neutral points.

Proof. When both players make optimal moves, the game infinitely returns to Figure 2.1.d.2. There is no reason to continue the game. Therefore, it is legitimate to end the game when two neutral points exist on the board.

Corollary 6: Both players form no territory when playing optimally, and the game is over.

Corollary 7 (Pass on a 2x2 board): Passing on the first move of a game does not affect the game's result if and only if both players play optimally. However, passing on the second move of the game may result in losing the game if the opponent gains two eyes by playing diagonally from the first move.

Definition 14 (Suicide move): A stone commits suicide when it is placed on a play point with no liberty but this move does not remove the last liberty of the opponent's group, resulting in the group's capture. A suicide move is illegal.

Definition 15 (Compensation): Compensation is the points given to white to cancel out black's advantage of taking the first move when an even game ends.

Theorem 1: The compensation for white on a 2x2 Go board is zero.

Proof. Compensation is the points given to white to cancel out black's advantage of taking the first move when an even game ends (Definition 15). A player cannot make an eye on a 2x2 board when both players play optimally because of the rule to play

alternately (Lemma 1). When both players make optimal moves and each player takes each corner, there is no reason to continue the game with two neutral points (Lemma 2). Neither player forms a territory when playing optimally, and the game is over (Corollary 6). Therefore, white does not need compensation.

APPENDIX B.

2X2 GO GAME TREE

2x2 Game Tree for One of the First Four Moves

This game tree shows ¼ of the entire game tree from black's first move placed on the left-bottom corner. In other words, this is the sub-tree starting from this location denoted T_0.

The whole game tree can be implied from this because the sub-trees of other moves by black are rotation invariant. Sub-trees of each position are denoted as T_1 through T_3.

The following simplifying assumptions are made.
Assumption 1. Both players never pass unless there is no place to play.
Assumption 2. A recurring board status marked with a circle reverts to the previous board status in the tree, which is identical and marked with *.

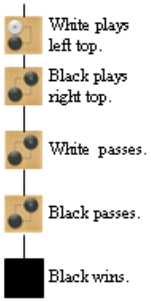
A standard concerning when to pass is necessary. A move is passed when it is judged that placing a stone is disadvantageous. Komi is an issue.

Legend

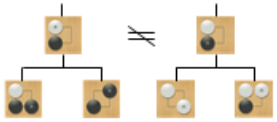
- Pass
  Two consecutive board statuses imply a pass. See an example on the right.
- Terminal state □
  Two consecutive passes ends a game.

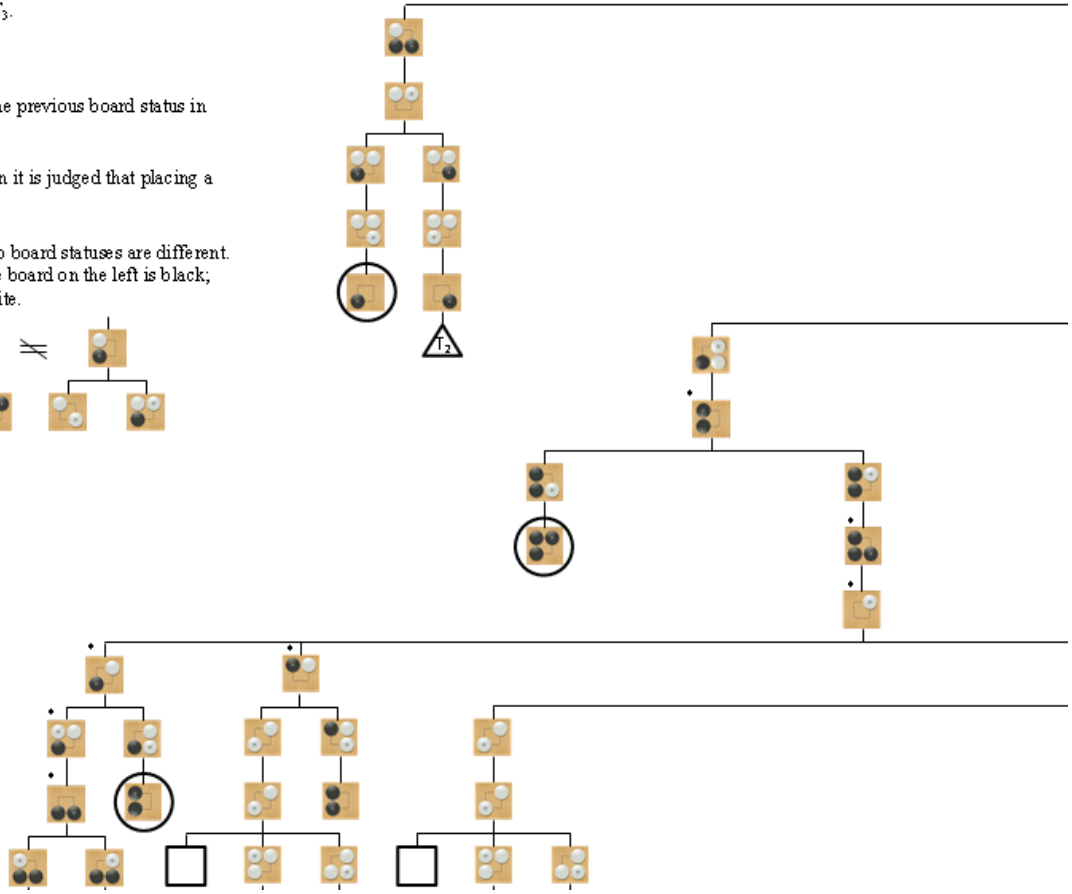
  Black wins ■ White wins □
- Recurring state ○
  Revert the identical parent board statue
- Sub-tree △
  A sub-tree for one of the starting moves T_n

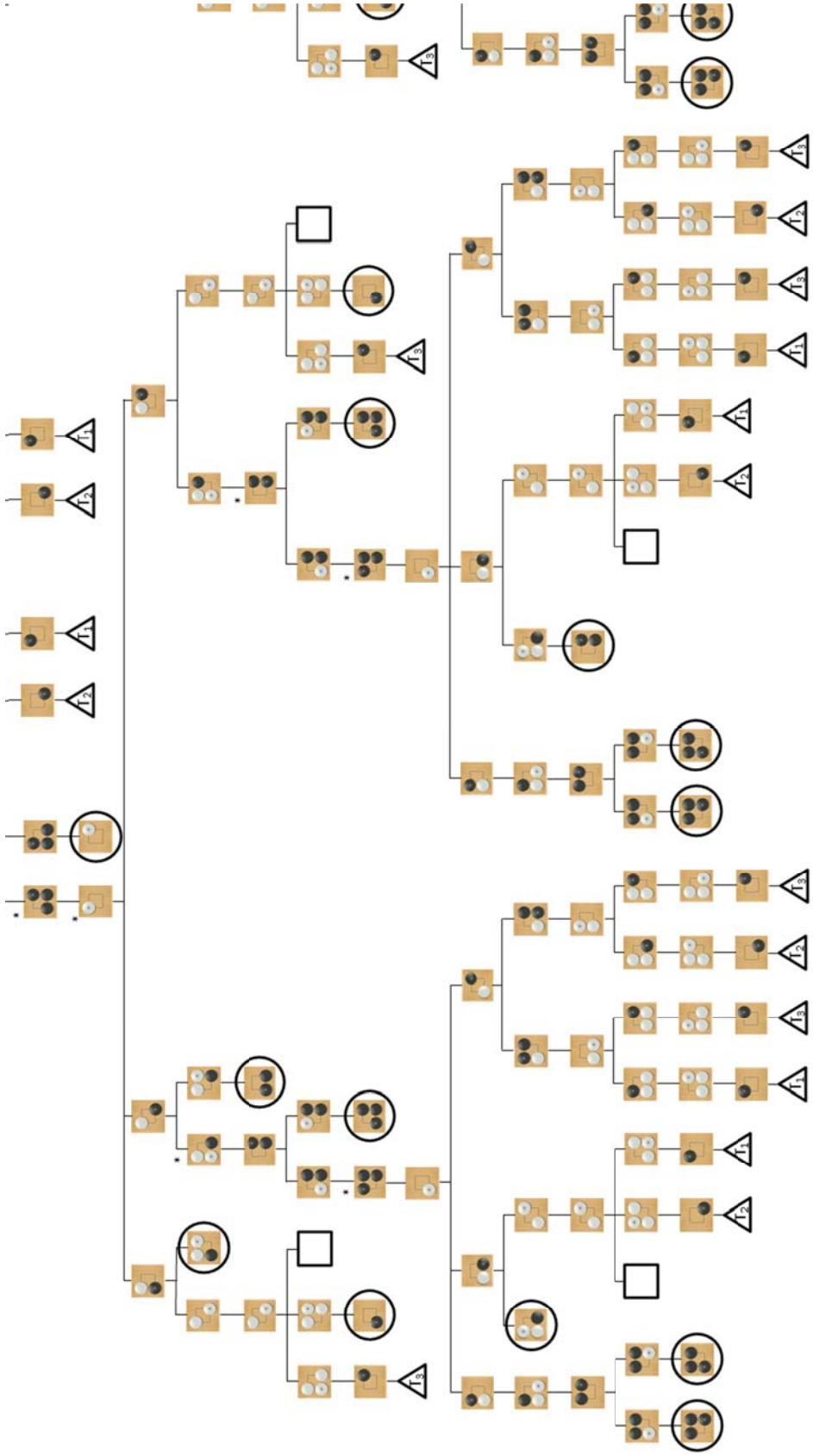
Example

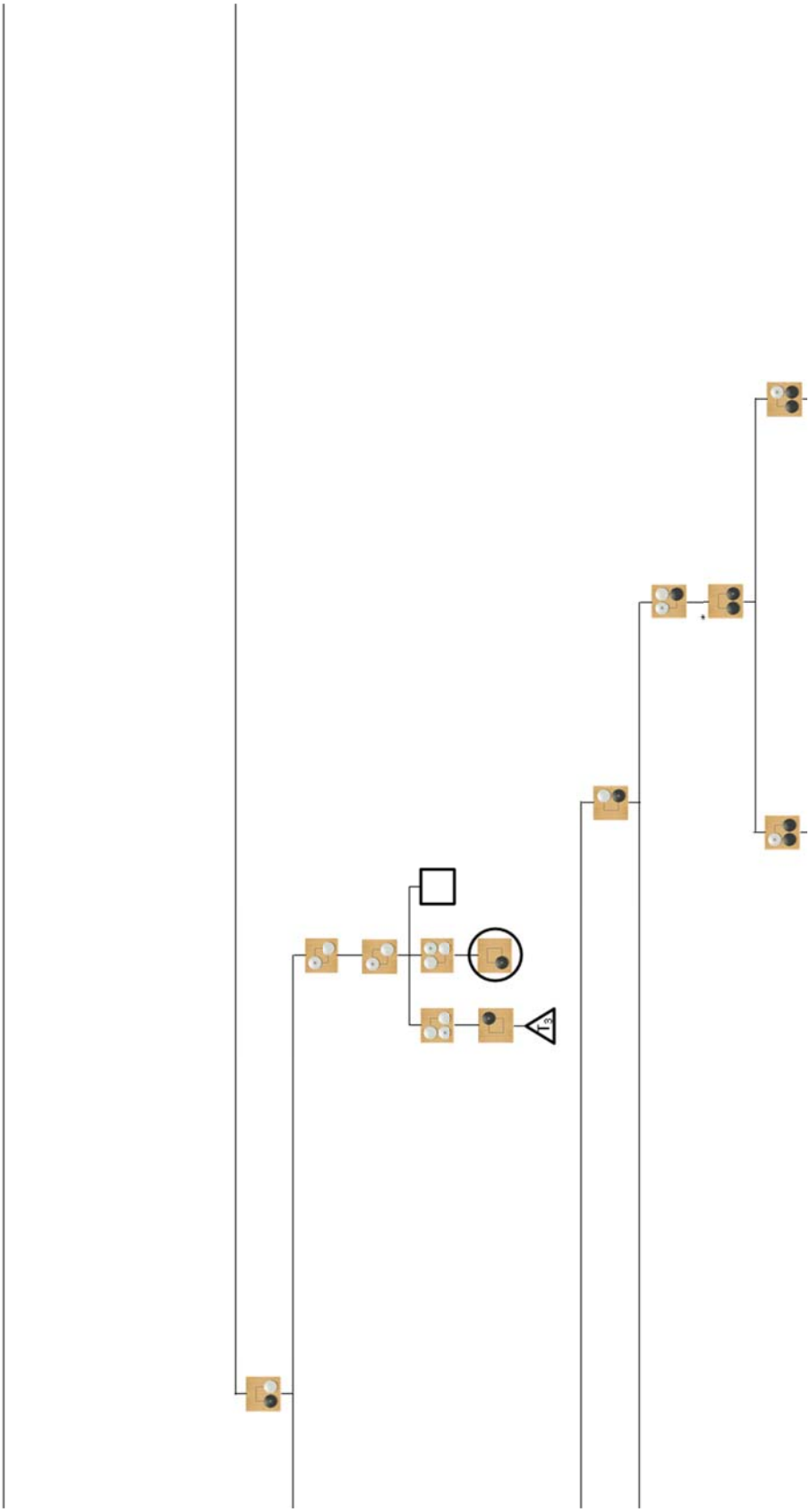


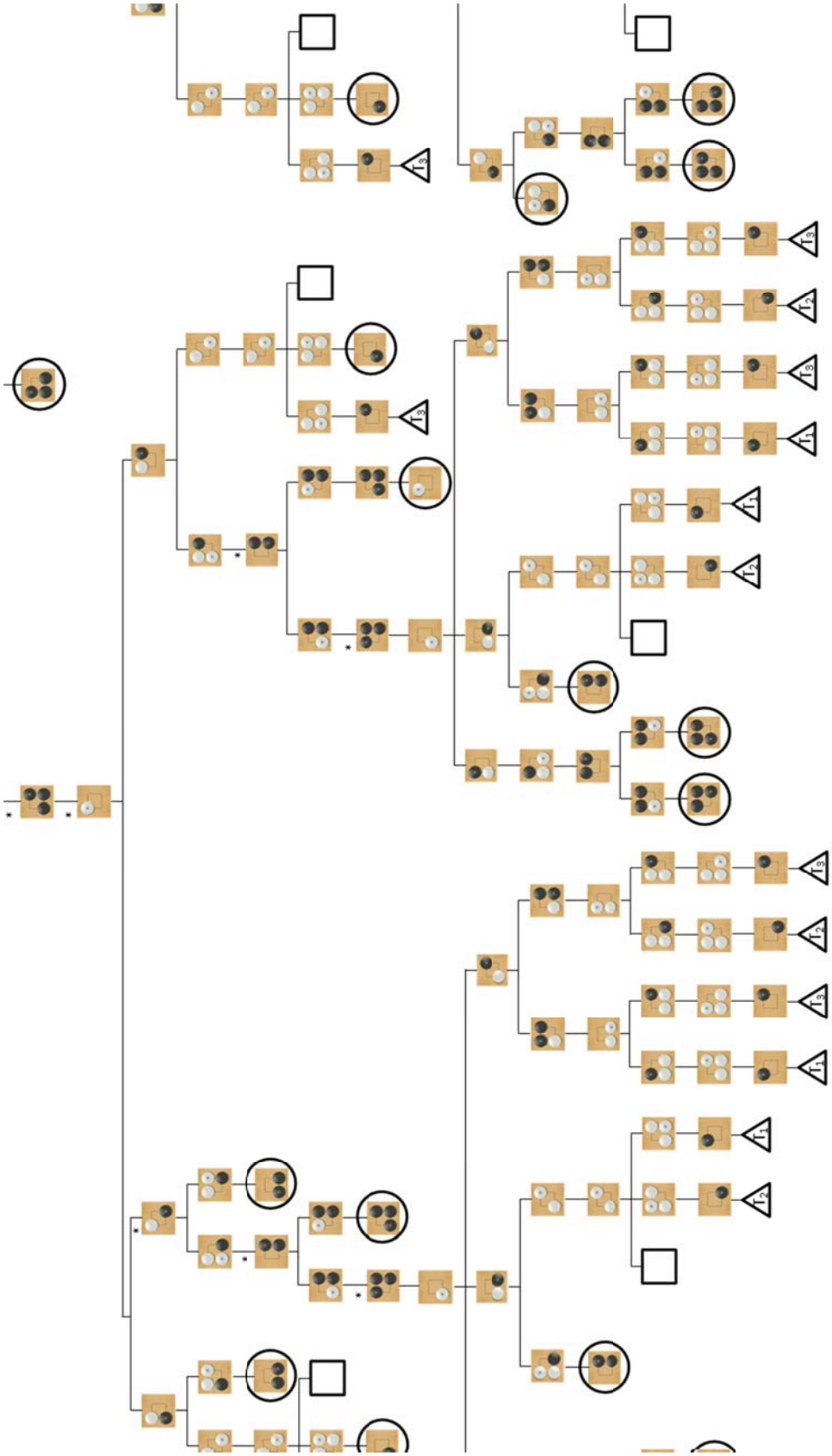
Note the following two board statuses are different. The next player for the board on the left is black; one for the right is white.

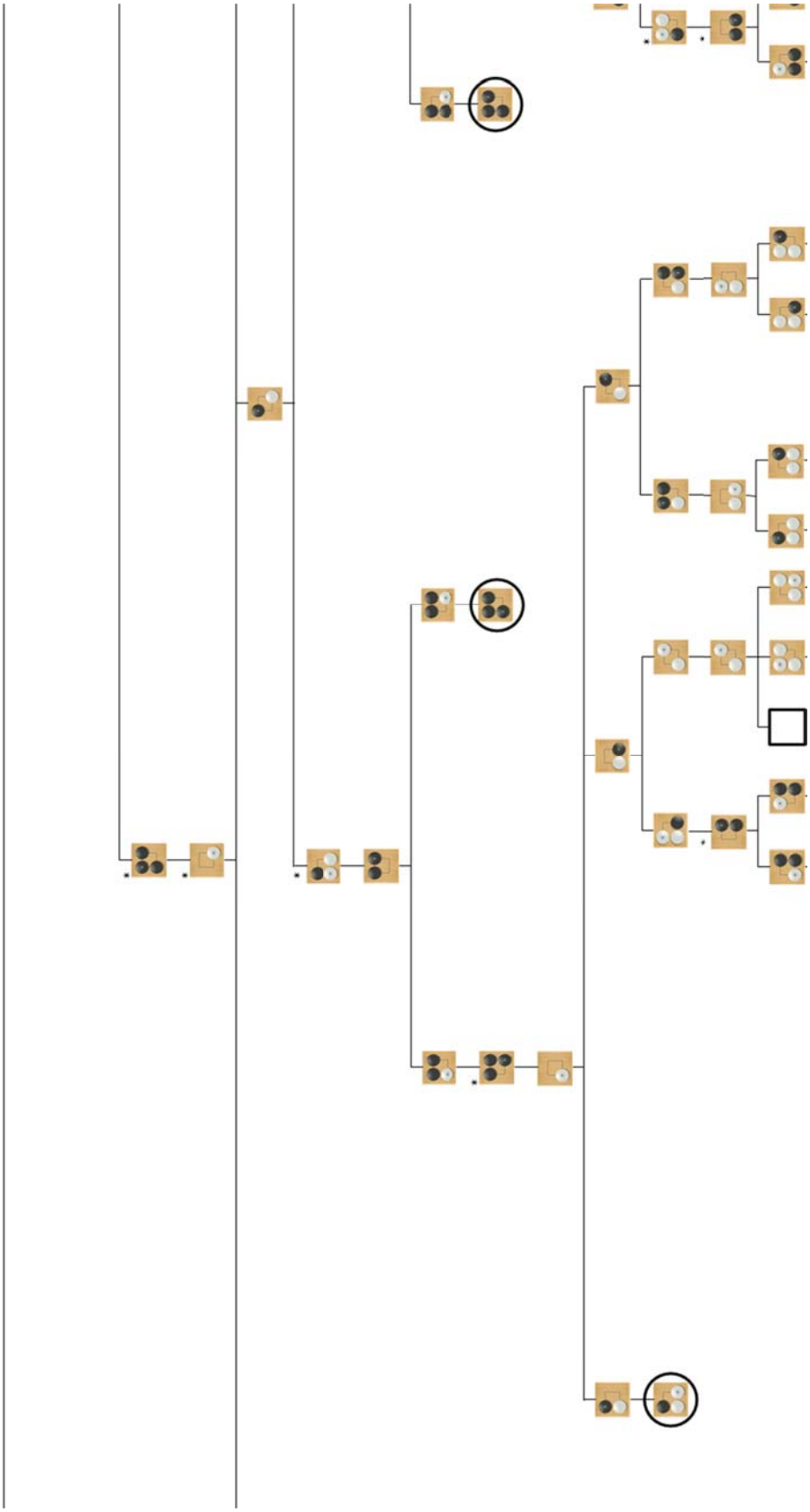


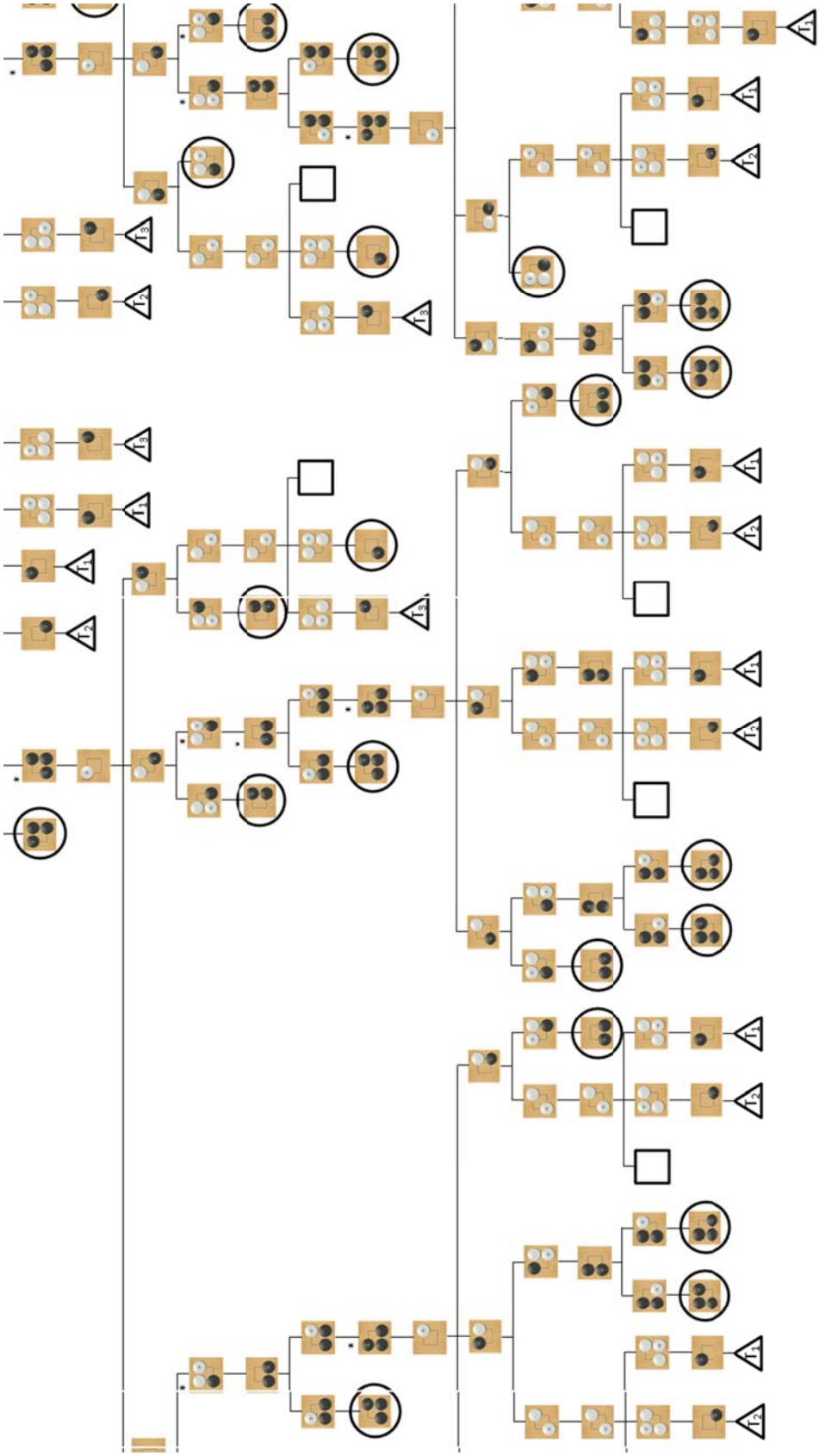


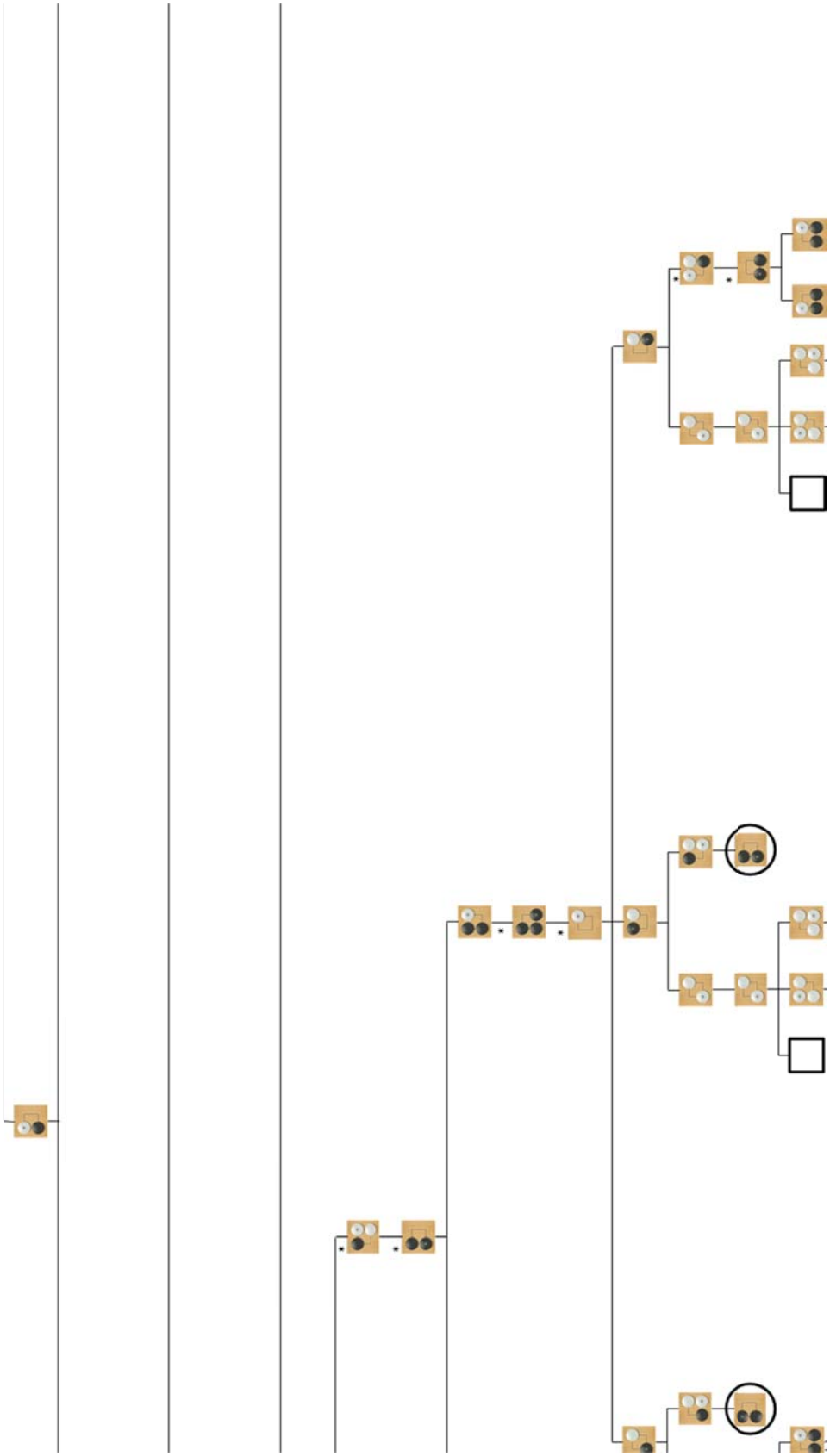


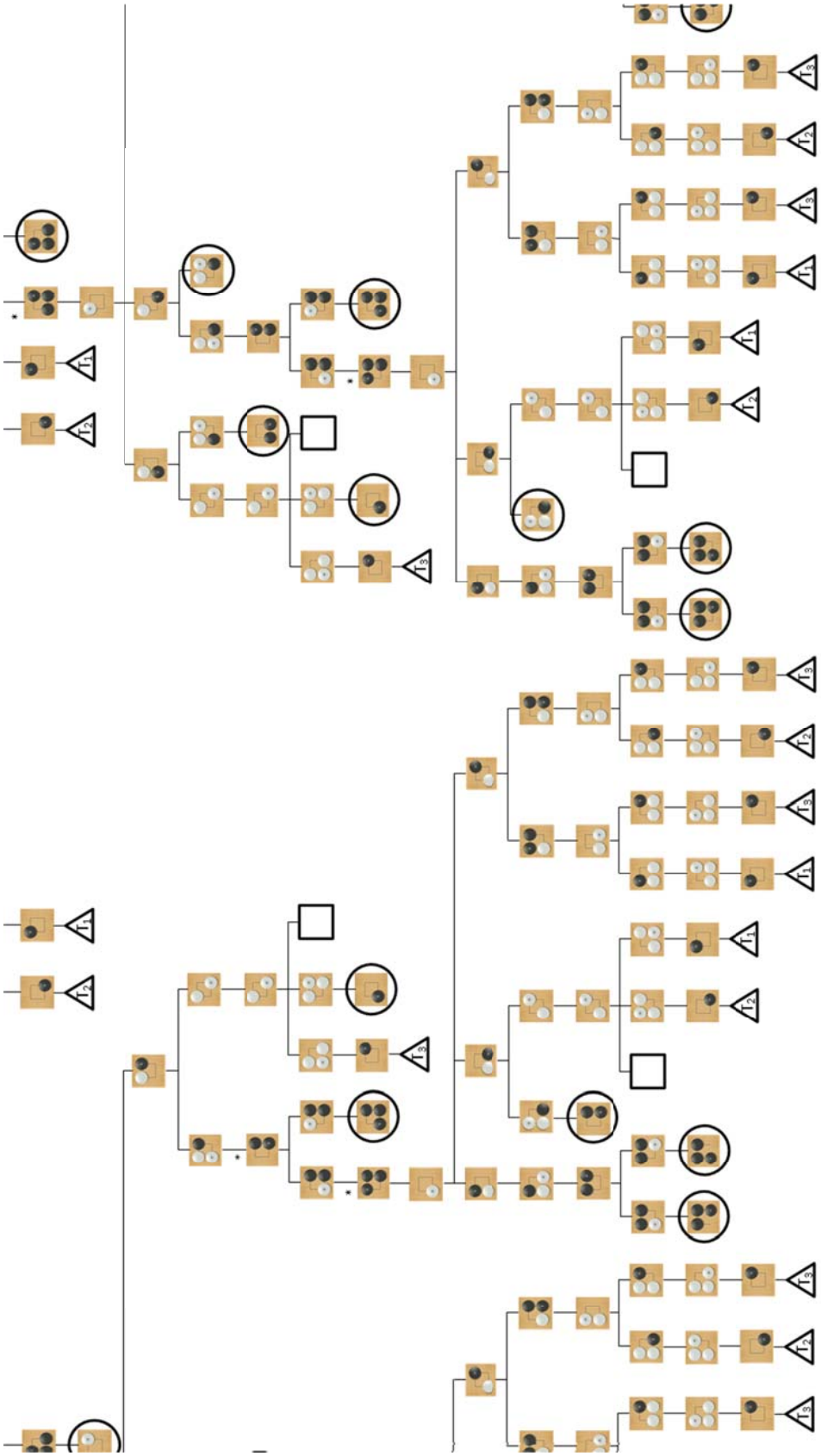


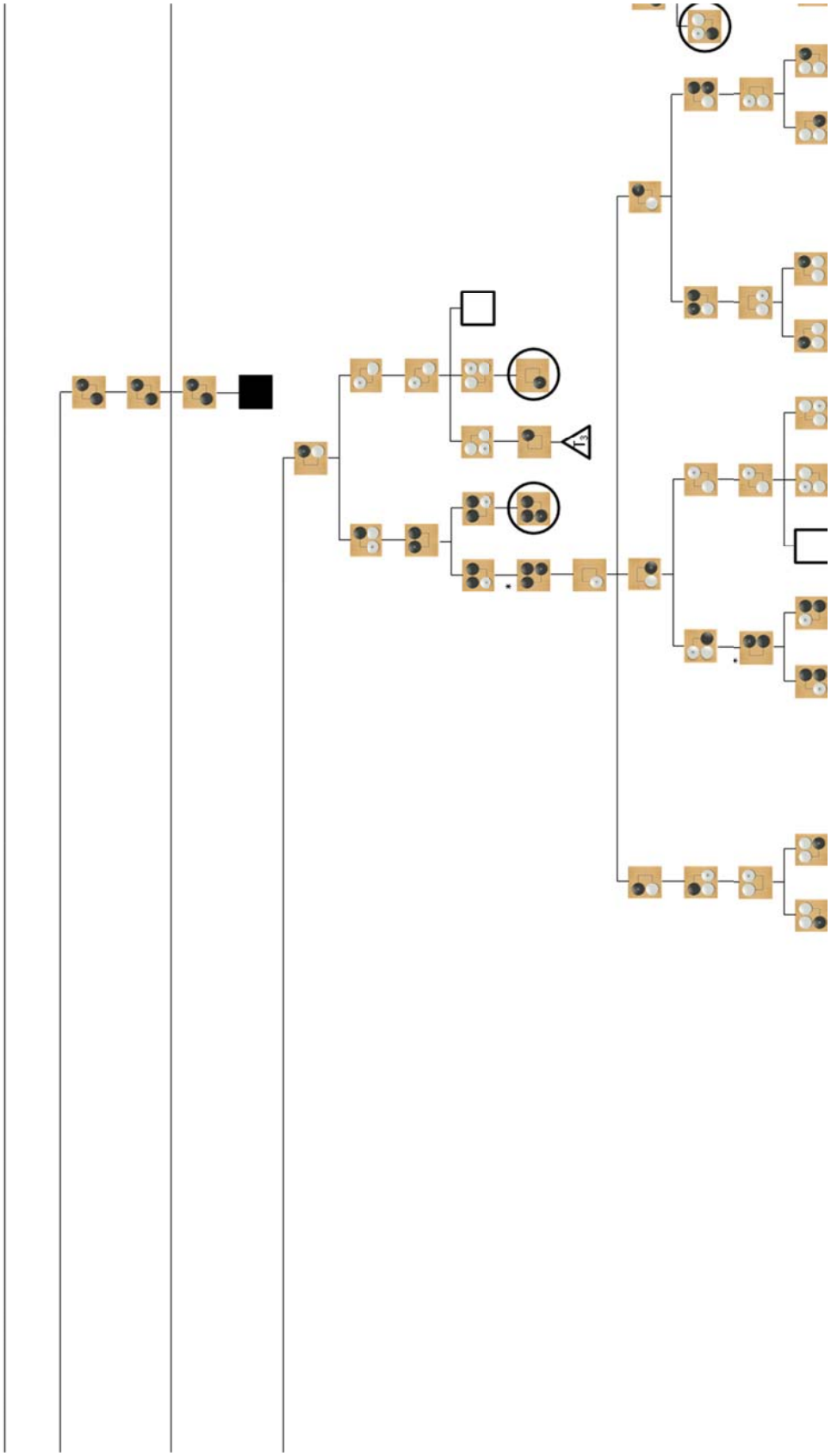


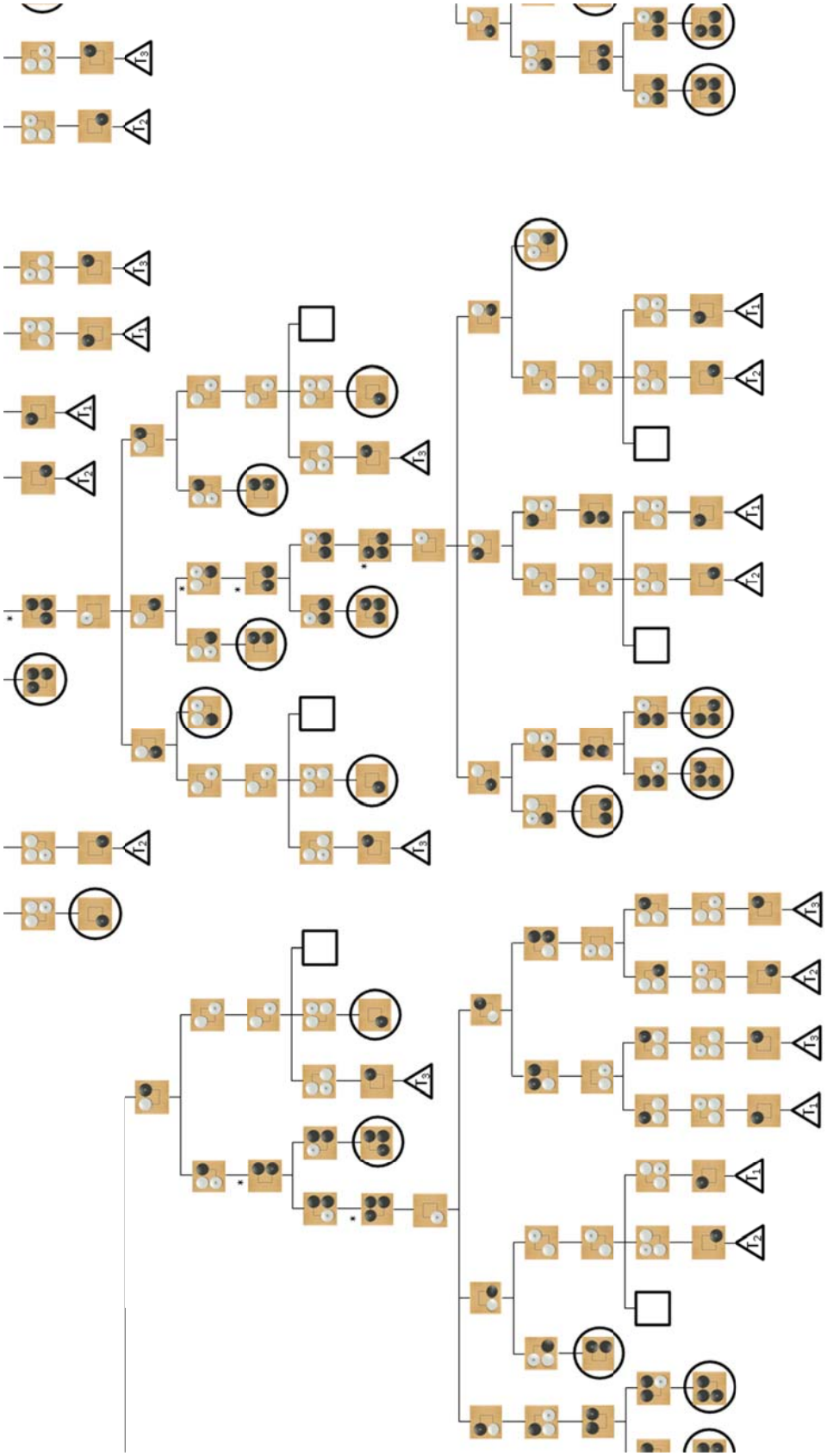


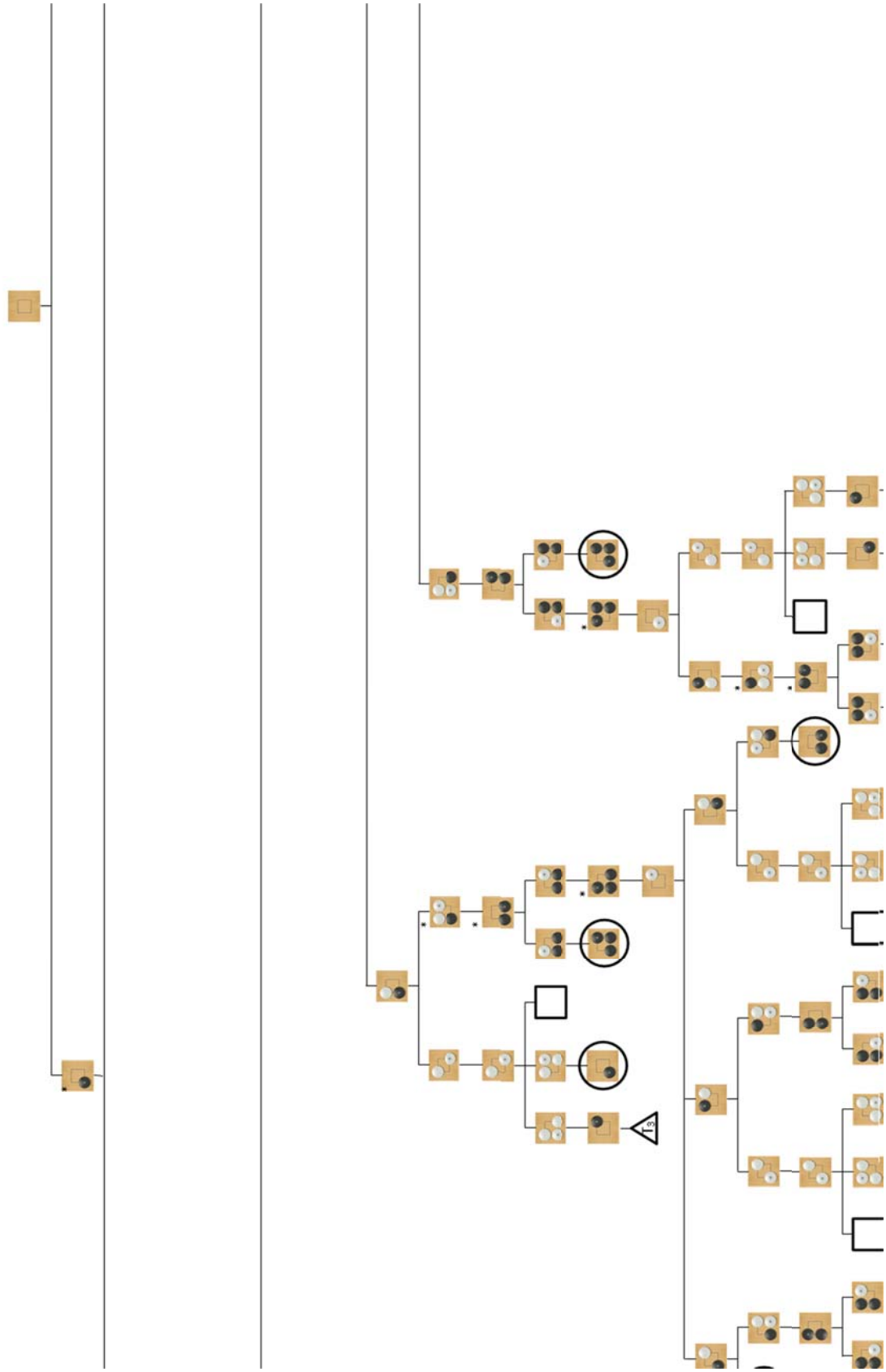


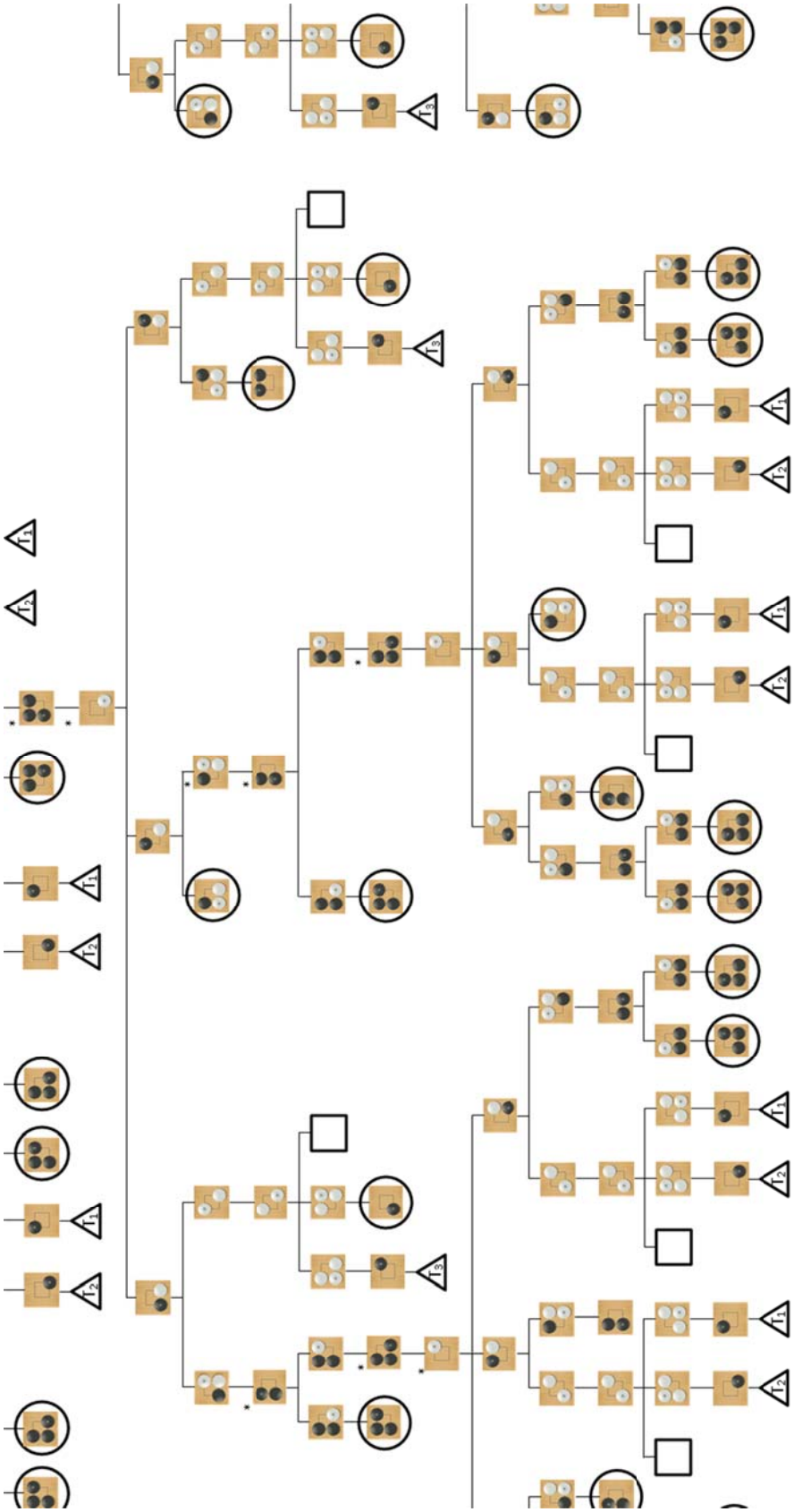


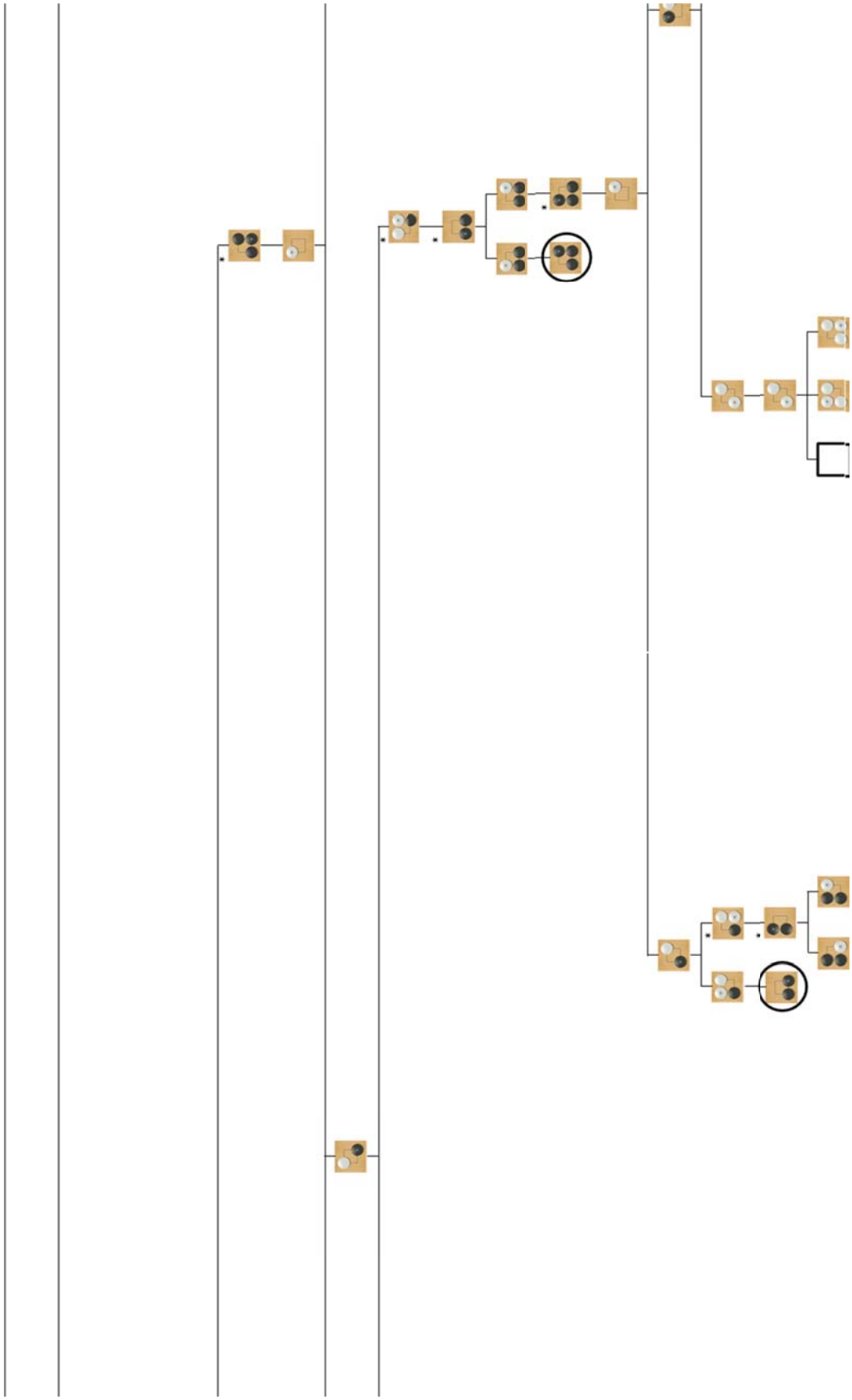


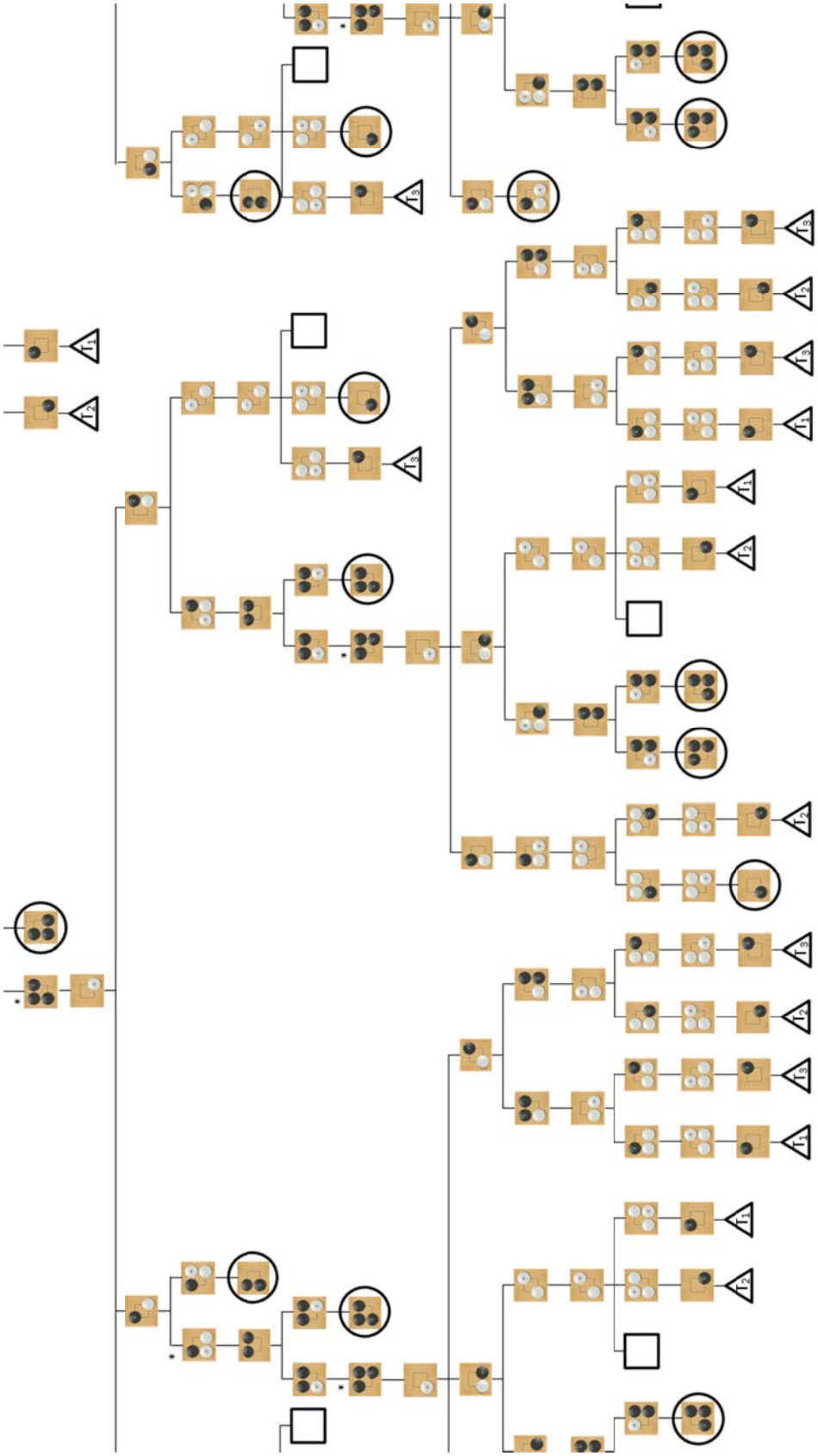


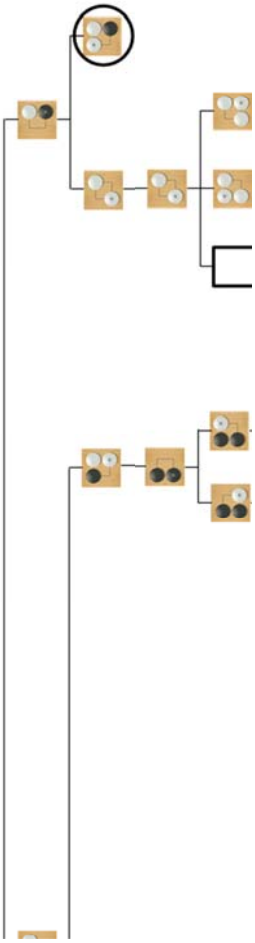


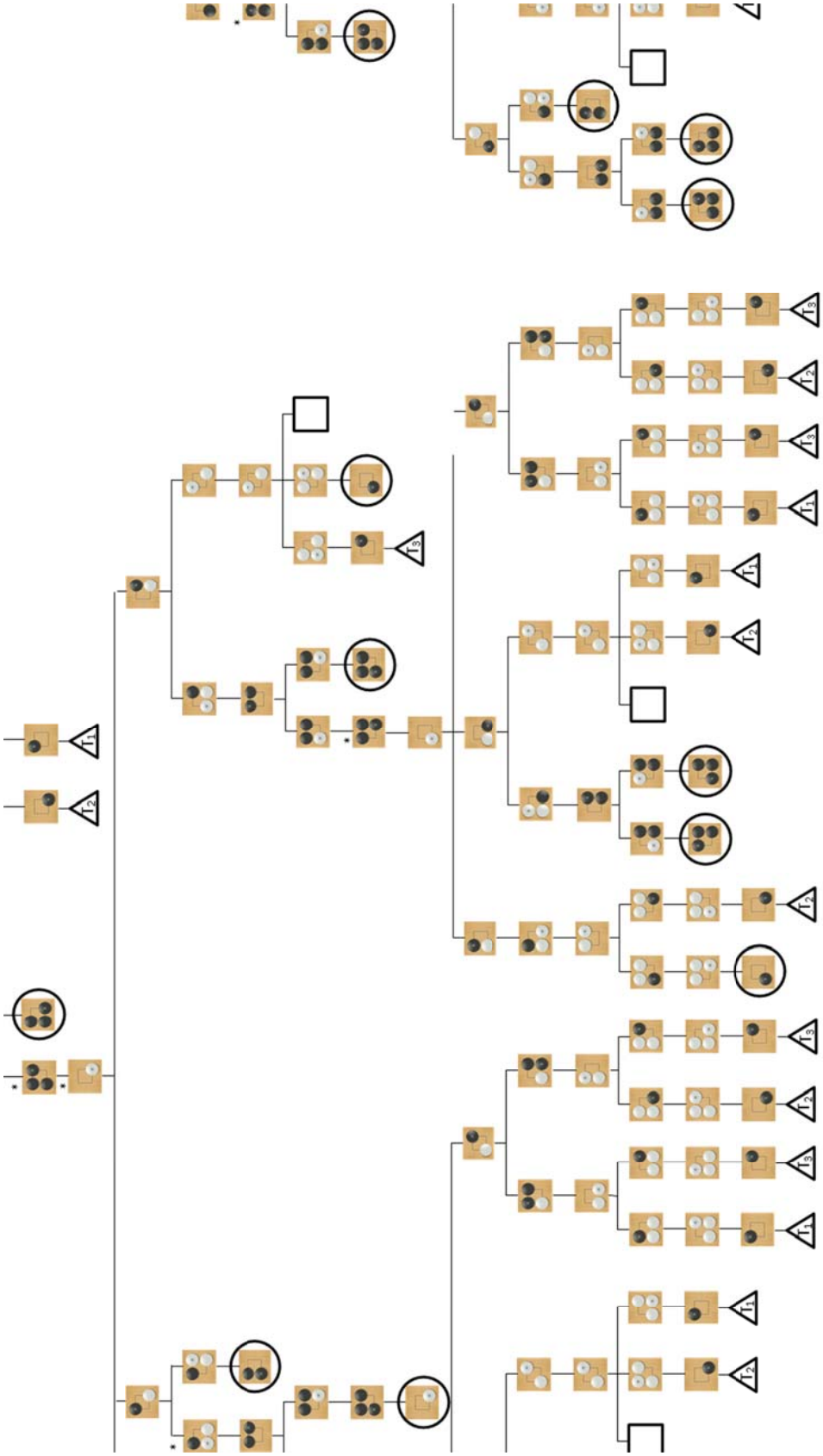


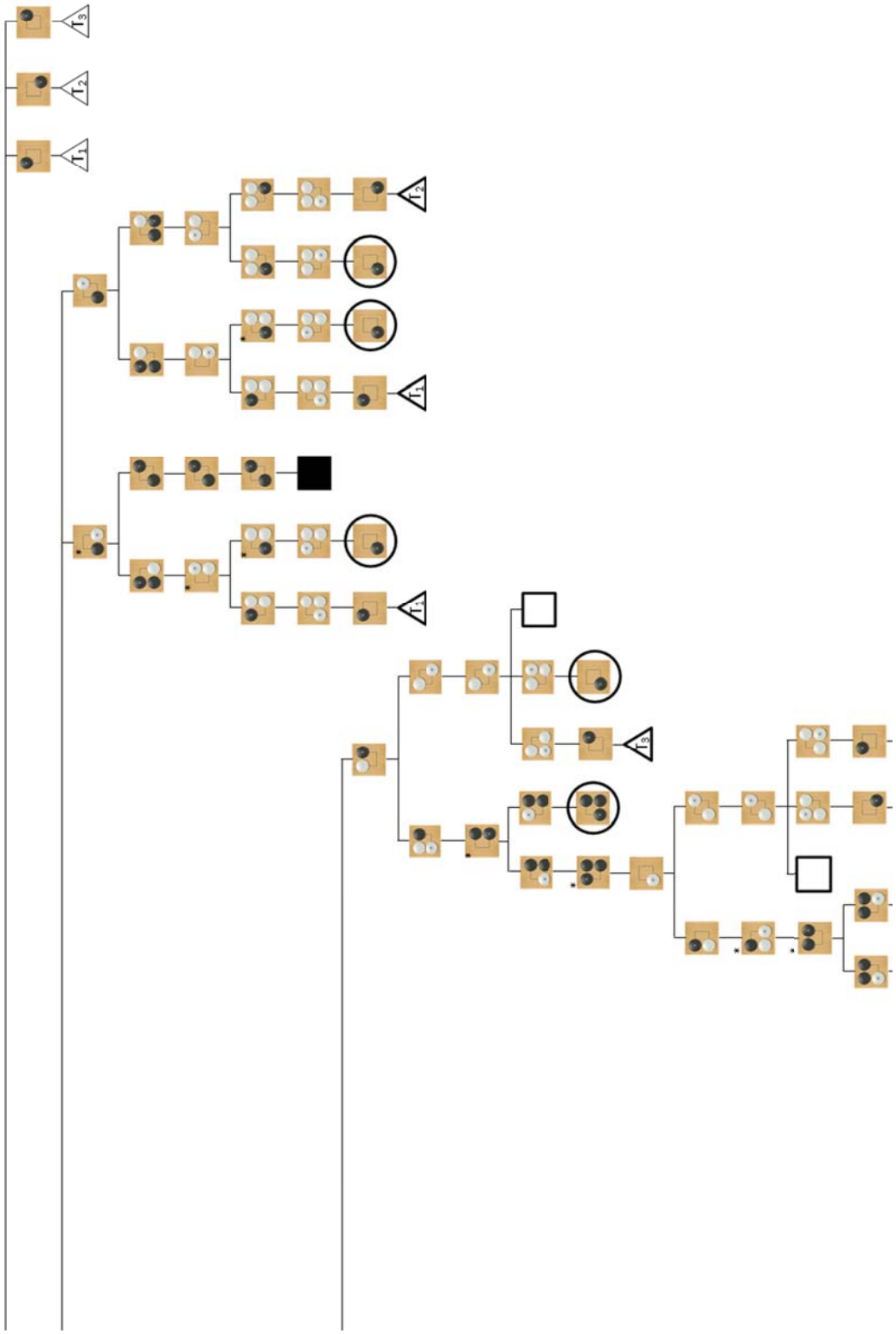


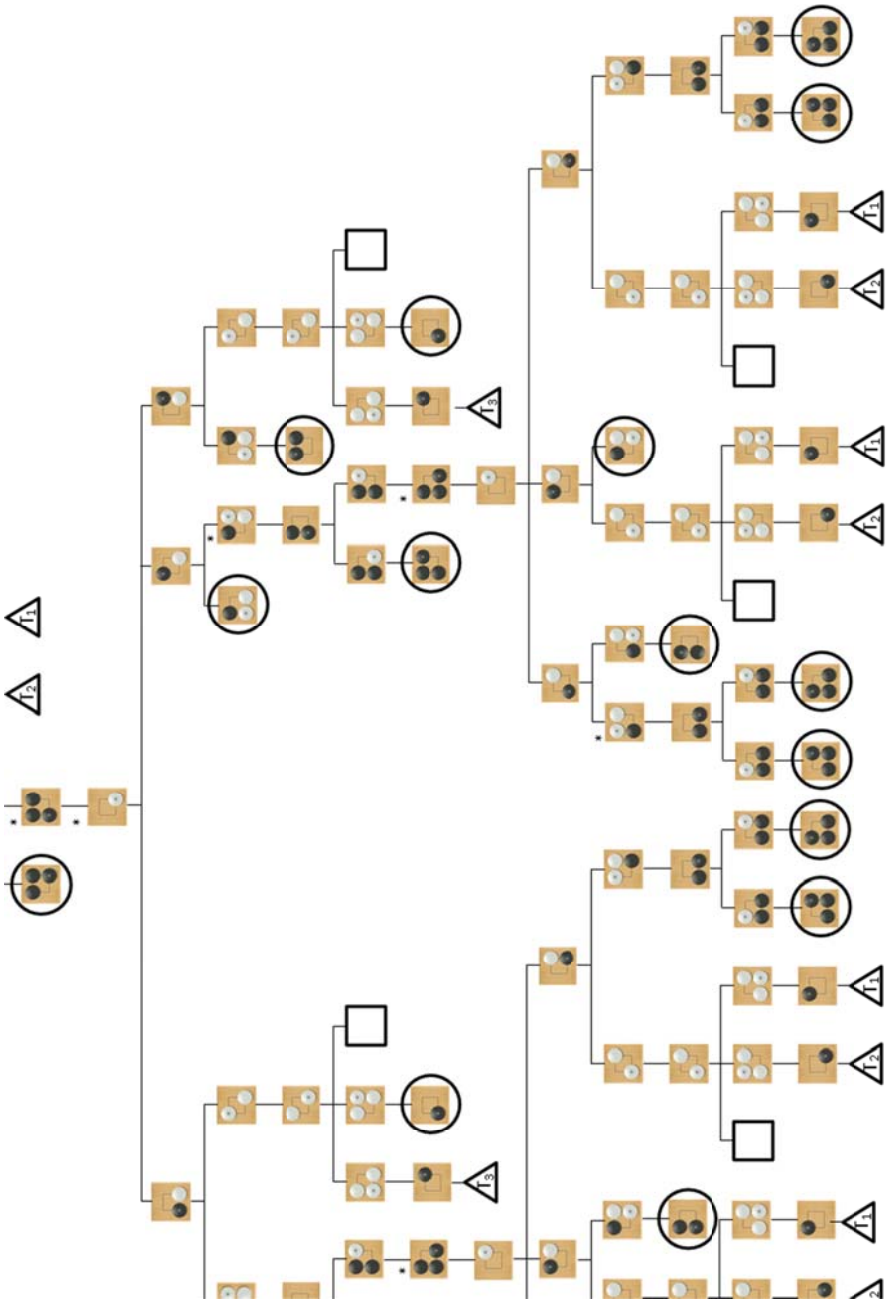









T_3
T_2
T_1
T_2
T_1
T_1
T_3



BIBLIOGRAPHY

[1] Andries P. Engelbrecht, *Computational Intelligence: An Introduction*. West Sussex, England: John Wiley & Sons, 2002.

[2] Janice Kim and Jeong Soo-Hyun, *Learn to Play Go*, I-V, Second printing ed. Denver: Good Move Press, 2005.

[3] Min hee Kim, "Research on the relevance of professional Baduk player's age to achievements," Myong-ji University, Seoul, South Korea, Master's thesis 2005 (in Korean, 김민희, 프로기사의 연령과 성적간의 관련성 연구).

[4] Duc Soo Lim, "Survey on Baduk academy owners' view on Baduk education, Baduk and professionalism," Myong-ji University, Seoul, South Korea, Master's thesis 2006 (In Korean, 바둑교실 원장의 바둑교육관 및 바둑관, 직업의식에 관한 조사 연구).

[5] Jong Ho Yang, "Research on the process to institutionalize Baduk in Korea," Myong-ji University, Seoul, South Korea, Master's thesis 2007 (in Korean, 한국바둑의 제도화 과정에 관한 연구).

[6] Korea Baduk Association. Hakuk Kiwon (in Korean). [Online]. http://www.baduk.or.kr/

[7] Chinese Weiqi Association. Zhonguo Weiqi Xiehui (in Chinese). [Online]. http://www.weiqi.org.cn/

[8] Japan Go Association. Nihon Kiin (in Japanese). [Online]. www.nihonkiin.or.jp

[9] Kansai Go Association. Kansai Ki-in (in Japanese). [Online]. www.kansaikiin.jp

[10] Taiwan Chi Yuan. Taiwan Go Association (in Taiwanese). [Online]. www.taiwango.org.tw

[11] European Go Professionals. [Online]. eurogopro.org

[12] American Go Association. [Online]. www.usgo.org

[13] Canadian Go Association. [Online]. www.go-canada.org

[14] Singapore Weiqi Association. [Online]. www.weiqi.org.sg

[15] European Go Federation. [Online]. www.eurogofed.org

[16] Online Go server. TYGEM (in Korean). [Online]. www.tygem.com/

[17] Online Go server. Cyberoro (In Korean). [Online]. cyberoro.com/

[18] Online Go server. HANGAME (in Korean). [Online]. baduk.hangame.com

[19] Online Go server. The KGS Go Server. [Online]. www.gokgs.com

[20] Online Go server. IGS, PANDANET Internet Go Server. [Online]. www.pandanet-igs.com

[21] International Go Federation. [Online]. intergofed.org

[22] Xindi Cai and Donald C. Wunsch II, "Computer Go: A Grand Challenge to AI," *Challenges for Computational Intelligence, Heidelberg: Springer Berlin*, vol. 63, pp. 443-465, May 2007.

[23] Martin Müller, "Computer Go," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 145-179, Jan. 2002.

[24] Bruno Bouzy and Tristan Cazenave, "Computer Go: An AI Oriented Survey," *Artificial Intelligence*, vol. 132, pp. 39-103, 2001.

[25] Jay Burmeister and Janet Wiles, "AI Techniques Used in Computer Go," in *Fourth Conference of the Australasian Cognitive Science Society*, Newcastle, NSW, Australia, 1997.

[26] Feng-Hsiung Hsu, "Cracking Go," *IEEE Spectrum*, vol. 44, no. 10, pp. 50-55, September 2007. [Online]. http://spectrum.ieee.org/computing/software/cracking-go/0

[27] Jan Krikke, "The Challenge of Go," *IEEE Intelligent Systems*, vol. 22, no. 6, pp. 11-18, Nov./Dec. 2007.

[28] Norbert Wiener, *Cybernetics: Or Control and Communication in the Animal and the Machine*. Cambridge, MA, USA: MIT Press, 1948.

[29] Charles E. Leiserson, Ronald L. Rivest, Clifford Stein Thomas H. Cormen, *Introduction to Algorithms*, 2nd ed. Cambridge, MA, USA: The MIT Press, 2001.

[30] Claude Shannon, "Programming a Computer for Playing Chess," *Philosophical Magazine*, vol. Ser.7, Vol. 41, no. No. 314, March 1950.

[31] Alan Turing, *Faster than Thought*, Pitman B. V. Bowden, Ed. London , 1953.

[32] Alan Turing. (1953) Faster than Thought reproduced by Der Spiegel Online. [Online]. http://www.spiegel.de/static/flash/schach/turingengine/turingenginehuman1953.htm

[33] IBM Research. Deep Blue. [Online]. http://www.research.ibm.com/deepblue/

[34] Albert L. Zobrist, "Feature Extraction and Representation for Pattern Recognition and the Game of Go," University of Wisconsin, Ph.D. dissertation 1970.

[35] Martin Mueller, "Not Like Other Games - Why Tree Search in Go is Different," in *In Proceedings of the Fifth Joint Conference on Information Sciences*, 2000, pp. 974-977.

[36] Byung-Doo Lee and Soo-Hyun Jeong, *Computer Go*. Seoul: ITMedia, 2008.

[37] Imran Ghory, "Reinforcement learning in board games," Department of Computer Science, University of Bristol, Technical Report CSTR-04-004 May 2004.

[38] David E. Moriarty, Risto Miikkulainen Norman Richards, "Evolving Neural Networks to Play Go," *Applied Intelligence*, vol. 8, pp. 88-96, 1998.

[39] CGoban, Feb. 2002. [Online]. http://www.igoweb.org/~wms/comp/cgoban/

[40] GoGui. [Online]. http://gogui.sourceforge.net/

[41] Go Modem Protocol. [Online]. http://www.britgo.org/tech/gmp.html

[42] GTP - Go Text Protocol. [Online]. http://www.lysator.liu.se/~gunnar/gtp/

[43] Rémi Coulom. Crazy Stone. [Online]. http://remi.coulom.free.fr/CrazyStone/

[44] Anders Kierulf, "Smart Game Board: a Workbench for Game-Playing Programs with Go and Othello as Case Studies," ETH Zuerich, PhD thesis 1990.

[45] Janusz Kraszek, "Heuristics in the life and death algorithm of a Go-playing program," *Computer Go*, vol. 9, pp. 13-24, 1988.

[46] Thomas Wolf, "Investigating tsumego problems with RisiKo," *Heuristic Programming in Artificial Intelligence*, 1991.

[47] Martin Mueller, "Race to capture: Analyzing semeai in Go," in *In Game Programming Workshop in Japan*, 1999, pp. 61-68. [Online]. igs.nuri.net

[48] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, USA: MIT Press, 1998.

[49] Bruno Bouzy and Bernard Helmstetter, "MONTE-CARLO GO DEVELOPMENTS," *Advances in Computer Games*, vol. 10, 2003.

[50] Kazuki Yoshizoe, Tomoyuki Kaneko, Akihiro Kishimoto, Kenjiro Taura Haruhiro Yoshimoto, "Monte Carlo Go has a Way to Go," in *the 21st national conference on Artificial intelligence*, Boston, MA, USA, 2006.

[51] Sylvain Gelly and David Silver, "Combining Online and Offline Knowledge in UCT," in *24th International Conference on Machine Learning*, Corvallis, OR, USA, 2007.

[52] Bernd Bruegmann. (1993) Monte Carlo Go. [Online]. http://www.ideanest.com/vegos/MonteCarloGo.pdf

[53] David Fotland, "Static Eye Analysis in The Many Faces of Go," *ICGA Journal*, vol. 25, no. 4, pp. 203-210, 2002.

[54] Ken Chen and Zhixing Chen, "Static analysis of life and death in the game of Go," *Information Sciences*, vol. 121, no. 1-2, pp. 113-134, 1999.

[55] Bruno Bouzy and Bernard Helmstetter, "Ddveloptments on Monte Carlo Go," *Advances in Computer Games*, 2003.

[56] Levente Kocsis and Csaba Szepesvari, "Bandit based Monte-Carlo Planning," *European Conference on Machine Learning in Lecture Notes in Computer Science*, vol. 4212, pp. 282-293, 2006.

[57] Martin L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 2nd ed.: Wiley, John & Sons Inc., 2005.

[58] Nicolo Cesa-Bianchi, Paul Fischer Peter Auer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2-3, May-June 2002.

[59] Sylvain Gelly and Yizao Wang, "Exploration exploitation in Go: UCT for Monte-Carlo Go," in *Nueral Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop*, Whistler, Canada, 2006.

[60] Yizao Wang, Remi Munos and Olivier Teytaud Sylvain Gelly, "Modifications of UCT with Patterns in Monte-Carlo Go," INRIA, Université Paris Sud - Paris XI, Technical Report 6062 2006.

[61] Yizao Wang and Sylvain Gelly, "Modifications of UCT and sequence-like simulations for Monte-Carlo Go," in *IEEE Symposium on Computational Intelligence and Games*, Hanolulu, Hawaii, USA, 2007.

[62] Sylvain Gelly. MoGo. [Online]. http://www.lri.fr/~gelly/MoGo.htm

[63] Louis Chatriot, C. Fiter, Sylvain Gelly, Jean-Baptiste Hoock, Julien Perez, Arpad Rimmel, Olivier Teytaud Guillaume Chaslot. (2008) Combining expert, offline, transient and online knowledge in Monte-Carlo exploration. [Online]. www.lri.fr/~teytaud/eg.pdf

[64] Sylvain Gelly and David Silver, "Achieving Master Level Play in 9 x 9 Computer Go," in *the Twenty-Third AAAI Conference on Artificial Intelligence*, Chicago, IL, USA, 2008, pp. 1537-1540.

[65] Computer Go Tournaments on KGS. [Online]. http://www.weddslist.com/kgs/index.html

[66] Mei-Hui Wang, Guillaume Chaslot, Jean-Baptiste Hoock, Arpad Rimmel, Olivier Teytaud, Shang-Rong Tsai, Shun-Chin Hsu, Tzung-Pei Hong Chang-Shing Lee, "The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 1, March 2009.

[67] US Go Congress. [Online]. http://www.gocongress.org/

[68] "Artificial Intelligence: Super-computer Defeats Human Go Pro," *Communications of the ACM*, vol. 51, no. 10, pp. 12-13, October 2008.

[69] (2008) MogoTitan vs. Kim MyungWan 8p during US Go Congress 2008. [Online]. http://files.gokgs.com/games/2008/8/7/MyungWan-MoGoTiTan-4.sgf

[70] News Staff. (2009, May) MoGo TITAN And Huygens Set A GO World Record. [Online]. http://www.science20.com/news_articles/mogo_titan_and_huygens_set_go_world_record

[71] Peter Drake and Steve Uurtamo, "Heuristics in Monte Carlo Go," in *the 2007 International Conference on Artificial Intelligence*, Las Vegas, NV, USA, 2007.

[72] David Fotland. The Many Faces of Go. [Online]. http://www.smart-games.com/manyfaces.html

[73] Open source program initialized by the University of Alberta. Fuego. [Online]. http://fuego.sourceforge.net/

[74] Bruno Bouzy, "Monte-Carlo Go Reinforcement Learning Experiments," in *IEEE Symposium on Computational Intelligence in Games*, Reno, NV, USA, 2006.

[75] Bruno Bouzy, "The Move Decision Strategy of Indigo," *International Computer Games Association (ICGA) Journal*, pp. 14-27, March 2003.

[76] Jahn-takeshi Saito, Guillaume Chaslot, Jos W. H. M. Uiterwijk Joris Borsboom. A Comparison of Monte-Carlo Methods for Phantom Go. [Online]. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.7715&rep=rep1&type=pdf

[77] Tristan Cazenave, "A Phantom Go program," in *the 11th Advances in Computer Games Conference, Lecture Notes on Computer Science*, Taipei, Taiwan, 2005.

[78] Remi Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," *Computers and Games, Lecture Notes in Computer Science*, vol. 4630, pp. 72-83, 2006.

[79] Gerald Tesauro, "Temporal Difference Learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, March 1995.

[80] Mohammed Raonak-Uz-Zaman, "Applications of Neural Networks in Computer Go," Texas Tech University, Lubbok, Ph.D. dissertation 1998.

[81] Peter Dayan, Terrence J. Sejnowski Nicol N. Schraudolph, "Temporal difference learning of position evaluation in the game of Go," in *Advances in Neural Information Processing Systems 6*, San Mateo, CA, USA, 1994, pp. 817-824.

[82] NeuroGo. [Online]. http://webdocs.cs.ualberta.ca/~emarkus/neurogo/

[83] Markus Enzenberger, "Evaluation in Go by a Neural Network Using Soft Segmentation," in *10th Advances in Computer Games conference*, Graz, Austria, 2003.

[84] Markus Enzenberger. (1996, September) The Integration of A Priori Knowledge into a Go Playing Neural Network. [Online]. http://webdocs.cs.ualberta.ca/~emarkus/neurogo/neurogo1996.html

[85] Thomas Philip Runarsson and Simon M. Lucas, "Coevolution versus self-play temporal difference learning for acquiring position evaluation in small-board Go," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 6, pp. 628-640, December 2005.

[86] Irwin King, John C.S. Lui Horace Wai-kit Chan, "Performance analysis of a new updating rule for TD(λ) learning in feedforward networks for position evaluation in Go game," in *Proc. of International Conference on Neural Networks*, Washington, DC , USA, Jun.3-6 1996, pp. 1716 - 1720, vol.3.

[87] Mohammed Raonak-Uz-Zaman and Donald C. Wunsch III, "TD methods applied to mixture of experts for learning 9×9 Go evaluation function," in *International Joint Conference on Neural Networks*, Washington DC, USA, 1999, pp. Vol. 6, 3734-3739.

[88] Raonak Zaman, Danil Prokhorov, and Donald C. Wunsch II, "Adaptive Critic Design in Learning to Play Game of Go," in *International Conference on Neural Network*, Houston, TX, USA, 1997, pp. 1-4, vol. 1.

[89] Lin Wu and Pierre Baldi, "Learning to play Go using recursive neural networks," *Neural Networks*, vol. 21, no. 9, February 2008.

[90] Jay Ramon and Hendrik Blockeel, "A survey of the application of machine learning to the game of Go," in *the First International Conference on Baduk*, Yongin, South Korea, 2001, pp. 1-10.

[91] Helmut A. Mayer, "Board Representations for Neural Go Players Learning by Temporal Difference," in *2007 IEEE Symposium on Computational Intelligence and Games*, Honolulu, Hawaii, USA, 2007, pp. 183-188.

[92] Alan Blair, "Learning position evaluation for Go with Internal Symmetry Networks," in *IEEE Symposium on Computational Intelligence and Games*, Perth, Australia, 2008, pp. 199-204.

[93] CGOS (Computer Go Server). [Online]. http://cgos.boardspace.net/

[94] Krzysztof Krawiec and Marcin Szubert, "Coevolutionary Temporal Difference Learning for small-board Go," in *IEEE Congress on Evolutionary Computation*, Barcelona, Spain, 2010, pp. 1-8.

[95] Charles E. Leiserson, Ronald L. Rivest Thomas H. Cormen, *Introduction to algorithms*, Third Edition ed.: MIT Press, 2009.

[96] Takuya Kojima, "Automatic Acquisition of Go Knowledge from Game Records: Deductive and Evolutionary Approaches," University of Tokyo, Tokyo, Japan, Ph.D. thesis 1998.

[97] Kazuhiro Ueda, Saburo Nagano Takuya Kojima, "An evolutionary algorithm extended by ecological analogy and its application to the game of Go," in *15th International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997, pp. 684-689.

[98] Kazuhiro Ueda, Saburo Nagano Takuya Kojima, "Flexible acquisition of various types of knowledge from game records: Application to the game of Go," in *IJCAI-97 Workshop on Using Games as an Experimental Testbed for AI Research*, Nagoya, Japan, 1997, pp. 51-57.

[99] Remi Coulom, "Computing Elo Ratings of Move Patterns in the Game of Go," in *Computer Games Workshop*, Amsterdam, Netherlands, 2007.

[100] Richard S. Sutton, Martin Mueller David Silver, "Sample-based learning and search with permanent and transient memories," in *the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008, pp. 968-975.

[101] Elwyn Berlekamp and David Wolfe, *Mathematical Go-Chilling Gets the Last Point*. Wellesley, MA, USA: A K Peters/CRC Press, 1994.

[102] Elwyn Berlekamp, Bill Spight Martin Mueller, "Generalized Thermography: Algorithms, Implementation, and Application to Go Endgame," International Computer Science Institute, ICSI Technical Report TR-96-030 October 1996.

[103] Martin Mueller, "Generalized thermography: A new approach to evaluation in computer Go," in *IJCAI-97 Workshop on Using Games as an Experimental Testbed for AI Research*, Nagoya, Japan, 1997, pp. 41-49.

[104] Martin Mueller, "Generalized Thermography: A new approach to evaluation in computer Go," in *reprinted in Games in AI Research, first published in Proceedings IJCAI-97 Workshop, pp. 41-49, Nagoya, Japan, 1997*, 2000, pp. 203-219.

[105] Martin Mueller, "Decomposition search: A combinatorial games approach to game tree search, with applications to solving Go endgames," in *International Joint Conferences on Artificial Intelligence*, vol. 1, 1999, pp. Vol. 1, pp.578-583.

[106] Martin Mueller and Ralph Gasser, "Experiments in computer Go endgame," in *Games of No Chance*.: Cambridge University Press, 1996, pp. 273-284.

[107] David Wolfe, "Go Endgames Are PSPACE-hard," in *Theoretical Computer Science*, 1999.

[108] Changhyuk You, *Introduction to life-death problem for beginners 1-3*. Seoul: Dasan Books, Mar. 2003, (in Korean).

[109] Erik C.D. van der Werf, Mark H.M.Winands, H. Jaap van den Herik, and Jos W.H.M. Uiterwijk, "Learning to predict life and death from Go game records," *Information Sciences: An International Journal*, vol. 175, no. 4, pp. 258-272, Nov. 2005.

[110] David B. Benson, "Life in the Game of Go," *Information Sciences*, vol. 10, pp. 17-29, 1976.

[111] Martin Mueller, "Playing it safe: Recognizing secure territories in computer Go by using static rules and search," in *Game Programming Workshop*, Tokyo, Japan, 1997, pp. 80-86.

[112] R. Vila and T. Cazenave, "When one eye is sufficient: a static approach classification," in *Conference on Advances in Computer Games*, 2003, pp. 109-124.

[113] Akihiro Kishimoto and Martin Mueller, "Df-pn in Go: Application to the one-eye problem," in *In Advances in Computer Games. Many Games, Many Challenges*.: Kluwer Academic Publishers, 2003, pp. 125-141.

[114] Ayumu Nagai and Hiroshi Imai, "Application of df-pn to Othello endgames," in *Game Programming Workshop in Japan*, Tokyo, Japan, 1999, pp. 16-23.

[115] Akihiro Kishimoto and Martin Mueller, "Search versus knowledge for solving life and death problems in Go," in *Twentieth National Conference on Artificial Intelligence*, Pittsburgh, PA, USA, 2005, pp. 1374-1379.

[116] Akihiro Kishimoto and Martin Mueller, "Dynamic decomposition search: A divide and conquer approach and its application to the one-eye problem in Go," in *IEEE Symposium on Computational Intelligence and Game*, Colchester, CT, USA, 2005, pp. 164-170.

[117] Xiaozhen Niu and Martin Mueller, "An improved safety solver for computer Go," in *Computers and Games: 4th International Conference*, 2006.

[118] Akihiro Kishimoto, Martin Mueller Xiaozhen Niu, "Recognizing seki in computer Go," *Lecture Notes in Computer Science, Advances in Computer Games. 11th International Conference*, vol. 4250, pp. 88-103, 2006.

[119] Byung-Doo Lee, "Life-and-Death Problem Solver in Go," Communication and Information Technology Research Technical Report 145 2004.

[120] Erik C.D. van der Wer, Mark H.M. Winands, H. Jaap Van Den Herik, and Jos W. H. M. Uiterwijk, "Learning to Predict Life and Death from Go Game Records," *Information Sciences: An International Journal*, vol. 175, no. 4, November 2005.

[121] Thomas Wolf, "The program GoTools and its computer generated tsume Go database," in *Game Programming Workshop in Japan*, Tokyo, Japan, 1994, pp. 84-86.

[122] Thomas Wolf, "About problems in generalizing a tsumego program to open positions," in *Game Programming Workshop in Japan*, Tokyo, Japan, 1996, pp. 20-26.

[123] Thomas Wolf, "Forward pruning and other heuristic search techniques in tsume Go," *Information Sciences*, vol. 122, no. 1, pp. 59-76, 2000.

[124] David Moriarty, Risto Miikkulainen Norman Richards, "Evolving Neural Networks to Play Go," *Applied Intelligence*, vol. 8, pp. 85-96, 1998.

[125] Julian Churchill, Richard Cant, and David Al-dabass. A New Computational Approach to the Game of Go. [Online]. ducati.doc.ntu.ac.uk/uksim/dad/webpagepapers/Game-14.pdf

[126] Yuri Malitsky, Christopher Fellows, and Gregory Wojtaszczyk, "Detecting the Community Structures in the Game of Go," in *IADIS European Conference on Data Mining*, Lisbon, Portugal, 2007.

[127] A charity match for Haiti victims Sedol Lee vs. Changho Lee. (2010, Feb.) [Online]. http://open.cyberoro.com/gibo/201002/100211-it-Lee.C.H.sgf

[128] Paul J. Werbos and Xiaozhong Pang, "Generalized maze navigation: SRN critics solve what feedforward or Hebbian nets cannot," in *Proc. of IEEE International Conference on Systems, Man, and Cybernetics*, Beijing, China, Oct. 14-17, 1996, pp. 1764-1769.

[129] Donald C. Wunsch II, "The Cellular Simultaneous Recurrent Network Adaptive Critic Design for the Generalized Maze Problem Has a Simple Closed-Form Solution," in *Proc. of the IEEE-INNS-ENNS International Joint Conference on Neural Networks IJCNN 2000*, Como, Italy, July 24-27, 2000, pp. 79-82.

[130] Roman Ilin, Robert Kozma, and Paul J. Werbos, "Cellular SRN trained by extended Kalman filter shows promise for ADP," in *Proc. of IEEE World Congress on Computational Intelligence*, Vancouver, Canada, July 16-21 2006, pp. 506-510.

[131] Roman Ilin, Robert Kozma, and Paul J. Werbos, "Efficient Learning in Cellular Simultaneous Recurrent Neural Networks - The Case of Maze Navigation Problem," in *Proc. of IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, Honolulu, HI, USA, April 2007, pp. 324-329.

[132] Roman Ilin, Robert Kozma, and Paul J. Werbos, "Beyond Feedforward Models Trained by Backpropagation: A Practical Training Tool for a More Efficient Universal Approximator," *IEEE Trans. on Neural Networks*, vol. 19, no. 6, Jun. 2008.

[133] Robert Kozma and Roman Ilin, "Optimization via efficient learning in CNNs: Cognitively-motivated temporal discount functions in SRNNs," in *12th International Workshop on Cellular Nanoscale Networks and Their Applications*, Berkeley, USA, Feb.3-5, 2010.

[134] Bruno R. Preiss, *Data Structures and Algorithms with Object-Oriented Design Patterns in C++.*: John Wiley & Sons, Aug. 1998.

[135] Marvin Minsky and Seymour Papert, *Perceptrons: an introduction to computational geometry*. Cambridge, MA, USA: MIT Press, 1969.

[136] Simon Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed.: Pearson Education, Inc, 1999.

[137] Yong Ren, Keith Anderson, Khan Iftekharuddin, Paul Kim, and Eddie White, "Pose invariant face recognition using Cellular Simultaneous Recurrent Networks," in *International Joint Conference on Neural Networks*, Atlanta, USA, Jun. 14-19, 2009, pp. 2634 - 2641.

[138] Yong Ren, Keith Iftekharuddin, and Eddie White, "Recurrent network-based face recognition using image sequences," in *IEEE Symposium on Computational Intelligence for Multimedia Signal and Vision Processing*, Nashville, USA, Mar.30-Apr.2, 2009, p. 41.

[139] Keith Anderson, Khan Iftekharuddin, Eddie White, and Paul Kim, "Binary image registration using cellular simultaneous recurrent networks," in *Proc. of IEEE Symposium on Computational Intelligence for Multimedia Signal and Vision Processing*, Nashville, USA, Mar.30-Apr.2, 2009, pp. 61-67.

[140] Lisa L. Grant and Ganesh K. Venayagamoorthy, "Voltage prediction using a Cellular Network," in *Proc. of IEEE Power and Energy Society General Meeting*, Minnesota, USA, 2010, pp. 1-7.

[141] William E. White, Khan Iftekharuddin, and Abdesselam Bouzerdoum, "Improved learning in grid-to-grid neural network via clustering," in *Proc. of International Joint Conference on Neural Networks*, Barcelona, Spain, Jul. 18-23, 2010, pp. 1-7.

[142] Helmut A. Mayer, "Board Representations for Neural Go Players Learning by Temporal Difference," in *Proc. of IEEE Symposium on Computational Intelligence and Games*, Apr.1-5, 2007.

[143] Xindi Cai and Donald C. Wunsch II, "A parallel computer-Go player, using HDP method," in *Proc. of International Joint Conference on Neural Networks*, Washington, DC, USA, Jul.15-19, 2001, pp. 2373-2375, vol.4.

[144] Graham Kendall, Razali Yaakob, and Philip Hingston, "An investigation of an evolutionary approach to the opening of Go," in *Proc. of Congress on Evolutionary Computation*, Portland, OR, USA, Jun.19-23, 2004, pp. 2052-2059, vol.2.

[145] Helmut A. Mayer and Peter Maier, "Coevolution of neural Go players in a cultural environment," in *Proc. of IEEE Congress on Evolutionary Computation*, Edinburgh, UK, Sep.2-5, 2005, pp. 1017, vol.2.

[146] James Kennedy and Russell Eberhart, "Particle swarm optimization," in *Proc. of IEEE International Conference on Neural Networks*, Perth, Australia, Nov.26-Dec.2,1995, pp. 1942-1948, vol. 4.

[147] Russell Eberhart and Yuhui Shi, "A Modified Particle Swarm Optimizer," in *Proc. of the IEEE International Conference on Evolutionary Computation*, Anchorage, AK, USA, May 4-9, 1998, pp. 69-73.

[148] Alex Lubberts and Risto Miikkulainen, "Co-Evolving a Go-playing Neural network," in *Genetic and Evolutionary Computation conference*, San Francisco, CA, USA, 2001.

[149] Chihyung Nam, *Contemporary Go Terms: Definitions & Translations*. Seoul, South Korea: Oromedia Publishing Co., 2004.

[150] Rui Mendes, "Population Topologies and Their Influence in Particle Swarm Performance," Ph.D. dissertation, April 21, 2004, Universidade do Minho.

VITA

Tae-Hyung Kim received his B.S. in Radio Communications Engineering from Yonsei University, Seoul, Republic of Korea, in February 2000. During his undergraduate study, he was on the Dean's list and joined the Republic of Korea Air Force in August 1995 to satisfy the compulsory military service, from which he retired as a sergeant in January 1998. In August 2002, he received his M.S. degree in Electrical and Electric Engineering from Yonsei University, Seoul, Republic of Korea. During his master's degree study, he received the LG Electronics Industry-Academia Scholarship and did an international internship (Praktikum) and thesis work (Diplomarbeit) for Siemens in Bocholt, NRW, Germany through the Siemens Student Program (SSP). In May 2012, he received his Ph.D. degree in Electrical Engineering from the Missouri University of Science and Technology, Rolla, Missouri, USA (formerly, the University of Missouri-Rolla). During his doctoral study, he received the Graduate Travel Fellowship for the International Conference on Cognitive and Neural Systems, Boston, Massachusetts, USA, both in 2007 and 2009.

He has published several conference and journal papers. He and two other colleagues won one of the Best Student Papers distinctions at the 2009 Intelligent System Center Research Symposium, Rolla, Missouri, USA, for their Go robotics work. His research interests include, but are not limited to, topics in computational intelligence with applications in wireless networks and computer Go. Tae-Hyung Kim is a member of the Institute of Electrical and Electronics Engineers and the International Neural Network Society. He has been involved as a member of the Applied Computational Intelligence Laboratory directed by Dr. Donald C. Wunsch II.