Selection and Evolution of Pattern Features for Supervised Learning of an Action Evaluation Function in Board Games

by

Huy Quoc NGUYEN

submitted to
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Associate Professor Kokolo Ikeda

School of Information Science
Japan Advanced Institute of Science and Technology

September, 2014

Abstract

The target of our research is the artificial intelligence of computer games. The the artificial intelligence makes computer games be more interesting. There are several areas of games that AI is contributed to, but we focus on two-player deterministic perfect-information games, because the rules of these games are simple, a lot of people know them, and these systems have been used as the simple testbeds of AI research. Our **goal** is to make strong computer players of board games.

In general **background**, game tree search is frequently used for making computer game player. State evaluation functions are often necessary to evaluate a node, and action evaluation functions are also often used to enhance the search. Machine learning is quite effective to learn good evaluation functions automatically, compared to manual design, and it is already known that selection of features used for machine learning is very important for the performance.

In our specific case, we focus on the Monte Carlo Tree Search for Othello game as a game tree search, it needs a good action evaluation function using some patterns. We employ Bradley-Terry Minorization-Maximization as the learner, because it is effective in the game of Go and attracts attention. In our case, only the patterns are used as the features. But there are a lot of possible patterns, then working with pattern shapes is our motivation because we believe that there are many hidden good pattern shapes which have not been discovered.

From the reasons above, our purpose is to establish the way to obtain good pattern shapes automatically from given game records. Thus, the **research questions** are how to obtain better pattern shapes and how to reduce the optimization cost.

We want to optimize the pattern shapes, so two approaches are proposed to evaluate them. One is to use statistic method to measure the hopefulness of a PS before learning. Another is to use the very early result of BTMM further with less learning data. Two approaches are our **contribution**, and they have some reasonable results.

The game of Othello is used as an environment of experiments in this thesis, but the methods can be applied to almost all board games such as Go, Connect-6, Lines of Action, Hex, Shogi, which patterns are used. From many experiments, we have some following **conclusions**.

Evolution of PSs is done not by using MCTS performance but by light-BTMM for reducing cost. If MCTS performance needs 100 games to evaluate a pattern shape and each game needs 3 minutes, then we need 300 minutes to evaluate a pattern shape. Instead of using MCTS performance, if the heavy-BTMM is used, then it needs 8 minutes to evaluate a pattern shapes. However, the light-BTMM needs nearly 1 minutes to evaluate a pattern shape. It is clear that using light-BTMM reduces cost significantly.

MLE of the optimized patterns is improved compared to the often-used patterns by evolution. Because the MLE of the often-used patterns is -1.5 before optimization, but the MLE of the optimized patterns is -1.4 after optimization. This means that we hope the MCTS performance will be improved if the optimized patterns are applied.

The MCTS performance itself is also improved a lot. The winning ratio between two MCTS programs using the often-used patterns is 50%. But the winning ratio is increased to 70% if the MCTS using the optimized patterns, compare with the MCTS using the optimized patterns.

There are many strange patterns after evolution, this means that it is difficult for programmers to optimize pattern features by manual.

Key words: Feature selection, pattern features, action evaluation function, supervised learning, board games

Acknowledgments

Firstly, I wish to express my sincere gratitude to my principal main supervisor Associate Professor Kokolo Ikeda and sub supervisor Professor Hiroyuki Iida of Japan Advanced Institute of Science and Technology for my constant encouragement and kind guidance during this work.

Secondly, sincere thanks are due to the discussion of my dissertation, including Associate Professor Nguyen Le Minh of Japan Advanced Institute of Science and Technology, Associate Professor Le Hoai Bac of University of Science, Vietnamese National University Ho Chi Minh City, and Assistant Professor Simon Viennot of Japan Advanced Institute of Science and Technology. Their diverse views helped broaden my horizons. They helped me see clearly the weaknesses of the study.

I owe a great debt of gratitude to my parents, my wife, my daughter, my son, and my brother for their continual supports and encouragements during the hard time of the graduate study. Without them, I would not have completed this study.

The last but not least, I would like to express my great gratitude to FIVE-JAIST project (a cooperation between top-five universities in Vietnam, the project 322, and JAIST). JAIST provided me an opportunity of 2.5 years to join the excellent research environment and supported me throughout the studies in this dissertation. The HCMC University of Science provided me the research environment in 1.5 years study in Vietnam.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificial intelligence (AI) is one of the most important research areas in our information society. A lot of methods have been proposed, and a lot of problems have been targeted and solved. Our target is to make strong computer game players, and to improve the AI methods used in these players.

This chapter introduces briefly the history of AIs for board games, the essential concept of "tree search" for board games, the frequent AI method of "machine learning" for enhancing the tree search, and the important concept of "feature" used in machine learning. Finally, our purpose of "feature selection" and research questions are stated, and two approaches are proposed.

1.1 Board games

Board games such as Chess, Go or Shogi (Japanese Chess) have a long history, and a lot of people play them in the world. Games and AI have a closed relationship. AI methods have been always the most important approaches for computer game playing, and also games have been an important testbed of AI research. In the 1940s, according to the development of modern computers and programming languages, AI research in games started and the Minimax algorithm [79, 10] was republished.

Usually, the purpose of such research is to make strong computer players using AI methods [32, 65], because such strength is needed as the base for entertaining human players, and further it is a great challenge for AI researchers to make a computer player which can win against human champions. A lot of AI methods such as tree search, optimization, machine learning and reasoning have been proposed and improved, and computer players are now already quite strong in many games.

The most impressive event is the win of DeepBlue, IBM chess program, against human champion Kasparov in 1997 [12]. Since then, chess programs have continued to improve, with Elo ratings above 3200 for top programs since 2008, notably higher than human

grandmasters.

Logistello, a strong Riversi program [9], won every game in a six-game match against world champion Takeshi Murakami in 1997. Ponanza, a strong Shogi program, won against a professional player of Shogi [41], Shinichi Sato, in 2013. The best AI programs for Chess, Draughts, Backgammon, and Riversi can compete successfully with the best human players in the world.

At present, the game of Go is one of the rare board games where human professionals are still stronger, so it is a big challenge for AI in the near future. Monte Carlo Tree Search (MCTS) is a novel and effective approach for many games especially for Go, then widely used and studied. In March 2014, Crazy Stone, a strong Go program using MCTS, was rated as amateur 6 dan on KGS Go Server.

There are a lot of games, and they are categorized by using their characteristics such as the number of players, type of play, goal of the game and information. Many classical and popular board games such as Chess, Go or Shogi are two-player, alternative, zero-sum, deterministic and perfect information games.

There are many games for one player or three or more players such as Mahjong, for each class some other search methods are needed than two player games. Usually board games are zero-sum game: you win if your opponent lose. The strategy of zero-sum games consists of maximizing the probability of winning and then minimizing the probability of opponent winning, which is simpler compared to non-zero-sum game such as the prisoner's dilemma.

If both players know everything about the game state including the next state after an action, the game is called "deterministic and perfect information" [58, 75]. If state transitions are stochastic, or if state information is not perfect, the game is more difficult both for humans and computer players.

In this thesis, we employ a popular game "Reversi" for experiments. Like Chess or Go, this game also belongs to two-player, alternative, zero-sum, deterministic and perfect information games, .

1.2 The Game Tree Search

In this section, game tree search is introduced. This is the most essential component of almost all computer game players. The nodes of a game tree represent the board states, the root node represents the current board state and the leaf nodes represent the last board states where the game values (such as Black Win, Black Loss, Draw) are determined. The edges between nodes represent the possible moves (actions) from the upper nodes.

One of two players is assigned to each nodes to play, for example Black player or White player in the case of Go or Reversi. Assuming the game value is $+1$ when Black wins, -1 when Black loses, and 0 when Draw respectively, the Black player wants to maximize the value and the White player wants to minimize the value. In such case, the best strategy for the Black player is to choose the move that maximizes the score, while the White player chooses the move that minimizes the score. Such strategy can be applied in game tree search, and is called Minimax search [79, 10].

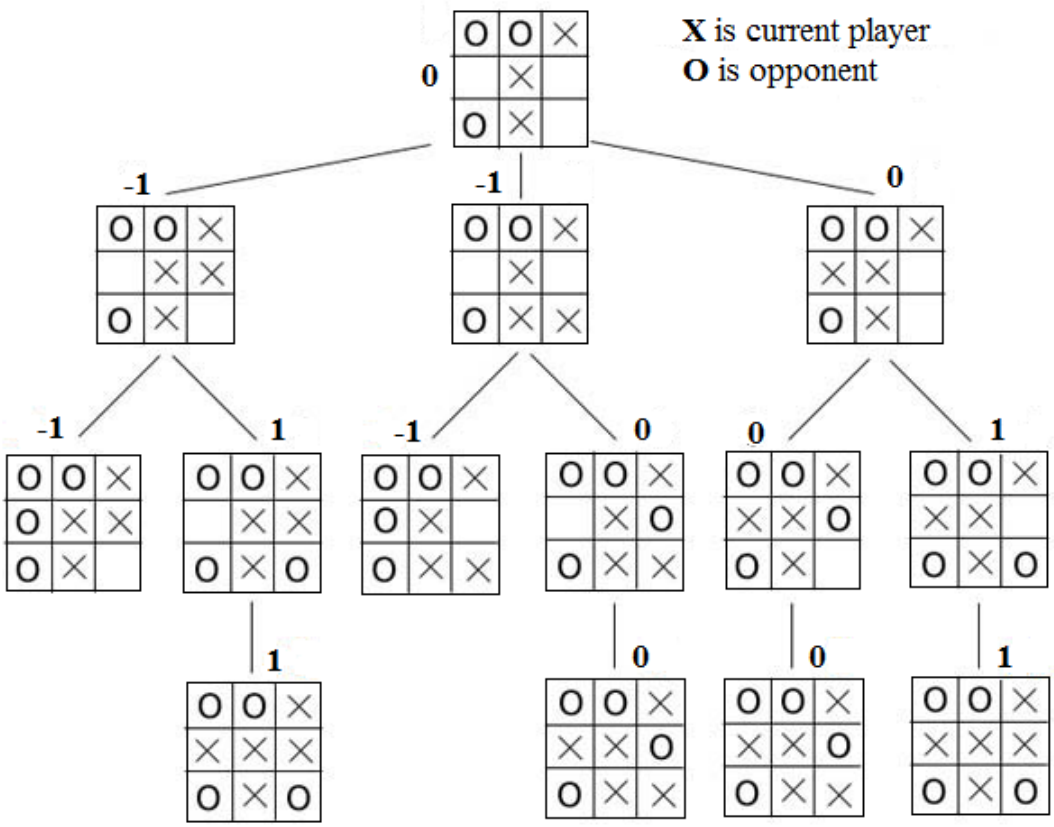


Figure 1.1: A game tree of Tic-Tac-Toe.

Figure 1.1 is an example of game tree of Tic-Tac-Toe, from a board where the 6^{th} moves have been played, to 6 possible leaf nodes. For example, player $\times$ should not move at (3,2) because then player **O** will move at (1,2) and player **O** will win. The most reasonable result of this tree is draw, value 0, and it is easily computed by extending such game tree.

However, if such game tree is wider and deeper, it becomes difficult or impossible to compute the result of the game and select the best move. The average number of edges at each node in the tree is called "branching factor", and the average number of moves to reach the leaf nodes can be called the "average depth". In Tic-Tac-Toe, the maximum number of possible moves is 9, and also the maximum depth is 9. The average depth of

Riversi is 60, because the board size is 8×8 minus the 4 default squares at the start of game. If the game tree is huge such as Riversi (10^{58} nodes), Lines of Action (10^{64} nodes), Connect6 [84] (10^{140} nodes), Amazons [48] (10^{212} nodes), Go 19×19 (10^{360} nodes), it is almost impossible to search the entire game tree.

When the average depth is big, leaf nodes with determined game values are usually not reachable during the tree search. In such case, search should be cut off at a certain depth, and the value (goodness) of the node should be estimated by using a "state evaluation function". Usually such state evaluation functions are designed by programmers using game specific knowledge so that high values are given for good board states, and low values are given for bad ones. In the case of Chess or Shogi, the number of pieces is an important factor of such functions, and the safeness of the King, the number of possible moves of pieces, are also important factors. Generally it is not easy to make a good state evaluation function by respecting so many factors [45, 46].

A big average depth can be managed by using a state evaluation function, but a big branching factor should be managed by another way. If there are many "possible, but not promising" moves, search of such moves should be avoided as much as possible, to reduce the total cost of tree search. Minimax search with alpha-beta pruning [53] is often used to prune such waste moves by using a branch and bound mechanism. For efficient alpha-beta pruning, it is known that the order of the possible moves is important, then often they are ordered by using an "action evaluation function". Sometimes, action evaluation functions are also used to limit the number of searched moves. For example, in some Go programs only about 20 moves in over 300 possible moves are searched [42, 16].

State evaluation functions are necessary to implement an alpha-beta search, but in some games such as Go, it is very difficult to make a good state evaluation function. Monte Carlo Tree Search (MCTS) is a novel framework which can be applied to almost all games without game specific knowledge like a state evaluation function. Evaluation value of a node can be computed by several Monte Carlo simulations where moves are randomly selected until the game ends, and the average of game values at the end of the simulations (usually, winning ratio) is used as the value of the node. Recently MCTS has been widely used [23, 29, 80, 2] especially for the game of Go. Almost all strong Go programs employ some variations of MCTS. MCTS gathers attention of many researchers because of its performance and its advantage that state evaluation function is not needed, and still there are many open questions on how to improve MCTS, for example which kind of simulations are better, or how exploitation and exploration should be managed in the tree search part of MCTS. Then MCTS is selected as the base method in this thesis.

1.3 Machine Learning in games

Machine learning is an area of artificial intelligence, and several different approaches of machine learning have been applied to games [74]. Reinforcement learning such as Q-learning [57] is one of machine learning methods, to learn through trials and errors how to act to maximize the expected reward. Here, only the reward is taught to the learner, the best action or the best state to transit is not taught. Reinforcement learning suits minor or new games, because it requires no prior knowledge, and reward can be calculated by their rules.

On the other hand, supervised learning is a learning where the questions and correct answers are given to the learner in prior. The dependencies between input (question) and output (answer) are learned to minimize the error, by using an input-output model such as a decision tree or a neural network. After the learning, if the function model and given data are enough, the learner can output frequently the correct answer even for new and unseen input. Supervised learning suits major games such as Chess or Go, because there are many **game records** available in such games. From the game records, we can obtain many pairs of a board state (input) and a good move (output) for the state.

Goodness of a move at a board state is not directly given by game records, but it is possible to consider that the selected move is better than the other possible moves in this state. By this idea, an action evaluation function [28] can be also optimized by supervised learning. Such action evaluation function is effective especially for MCTS framework, in the same way a state evaluation function is necessary for alpha-beta search. Even though MCTS can work without any knowledge specific to the target game, knowledge represented as an action evaluation function can enhance the search and simulation of MCTS. In particular, it can be used to bias the simulation to be more natural, to prune some hopeless branches, and to search more intensively hopeful branches [42].

One of the most famous successes of such supervised learning on games is Bradley-Terry model optimized by Minorization-Maximization algorithm (BTMM) applied to CrazyStone, one of the strongest Go programs [16] by Remi Coulom. Bradley-Terry model is an input-output function to compute the selection probability of a move, and its parameters (weights of features) are optimized by Minorization-Maximization algorithm using game records. The learned action evaluation function is used for pruning branches and for biasing simulations, it has been proved that such approach is promising if features are well designed. Then in this thesis also BTMM is employed to improve the performance of MCTS, and we focus on the design of features.

1.4 Features in board games

For machine learning, state or action itself is difficult to deal with directly, then only some values about some "features" are retrieved and used as the inputs. For example, when the health of a man is estimated by machine learning, *all* information about him will not be used, only a part of features related to health will be used, like height, weight, blood pressure or blood glucose level etc. There are many kinds of features for many approaches such as Speech recognition, Natural Language Processing, Computer Vision, Robot Control, Computational Biology, and Computer Games. Though there are also many kinds of features on each game, we focus on "pattern" of board cells, because it is used in many games such as Reversi, Connect-6 or Go.

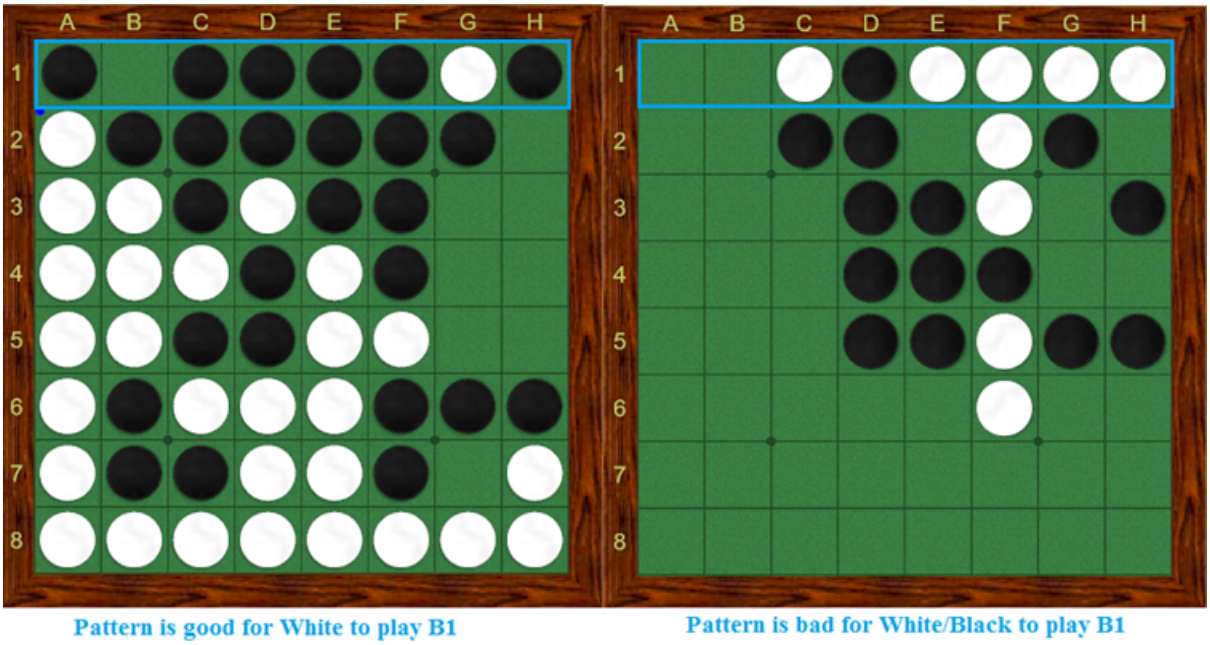


Figure 1.2: Good pattern and bad pattern.

Considering how human players of Reversi or Go evaluate the hopefulness of a move, it is very natural that the "patterns" of disks on neighbored cells are referred for the evaluation. Some patterns are very attractive to play for Black player (or White player), and on the contrary some patterns are not attractive at all. We can judge the hopefulness of a move by such patterns [16], at least partly. Two example boards about a position B1 are shown in Figure 1.2. In the left board, B1 is attractive for White player, considering only the pattern of 8 horizontal cells from A1 to H1. On the other hand in the right board, B1 is not attractive for both Black and White players, because after a move at B1, the opponent can occupy the important cell A1 in the corner.

In this thesis, Black disk is coded by 1, White by 2, and an empty cell by 0 respectively.

Then, 8 horizontal cells are coded by 10111121 for the left board, and 00212222 for the right board. Also, two other components are associated to a pattern, first the cell which is the target of evaluation, and second the player who should move next. So, a pattern is represented by a format (target position, player, states of cells). For example about Figure 1.2, (B1, Black to play, 10111121) is a good pattern, both (B1, Black to play, 00212222) and (B1, White to play, 00212222) are bad patterns.

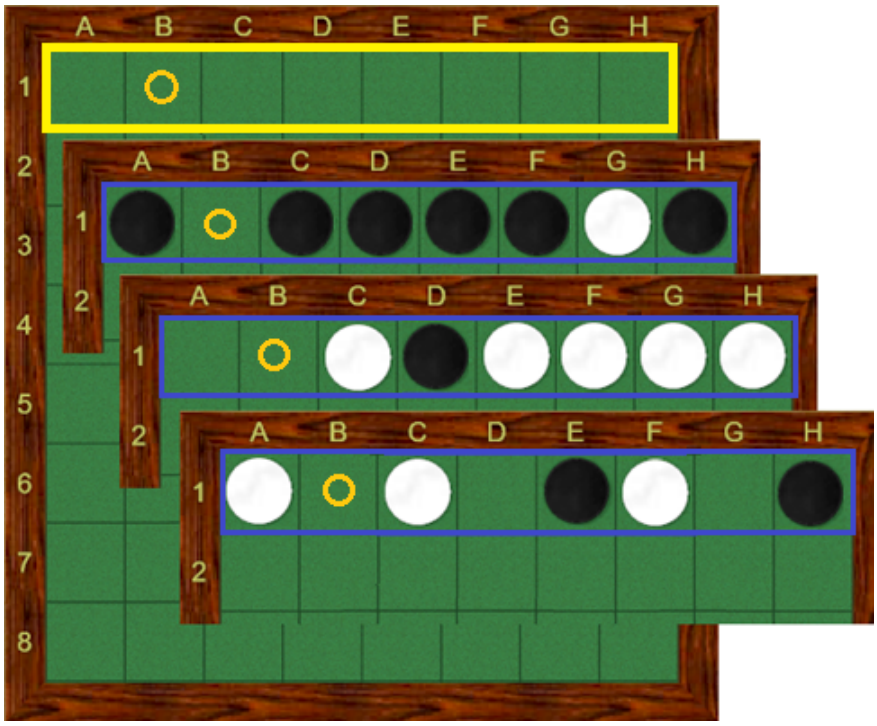


Figure 1.3: Pattern shape {B1,(A1,..,H1) and its patterns}.

In these examples, 8 horizontal cells from A1 to H1 are referred to evaluate B1 for each player, and there are many possible patterns on the 8 cells as shown in Figure 1.3. We call such a set of referred cells a "pattern shape", regardless to the actual status of cells (empty or occupied by a disk). A pattern shape is described in a format {cell to be evaluated, (cells)}, for example the case of Figure 1.3 is {B1, (A1, B1, C1, D1, E1, F1, G1, H1)}. The number of possible patterns in a pattern shape is $2 \times 3^{(length-1)}$, in this case 4374, where the "length" is the number of cells of the pattern shape, in this case 8. It should be noted that, multiple pattern shapes can be used to evaluate one target cell, for example {B1, (A1..H1)}, {B1, (B1..B8)} and {B1, (B1,C2,D3,E4,F5,G6,H7)} for the target cell B1.

There are many possible pattern shapes, distributed ones and concentrated ones, with few cells and with many cells, etc. It is clear that some pattern shapes can be good features for machine learning, and some other ones are not. For example, in Figure 1.4, cells in yellow window {A1, B1, C1, D1, E1, F1, G1, H1} will be a good pattern shape to

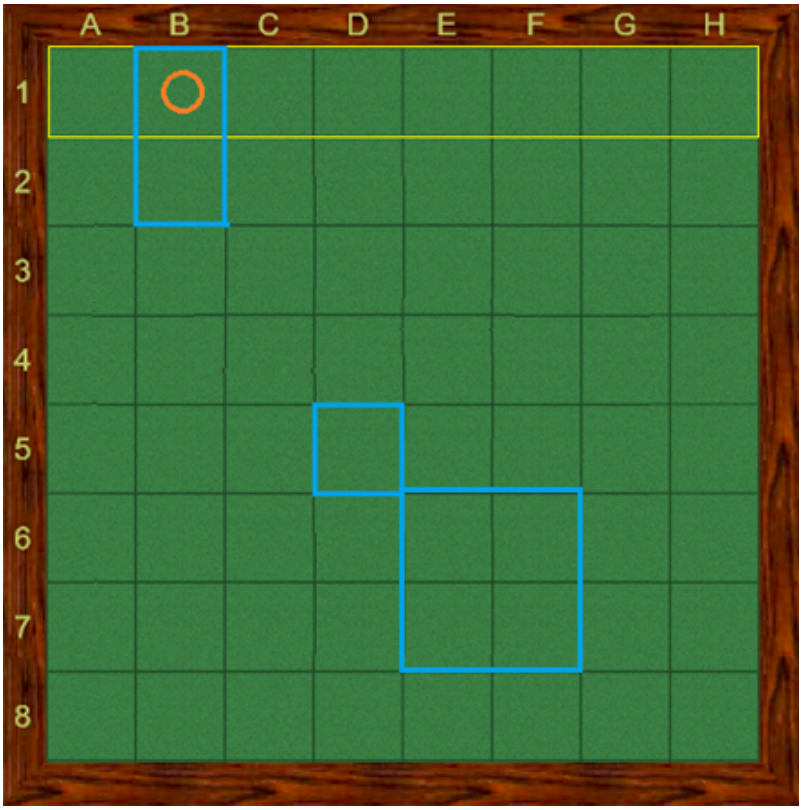


Figure 1.4: Good pattern shape and bad pattern shape.

evaluate B1, because we often refer such area. On the other hand, cells in blue windows {B1, B2, D5, E6, F6, E7, F7} will not be good pattern shape to evaluate B1, because we usually ignore such area for evaluating B1. However, generally it is not easy to distinguish which pattern shape is good and which pattern shape is bad for machine learning and game tree search.

The performance of machine learning depends on the set of features employed, in other words "feature selection" is important [68, 32, 13]. In many cases features are designed and selected by hand, sometimes they are optimized automatically. Also in the case of board games, feature selection is important for machine learning, then also for tree search using a learned evaluation function. Patterns are often used as features, and their shapes are usually designed by hand [11]. But in this thesis, **pattern shapes themselves are optimized** automatically, to improve the performance of BTMM, and then the performance of MCTS.

1.5 Research purposes and research outline

Feature selection is important for performance of machine learning and game programs using evaluation function, usually features are carefully designed by programmers. For

example, 10 types of features are used in CrazyStone [16], such as pass, capture, extension, self-atari, atari, distance to the last move, etc., and one of them is a pattern feature whose shape refers 36 cells (intersections since the target game is the game of Go). Remi Coulom, the author, stated in 2013 that almost all recent progress of CrazyStone came from the improvement of features, this fact means that design of features is efficient but very expensive, and the best features may be still not found. Thus, our **purpose** is to establish a method to obtain good pattern shapes automatically, and to reduce the cost as well as possible. As the benefits of this research, we expect to make it easier to design good pattern shapes with low cost even for new games.

In this thesis, limited kinds of methods and targets are employed, however we believe the benefits of our proposal method is not so limited.

1. Riversi is used as an environment of experiments in this thesis, but the method can be applied to almost all board games such as Go, Connect-6, Lines of Action, Hex, Shogi, where patterns are used.

2. Strength of computer player is measured and compared in this thesis, but evaluation functions are not only helpful for improving computer player performance, but also for creating "entertaining" player or "educating" player.

3. Evaluation function is used for pruning and biased simulation of MCTS in this thesis, but it can also be used for other tree search methods. For example, move-ordering of alpha-beta, or realization probability search [78].

4. BTMM is employed as the machine learning in this thesis, but other supervised learning algorithm such as Bonanza method [35] or back propagation of neural network [67].

Here, we summarize the general and specific elements of our work in Figure 1.5. Generally, game tree search is frequently used for making strong computer game players (1). State evaluation functions are often necessary to evaluate a node, and action evaluation functions [28] are also often used to enhance the search (2). Machine learning is quite effective to learn good evaluation functions automatically, compared to manual design (3), and it is known that selection of features used for machine learning is very important for the performance (4).

In our specific case, we employ Monte Carlo Tree Search (MCTS) as a game tree search for Reversi, because MCTS is novel and promising, Reversi is popular and simple (5). MCTS needs a good action evaluation function using some patterns (6), and Bradley-Terry Minorization-Maximization (BTMM) is employed as the learner because its performance is known (7). In our case only the pattern shapes are used as the features, but there are a lot of possible pattern shapes, then we optimize the set of pattern shapes (8).

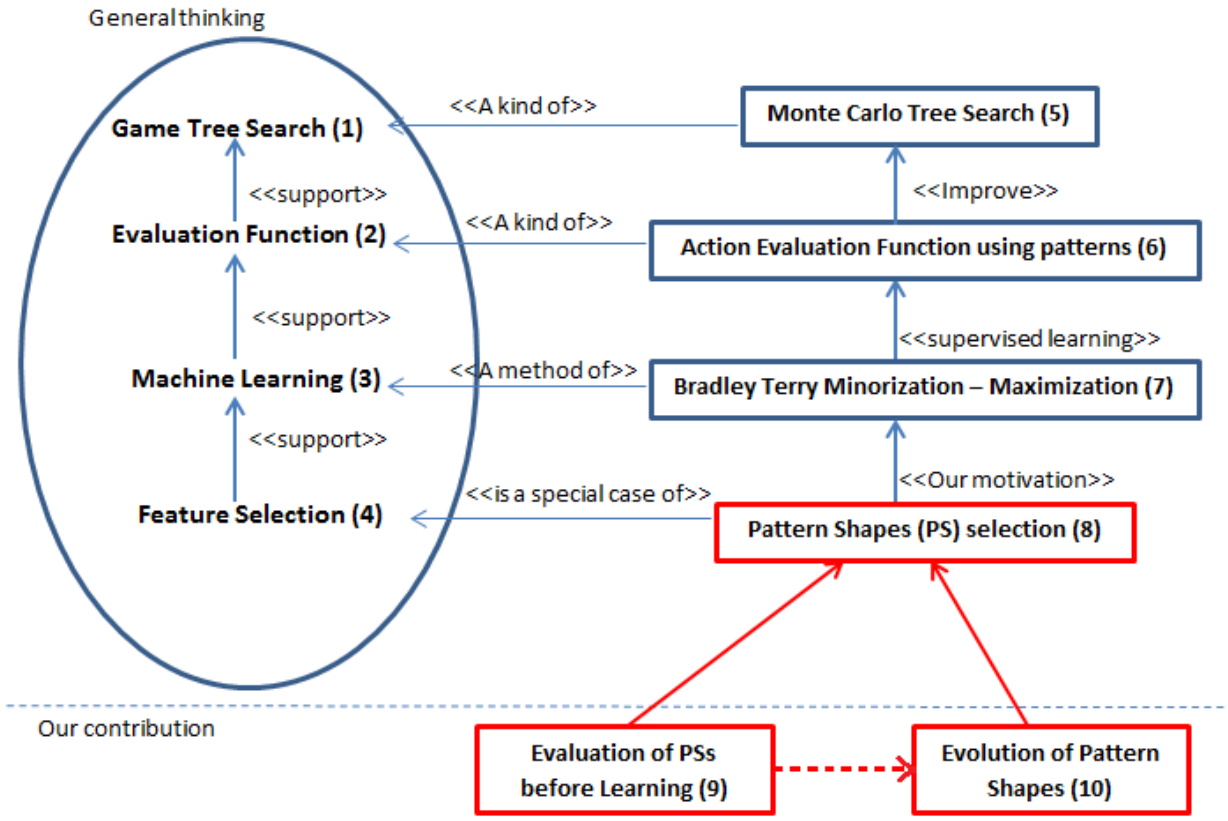


Figure 1.5: The related areas of our research.

There are many ways to select good pattern shapes. Our goal is to optimize the performance of MCTS, then it is straightforward to optimize PSs by using performance of MCTS, game results. However, this optimization will be too expensive, roughly it takes 5 hours for only one evaluation of a set of pattern shapes, if 100 games are done and 3 seconds are used for one move. Then, how to select good pattern shapes with reasonable cost is our **research question**.

We propose two approaches, one is to use statistic method to **measure the hopefulness of single pattern shape** before learning (9). Another is to evolve a set of pattern shapes through simulated annealing, by using early result of BTMM with smaller learning data (10). Finally the first approach (9) is used to enhance the evolution (10).

Here, we give more details about the two approaches.

[**Approach 1: Evaluation**] One of the important requirements of "good" pattern shape (PS) is that, there are preferred patterns to be played and also disliked patterns, because patterns are used to evaluate and order the possible moves. For example, suppose

that we have two pattern shapes for B1 as Figure 1.4.

- PS_A {A1, B1, C1, D1, E1, F1, G1, H1}: a pattern a_1 appears 100 times and B1 is selected 99 times, in other words preferred, a_2 appears 200 times but B1 is selected only 2 times, disliked.
- PS_B {B1, B2, D5, E6, F6, E7, F7}: a pattern b_1 appears 100 times and B1 is selected 30 times, b_2 appears 200 times and B1 is selected 61 times, b_3 appears 300 times and B1 is selected 89 times, in other words the "selected ratio" is similar.

We can assume that "a pattern shape is not important if the distribution of selected ratio of its patterns is flat". Under this assumption, a measurement of the importance of single PS is proposed, and PSs with lower importance values can be filtered (see figure 1.6).

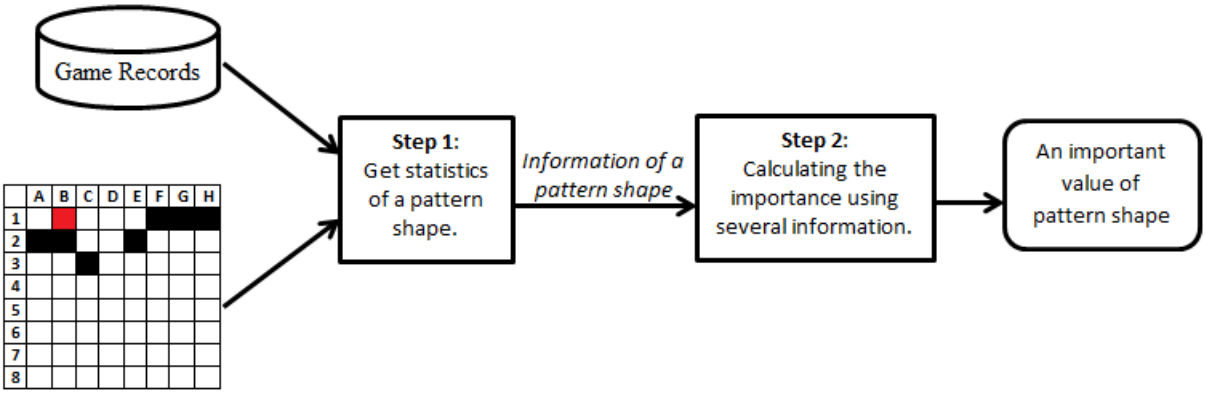


Figure 1.6: Process of pattern shape evaluation.

[**Approach 2: Evolution**]

Evolution is often used for optimization through trials and errors. The variables, in this case a set of PSs, are initialized, changed and evaluated then selected (see figure 1.7). There are many methods such as local search, simulated annealing or genetic algorithms to evolve the variables, but in this case the most important is how to evaluate the set of PSs. The straightforward way is to use the performance of MCTS, after learning of BTMM using the PSs, but this is expensive. Then, we assume that "if the result of BTMM is better, then the result of MCTS will be also better" and use BTMM performance for evaluation of PSs. Furthermore, we assume that "if the early result of BTMM using smaller data is better, then the final result of BTMM using full data will be also better" and use such "light" BTMM performance for evaluation of PSs.

Finally, the evaluation of approach 1 is used to help the evolution of approach 2. In the procedure of evolution, many candidates of new PS are generated, but a large part of

them are hopeless. Light-BTMM is still a bit expensive, then approach 1 is used to filter such hopeless candidates before light-BTMM.

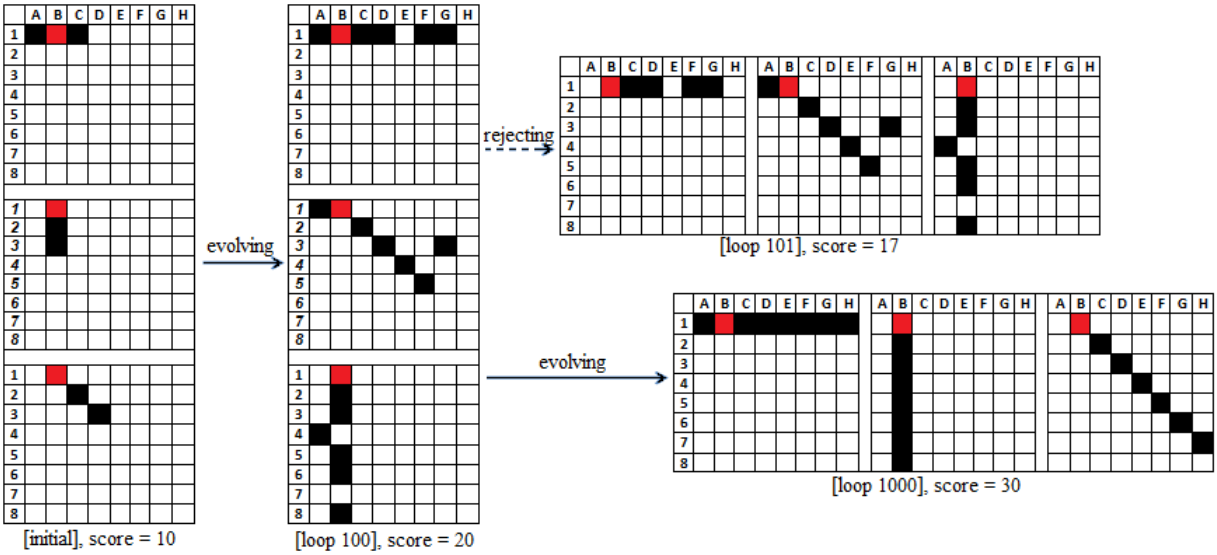


Figure 1.7: An example of pattern shape evolution.

1.6 Thesis Overview

The thesis has seven chapters: Chapter 1 is the introduction and outline of this thesis, Chapter 2 discusses about MCTS, it includes the development history, the basic algorithm and how to use evaluation functions. Chapter 3 mainly introduces how BTMM optimizes the parameters of game move features, and its efficiency on MCTS. Chapter 4 introduces some key points in Evolutionary Computation, and its techniques. Chapter 5 and chapter 6 are our contributions. Chapter 5 shows the evolution approach to find a good set of pattern shapes. Chapter 6 proposes a measurement to evaluate the importance of a single pattern shape, and how to help the evolution. Chapter 7 shows the conclusion of this thesis and future works.

Chapter 2

MONTE CARLO TREE SEARCH

Monte-Carlo methods are currently the best known algorithms for the game of Go. They have been applied to the game of Go since 1993 by Brugmann[8], and the Upper Confidence bound applied to Tree (UCT) was discovered in 2006 by Kocsis and Szepesvari [49]. At present, UCT and its variations are the mainstream algorithm of all MCTS programs. As the backgrounds of our research, this chapter introduces overview of MCTS, its advantages, and some extensions to MCTS by using game knowledge.

Monte Carlo Tree Search (MCTS) is a sampling-based search algorithm using Monte-Carlo simulations for estimating the expected winning ratio of a node (board state) [2]. The most important point of MCTS compared to classic tree-search methods such as alpha-beta and A* is that, MCTS does not need any game specific knowledge, or state evaluation function. Then, MCTS can be applied to many deterministic games with perfect information, with almost the same manner, and especially MCTS is effective in such games that it is difficult to make good state evaluation funcion, for example the game of Go. However it is also possible to introduce game specific knowledges to a part of MCTS. For example, the quality of simulations can be improved by selecting good moves with high probability by using an action evaluation function.

In 2006, Kocsis and Szepesvri proposed a method UCT that optimal move is guaranteed if sufficient time is given, and error probability is minimized even if the search is stopped suddenly. Two purposes, 1) to minimize the estimation error of winning ratio of promising nodes and 2) to minimize the risk of underestimation of not-so-tested nodes, are called as exploitation and exploration respectively. Generally exploitation and exploration are competitive about search resource, then it is necessary to balance the trade-off of them. UCT employs an equation (shown in 2.1, the detail will be explained in the next section) to deal the trade-off, and the effectiveness have been proved by many programs.

MCTS using UCT formula and its variants has been intensively studied as an effective framework for computer game playing. MoGo, a Go program using MCTS, achieved 1-dan level at 9×9 Go [30, 64, 63] in 2008. Fuego, an open-source framework for board

games and Go engine based on MCTS, beat a top human professional at 9×9 Go [23] in 2009. MoHex, a MCTS program built on the code base of Fuego, became the world champion Hex player [1] in 2009.

There are many techniques used in MCTS framwork such as Rapic Action Value Estimation (RAVE) algorithm using All Moves As First (AMAF) heuristic [29], Progressive Bias [14], Opening Books [52], and techniques using patterns. Patterns are useful for board games to prune hopeless nodes in search tree, to order possible moves, and to improve the quality of random simulations of MCTS.

Some MCTS programs uses the pattern techniques such as pattern matching [20, 80]. The pattern heuristic involves a table of patterns. Each pattern represents a 3×3 region of the board, with each cell marked as either black, white, vacant, or off-board. Associated with each pattern is a weight indicating the value of playing in the center.

2.1 Structure of MCTS

MCTS is a process that repeats the following four steps in a given time, shown in Figure 2.1. **Selection**, from the root node (the current board state) to a leaf node, a variation of moves (path) is selected to be evaluated more. **Expansion**, one child node is added to the leaf node of the selected path, unless the game ends at the leaf node. **Simulation**, one simulated game is played from the expanded node (board state), and the result of the game (for example win and loss) is determined. **Backpropagation**, the result is given to all nodes on the path by a backpropagation manner, and statistics such as the number of visits and the number of wins are updated in each node.

Through the iterations of these steps, game tree grows wider and deeper. As promising moves that has higher winning ratio are intensively selected in Selection step, then subtrees after such moves grows more wider and deeper, and estimation of winning ratio is more accurate. After the search, the most visited move (or sometimes the move with the highest winning ratio) at the root node is selected to play.

[**Selection**] For each play of a game, a limited thinking time such as 10 seconds is given to computer player to select the move among the set of legal moves. Usually it is not effective to search equally all the variations (paths) considering the limited thinking time. Then in the selection step, a path is selected to be evaluated, from the root node to a leaf node, according to the current tree and stored statistics such as winning ratio at each node. Here the balance between exploitation and exploration is controlled by a policy, in other words the promising moves will be frequently selected (exploitation), but the less promising moves still must be tried (exploration) to avoid underestimation.

Such balancing is also needed for Multi-Armed Bandit (MAB) problem [15, 77], where a gambler faces to many slot machines and plays many times, and decides which machine

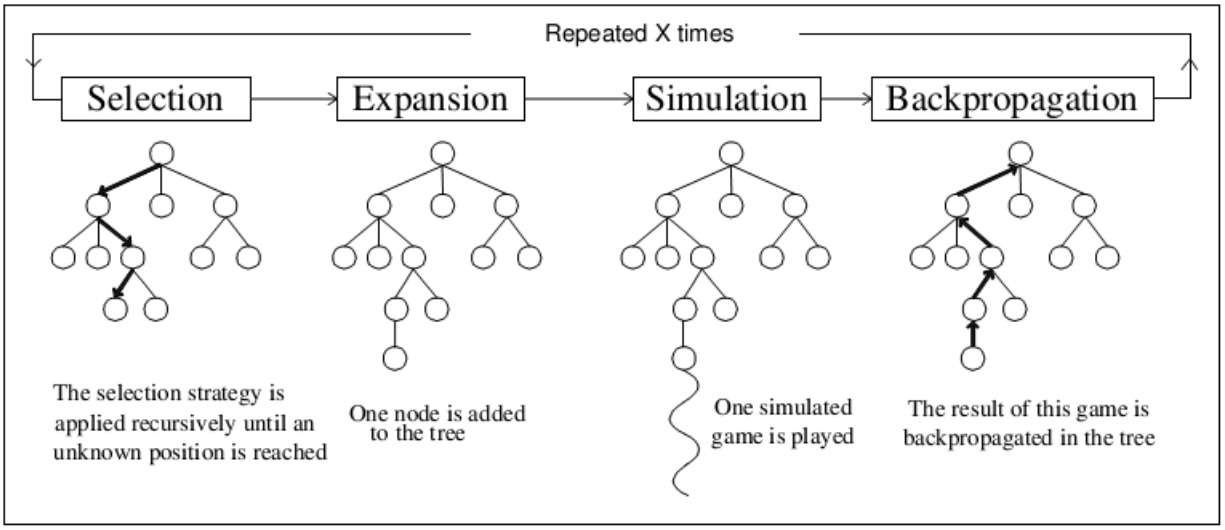


Figure 2.1: The steps of Monte Carlo Tree Search.

to play each time such that the final estimated reward is maximized. The slot machines look like the legal moves of a current node, because the true winning ratio (reward) of a move is unpredictable before many trials. To balance the exploitation and exploration for MAB, Upper Confidence Bound (UCB, equation 2.1) formula was introduced [77] and then imported to MCTS [49].

$$UCB_i = \frac{w_i}{n_i} + C\sqrt{\frac{\ln N}{n_i}} \tag{2.1}$$

Here $\frac{w_i}{n_i}$ is the winning ratio of child node i, w_i is the number of times that the player wins at the end of simulations after the node, n_i is the number of times that the node has been visited, N is the number of times that its parent node has been visited, and C is a tunable parameter. At the root node, the child node with the highest UCB value is selected, then next the children of the node is compared by the UCB formula and the best one is selected, repeatedly. The first term leads the exploitation, in other words child with high winning ratio tend to be selected. The second term enhance the exploration, in other words child with low number of visits tend to be selected. If C is low, exploitation will be emphasized, else if C is high, exploration will be emphasized, respectively. Some variations of UCB have been also proposed, such as UCB1-TUNED[31] or PBBM[17].

[**Expansion**] The process of expansion is to determine whether a given leaf node will be expanded by storing one or more of its children in UCT tree, but the popular expansion strategy is that one node is added per simulated game. The MCTS tree is grown up for each iteration of MCTS by the expansion step.

[**Simulation**] A simulation starts from the added node of the expansion step to the end of game, according to a "simulation policy". Completely random simulations can

be employed, however, the simulation policy is very important to improve the estimation accuracy of winning ratio (goodness) of a node, then to improve the performance of MCTS. Simply, to estimate accurately the goodness of a node, simulations should satisfy "if Black(White) is ahead at a node, Black(White) should win at the end of the game after simulation". If the quality of simulation is bad, even when Black is ahead White can win at a certain probability. Then, "realistic" simulations are desired, and some probabilistic model (containing heuristic knowledge) are often used to bias the selection probability of moves in simulations.

The computational cost should also be considered when using knowledge for simulation policy. Surely such knowledge or probabilistic model can improve the estimation accuracy of winning ratio, however, usually the computational cost will be more expensive, the number of possible simulations will be lower, and then the player may be weaker.

[**Backpropagation**] Backpropagation propagates the results of simulated games backwards from the leaf node to the root node by following the selected path. This step updates node statistics that will be used in selection step in future. The result is counted positively if the game is won, negatively if the game is lost, and averagely if the game is draw. The most popular policy of backpropagation is average counting of all simulated game results.

The more the thinking time is, the more the number of simulations is. The result of simulated game is usually back-propagated immediately, so it ensures all values of nodes are always updated after each iteration. Then, the algorithm can be stopped at any time to return the best estimated action, this is an advantage of MCTS compared to alpha-beta tree search with fixed depth.

2.2 Implementation of UCT algorithm

In this section the actual implementation of UCT algorithm is shown, referring a paper [7]. This algorithm can be used for any two players, determinisitic, discrete state and action and zero-sum board games. Each node v has four pieces of data: the state $s(v)$, the incoming action $a(v)$, the total simulation reward $Q(v)$, and the visit count $N(v)$. The result of function UCT_SEARCH(s_0) is an action possible at state s_0 that leads to the child with the biggest number of visits.

function UCT_SEARCH(s_0)
    create root node v_0 with state s_0
    while within computational budget **do**
        v := Selection(v_0, c)
        v_l := Expand(v)
        $reward$:= Simulation($s(v_l)$)
        BackPropagation(v_l, $reward$)

return the most visited child node of v_0

function Selection(v, C)

while v is not terminal **do**

$v := \text{argmax}_{v' \in v_{child}} \frac{Q(v')}{N(v')} + C\sqrt{\frac{\ln N(v)}{N(v')}}$

return v

function Expand(v)

choose $a \in$ random actions from $A(s(v))$

add a new child v' to v

with $s(v') = f(s(v), a)$

and $a(v') = a$

return v'

function Simulation(v)

while s is non-terminal **do**

choose $a \in A(s)$ uniformly at random

$s := f(s, a)$

return *reward* for state s

function BackPropagation($v, reward$)

while v is not null **do**

$N(v) := N(v) + 1$

$Q(v) := Q(v) + reward$

$reward := -reward$

$v :=$ parent of v

2.3 Characteristics of MCTS

MCTS framework is completely different from conventional tree search such as alpha-beta, and there are some positive characteristics that make MCTS a popular choice for many games [7].

Heuristic state evaluation function is not necessary. This is the most significant benefit of MCTS. Independence of domain-specific knowledge makes MCTS readily applicable to any domain that can be modeled by a tree. Although full-depth minimax is optimal in the game-theoretic sense, the performance in the case of depth-limited minimax depends significantly on the quality of heuristic state evaluation functions. For example, in the case of Chess, reliable heuristic state evaluation function have emerged after decades

of research, finally minimax performs admirably well. But, in the case of Go, useful state evaluation functions are much more difficult to formulate, so the performance of minimax degrades significantly.

Search can be stopped anytime. The game tree is built by using simulations. This ensures all values are always updated following every iteration of the algorithm. Then, the algorithm is able to return a reasonable action from the root at any moment in time. And if running for additional iterations, the result is more accurate.

Theoretical guarantees are given. The most important contribution of Kocsis and Szepesvri on UCT is that the regret (estimation error) is theoretically bounded, even in the case of nonstationary reward distributions. Also it has been shown through experiments that the probability of selecting a non-optimal action at the root of the tree converges to zero, at a polynomial rate as the number of games simulated increases.

Tree grows in an asymmetric manner. In the case of usual alpha-beta search, the search depth is fixed in an iteration, and tree grows from left to right with the same depth. In the case of MCTS, tree grows adaptively according to the hopefulness of each moves, in an asymmetric manner. In other words, the building of the partial tree is skewed toward more promising and thus more important regions.

2.4 Improvements of MCTS with knowledge

Though MCTS can work without any game-specific knowledge, many kind of enhancing techniques using knowledge have been proposed and recognized to be effective [16, 42]. In some kind of games such as Go, it is quite difficult to design good state evaluation functions, but it is not so difficult to design good action evaluation functions by using some features. In 2007, Coulom [16] introduced the Bradley-Terry model to optimize automatically the weights of features for an action evaluation function $f(a)$. And such action evaluation functions can be used in many ways for enhancing MCTS framework. In this section, three types of them are introduced.

[**Biased Simulation**] In the simulation phase, a move is selected from all possible moves. Default MCTS select a move randomly, with the same probability. However, as being already mentioned, more "realistic" simulation is usually better to estimate the goodness of a node accurately. Even when one action evaluation function is given, there are many action selection methods such as Roulette-wheel, tournament selection, Reward-based selection, Stochastic universal sampling, Rank-base selection. Coulom defined the probability that an action a is selected by $p(a) = f(a)/\sum_{a' \in A} f(a')$, using action evaluation function $f(a)$ learned by BTMM. Recently such simulations are widely used because the efficiency is significant, we also use this.

[**Progressive Widening**] The estimation of winning ratio in MCTS will be more

accurate when the number of simulation is higher. Then, if the branching factor (the number of possible moves at a state) is higher and the number of possible simulations is lower, the accuracy of winning ratio will be not good, then the performance of MCTS will be also not good. In such case, hopeless moves (with lower action evaluation values) among a lot of possible moves should be ignored, in order to concentrate the limited search resources to a limited promising moves. Progressive widening is such kind of techniques, this limits the number of moves searched to a smaller number for example only 2 or 5 at first, and gradually increase the number for example to 20, if the parent node is frequently visited [16]. The effect of progressive widening is significant in games with high branching factor such as Go, for example Nomitan can win about 60% of games against Fuego, but almost 0% if without progressive widening [42]. Unfortunately, in the case of Reversi we employed, the number of possible moves is not so high, then progressive widening is not so effective.

[**Biased Search**] In the case of progressive widening, a part of possible moves are searched fairly by using UCB formula, and the rest are completely ignored. However it is possible to add a bonus value to UCB of a move, if action evaluation value of the move is good, in other words the move is promising considering the game-specific knowledge. By using such bonus, biased search can be done so that promising moves are intensively searched. Many strong programs use some kind of bonus, for example Erica [69] and Zen [69], and the effectiveness is proved.

More particularly, Chaslot et al. proposed the following formula in 2009 [14],

$$UCB_{Bias1}(i) = \alpha \times \frac{w_i}{n_i} + \beta \times p_{rave}(m_i) + \left(\gamma + \frac{C}{\log(2+n_i)}\right) \times H_{static}(m_i) \tag{2.2}$$

where $p_{rave}(m_i)$ is the value of move m_i with the RAVE heuristic, and $H_{static}(m_i)$ is the evaluation value of move m_i based on a model learned offline.

In 2011, Shih-Chieh Huang proposed a similar formula in his Ph.D. thesis [37],

$$UCB_{Bias2}(i) = \alpha \times \left(\frac{w_i}{n_i} + C \times \sqrt{\frac{\ln n}{n_i}}\right) + (1-\alpha) \times p_{rave}(m_i) + C_{PB} \times \frac{1}{n} \times H_{static}(m_i) \tag{2.3}$$

By referring two formula 2.2 and formula 2.3, K. Ikeda and S. Viennot [42] proposed the formula.

$$UCB_{Bias}(i) = \frac{w_i}{n_i} + C \times \sqrt{\frac{\ln n}{n_i}} + C_{BT} \times \sqrt{\frac{K}{n+K}} \times P(m_i) \tag{2.4}$$

where C_{BT} is the coefficient to tune the effect of bias, and K is a parameter that tunes the rate at which the effect decreases. We choose formula 2.4 to bias the search.

2.5 For our research problem

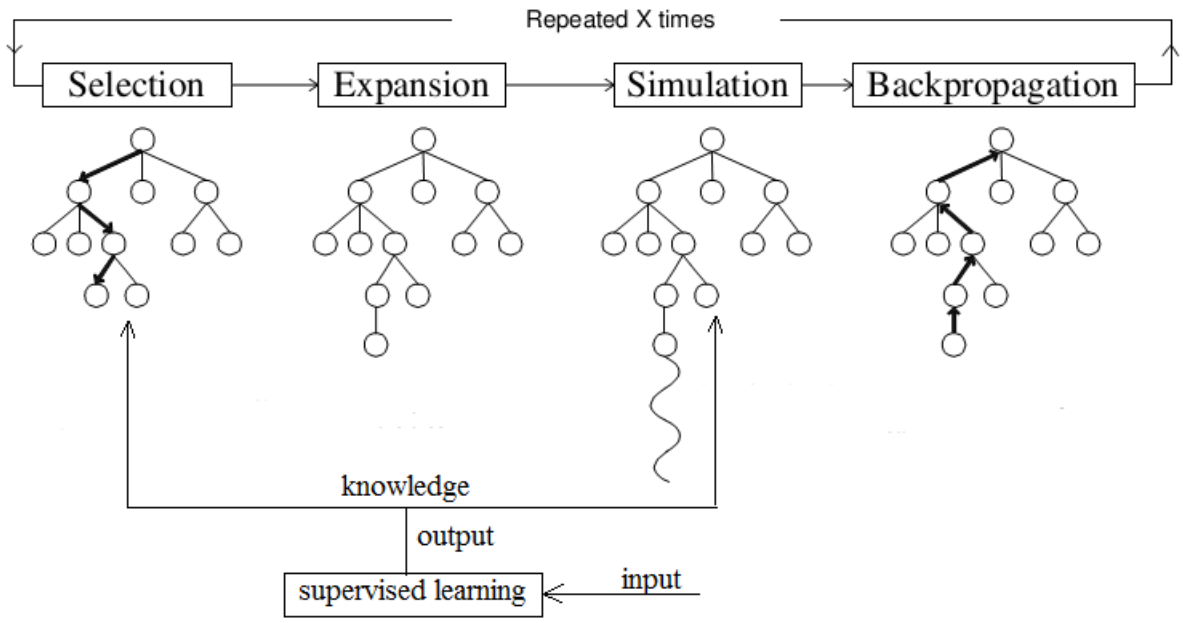


Figure 2.2: MCTS and enhancement using knowledge

Although MCTS is unified framework for many games, performance of naive and pure MCTS program is not so good compared to the improved ones. There are many possible improvements of MCTS, we focus on the utilizations of action evaluation function learned by BTMM. As mentioned in the previous section, knowledge can be used both for search part and simulation part of MCTS, as shown in Figure 2.2.

Working with MCTS, Coulom [16] introduced a supervised learning algorithm, BTMM, to learn effectively the coefficients of features from the game records. This algorithm and the overview of machine learning are introduced in the next chapter.

Chapter 3

MACHINE LEARNING AND BRADLEY-TERRY MINORIZATION-MAXIMIZATION

Like human learning from past experiences, computer systems can learn some relations between the given data. Machine learning is a very wide and popular concept of such algorithms, and has been used for computer games [3]. Our research focuses on supervised learning algorithms in board games, and this chapter presents an overview of them, and a specific algorithm called BTMM. BTMM algorithm is a new and famous method for weighting the features to make an action evaluation function.

3.1 Supervised learning

Supervised learning is a category of algorithms, each algorithm learns a relation between a given set of input variables $\{x_i \in X\}$ (also called input features) and a given set of output variables $\{y_i \in Y\}$, and applies the relation into a model to predict the outputs of new inputs. In other words, a predictive function is trained on a set of inputs and desired outputs (learning data). It is important that the examples are *given* by a supervisor.

Given a set of N training examples (or training set) which is represented by the form $S = \{(x_1, y_1), (x_2, y_2), .., (x_N, y_N)\}$, the task of supervised learning is to find a function $f(x) : X \to Y$ so that $f(x_i)$ is a "good" predictor of y_i, even for unseen examples (x', y'). Figure 3.1 shows the outline of supervised learning, at the first the training set is given, then supervised learning algorithm is done and the function f is learned, finally f can be used to estimate the output y' for the input x'.

When the output Y is a set of discrete labels such as "YES or No", "Cat, Dog or Pig", the task is called a classification [59, 61, 18, 76]. On the other hand, when Y is a space

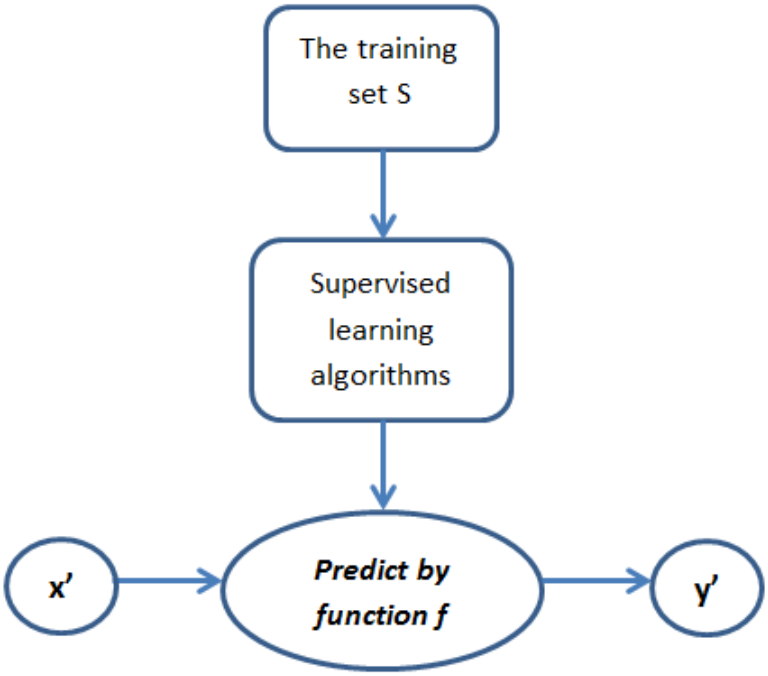


Figure 3.1: Supervised Learning.

of continuous values, the task is called function regression.

Usually, the following steps are proceeded to solve a supervised learning problem, though the order is not necessary to be kept.

1. To determine which kind of data to use as a training set.
2. To determine the input space X and output space Y.
3. To prepare the training set and test set.
4. To determine the learning algorithm and its parameters.
5. To run the learning algorithm on the training set.
6. To evaluate the accuracy by using the test set.

In step 2, sometimes only a part of given input values is used for learning. For example, suppose that we need to consider a classification task from personal information $x_i = \{age, weight, height, bloodpressure, name, phone, ...\}$ to his/her health $y_i \in Y = \{healthy, nothealthy\}$, it is clear that name and phone should not be referred in learning. Such selection of input information is called "feature selection" and recognized to be a necessary procedure before learning [68, 59].

There are many supervised learning methods proposed, but each of them has its advantages and weaknesses, it is impossible to select "the best" algorithm for all the problems. Especially, the function model should be carefully selected, to avoid both under-fitting and over-fitting [5].

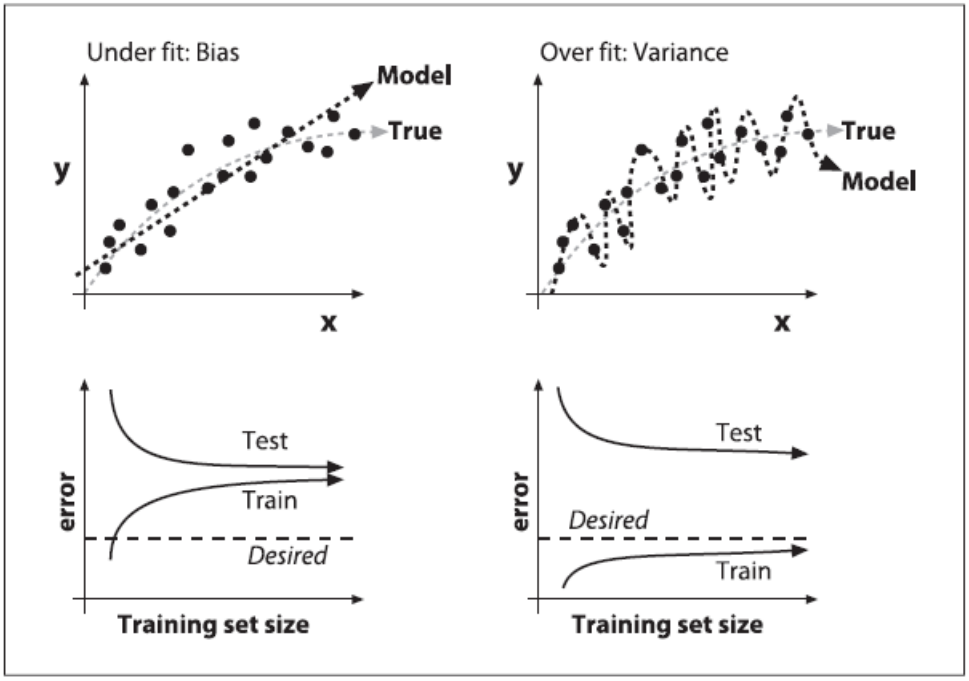


Figure 3.2: The bias and variance problems [5].

Figure 3.2 shows an example of under-fitting (left side) and an example of over-fitting (right side). The target problem is a function regression from a value x to a value y, the true relation of inputs and outputs is drawn by a silver broken curve (top side), and many noisy samples are given (dots, top side) to learners as the learning data. If a very poor model such as a linear function $f(x) = ax + b$ (left top) is employed, the model cannot represent the given data well, then the prediction error is higher than the desired level (left bottom, under-fitting). On the other hand, if a very rich model is employed, the model may be able to represent all the given data with no error (right top), but the performance for unseen data will be much worse (right bottom, over-fitting), because the learned function fitted too much only to the given samples and lost the generalization ability.

Considering the over-fitting problem, it is necessary to employ "test data" not used in learning phase but used only in evaluation phase. In the simplest case, learning data and test data can be prepared by dividing the original data set in two. However in this case the performance strongly depends on the way of division, sometimes lucky and sometimes unlucky. Then, statistical sampling called cross-validation [50] is often used

for more accurate evaluation of performance. There are several cross-validation strategies proposed, such as K-folding, repeated-random sub-sampling, and leave-one-out.

3.2 Pattern and Supervised Learning

It is well known that feature selection is effective for supervised learning [56], because too poor set of features causes under-fitting, too rich set of features causes over-fitting and slower calculation. Thus, optimization of features is necessary for supervised learning algorithms. Pattern is also the target of optimization, because pattern is one of the most frequent and important features.

Coulom proposed BTMM [16] using many features including patterns to evaluate the goodness of each action, and this approach is also employed in this thesis, but there are many related works. Bouzy and Chaslot [4] used a pattern of Go including a selected move and the neighboring states to predict which pattern is attractive to move. They used a Bayesian method to compute the selection probability of each pattern in the given game records, in order to determine good patterns and bad patterns to move. Stern and Herbrich [73] calculated a probability distribution over legal moves of a given position in Go by using a Gausian belief to define a full Bayesian Ranking model. This method used some Go features of a move such as patterns, liberties of new chain, liberties of opponent, Ko, atari escape, distance to edge.

State evaluation functions are also designed by using features including patterns. Michael Buro [11] used the combination of many patterns related to a legal move, and the pattern weights are optimized by using linear regression.

Usually, the whole board is too wide and rich to refer to evaluate a state or an action, then local pattern shapes of limited regions such as 5×5 or 3×3 are frequently used. Silver et al [71] said that the good pattern shapes can maximize efficiently the tactical advantage. Expert players usually refer many kinds of patterns such as "joseki" and "tesuji" to evaluate the legal moves for selecting the best move in Go, and such patterns have many different scales and specific location on the board.

How wide pattern shapes should be used, this is very difficult and essential question. Figure 3.3 shows a target cell to be evaluated (red) and some related cells to be referred (black). It is possible to consider "solo" pattern shape with these 21 black cells to evaluate the red cell, such wider pattern shape referring many information at once is sometimes called "all-mighty pattern". However, if many cells are referred, many possible patterns exist (in this case 2×3^{21}), then there will be only few examples in the given game records for each pattern. In such case, the generalization performance will be not good by the over-fitting problem, because the model is too rich. Then, frequently "several" pattern shapes with fewer cells are used in combination.

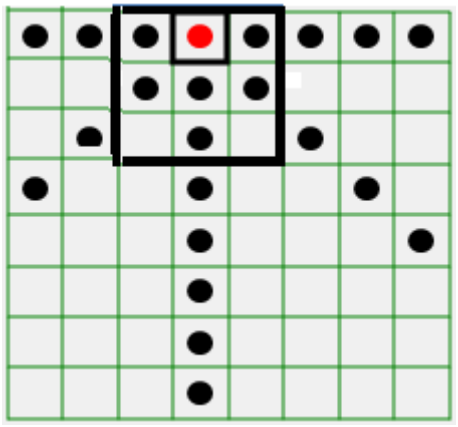


Figure 3.3: An example of all-mighty pattern in Riversi.

Also in this thesis, we use several pattern shapes, instead of using one wide pattern shape, for each position to evaluate its goodness. This way is similar with the way of Coulom [16], he considered that a move is as a team of 10 features including pattern. Figure 3.4 is an example, where position B2 is evaluated (left), and two pattern shapes are used, horizontal one (center) and vertical one (right). The goodness of these two patterns is different, and combined to one value in a way shown in the next section.

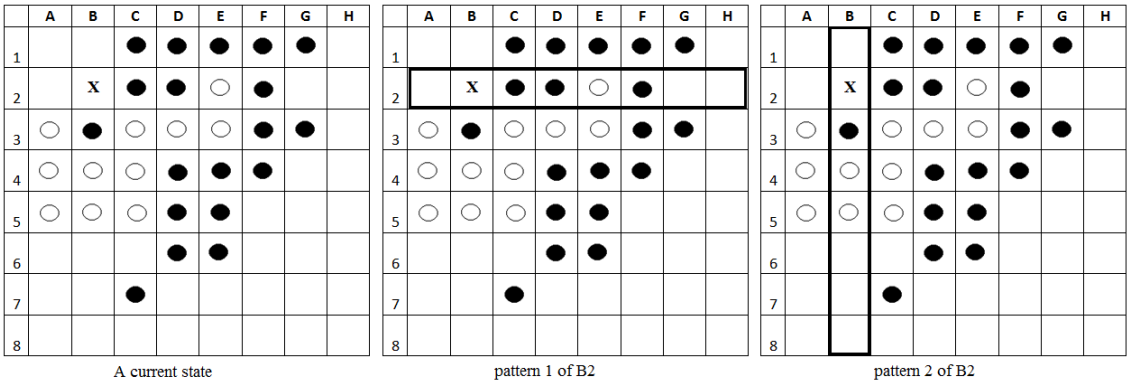


Figure 3.4: The features related to the move B2.

3.3 Bradley-Terry Minorization-Maximization (BTMM) model

Coulom employed Bradley-Terry ranking model to evaluate and order the moves, and applied a well-known optimization algorithm Minorization-Maximization to it [16], this is called BTMM. BTMM [16] is a supervised learning algorithm that is very suitable for pattern learning in board games.

3.3.1 Bradley-Terry model

The Bradley-Terry model assigns a parameter γ to be a strength of an individual of a competition. In the simplest case, the winning probability of a player i against j is calculated as follows:

$$P(i) = \frac{\gamma_i}{\gamma_i + \gamma_j} \tag{3.1}$$

In case of a competition having many individuals, the winning probability of individual i among N players can be calculated as follows:

$$P(i) = \frac{\gamma_i}{\gamma_1 + \gamma_2 + .. + \gamma_N} \tag{3.2}$$

When two or more players cooperate as a "team", the strength of the team is calculated by multiplying the strengths of individuals. Then, in the most general case, the winning probability of a team among many teams can be calculated as follows:

$$P(2-4\_wins\_against\_1-2-5\_and\_1-3-6-7) = \frac{\gamma_2\gamma_4}{\gamma_1\gamma_2\gamma_5 + \gamma_2\gamma_4 + \gamma_1\gamma_3\gamma_6\gamma_7} \tag{3.3}$$

The Bradley-Terry model can be used for evaluating the strength (goodness) of a legal move in a board state, and further, the winning probability of the move to be selected. Each move is considered as a team of members, and members are the features related to the move, such as patterns.

In a previous figure 3.4, B2 is a legal move of a current state, horizontal pattern shape (center) and vertical pattern shape (right) can be used as the features related to B2. When the next turn is black, we can see pattern1 $(B2, Black, 00112100)$ and pattern2 $(B2, Black, 00122000)$. Let γ_1 and γ_2 be the strengths of pattern1 and pattern2, then the strength of team B2 is $\gamma_1 \times \gamma_2$.

3.3.2 MM algorithm for Bradley-Terry model

As figure 3.5, BTMM is a supervised learning algorithm where MM algorithm is applied into a Bradley-Terry ranking model in order to optimize the parameters of relevant features such as patterns of a move. When game records G are given, and a set of features is determined, the strengths of every patterns can be learned automatically by MM optimization procedure, then an action function evaluation function f is learned. After all available moves are evaluated, a winning move can be randomly selected under the probabilities like Equation (3.3).

Formally, the probability that a move m_j is selected can be written as follows:

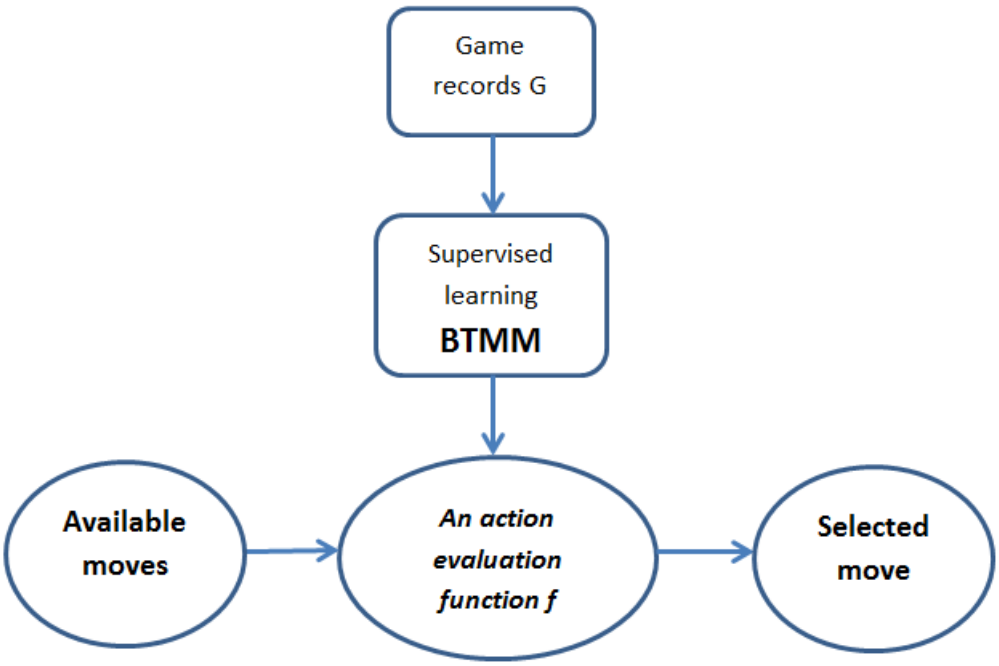


Figure 3.5: BTMM and action evaluation function.

$$P(m_j) = \frac{\prod\limits_{feature_i \in m_j} \gamma_i}{\sum\limits_{m \in legalmoves} \left(\prod\limits_{feature_i \in m_j} \gamma_i \right)} \tag{3.4}$$

BTMM computes the parameters γ such that the likelihood $L(\{\gamma_i\}) = \prod_{j=1}^{n} P(m_j)$ is maximized. Coulom proposed a method based on minorization-maximization algorithms [16], here each parameter γ_i can be tuned as follows:

$$\gamma_i = \frac{W_i}{\sum\limits_{j=1}^{N} \frac{C_{ij}}{E_j}} \tag{3.5}$$

where γ_i is the parameter of pattern i, W_i is the occurrence number of feature i in all moves of game records, C_{ij} is the product of all patterns which are teammates of pattern i in current state j of game board, E_j is the sum of strengths of legal moves at current state j. N is the number of positions found in the game records.

If all the parameters are initialized to 1, and the number of features for each competition is the same, the *first* iteration of minorization-maximization computes the winning frequency of each feature. This fact provides a Bayesian justification of frequency-based pattern evaluation like the method used by Bouzy and Chaslot [4]. But further, by re-

peating more loops of equation 3.5, better parameters can be achieved.

The objective function of MM algorithm is likelihood $L(\{\gamma_i\}) = \prod_{j=1}^{n} P(m_j)$, but usually mean log evidence (MLE) in equation 3.6 is used to show the performance of training or testing results of BTMM algorithm. A bigger MLE on the testing data means that better values of the parameters were obtained.

$$log - likehood = \frac{\sum_{i \in M} (log(prob(m_i)))}{|M|} \tag{3.6}$$

$$prob(m_j) = \frac{strength(m)}{E_j} \tag{3.7}$$

$$strength(m) = \prod_{i \in m} (\gamma_i) \tag{3.8}$$

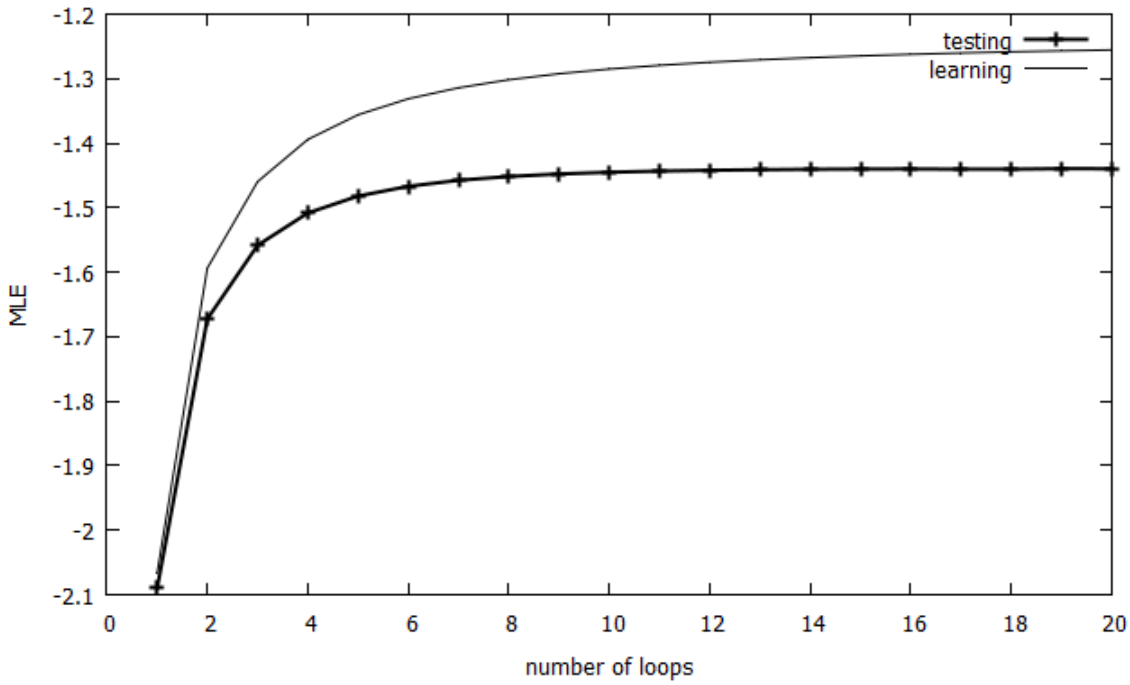


Figure 3.6: An example of learning/testing curves by equation 3.6 for BTMM alg.

Figure 3.6 shows an example of "learning curve" and "testing curve" for parameters of features γ_i. The y-axis is the values of MLE, the x-axis is the number of loops for optimizing the parameters γ_i. We can see that MLEs for both data improves gradually loop by loop, and also the difference between MLE of learning data and that of testing data is increasing, because of over-fitting. It should be remembered that sometimes MLE of testing data starts decreasing after some loops.

3.3.3 Working with patterns by BTMM

With patterns in board games, BTMM will learn the goodness of patterns from the huge number of game records, and Bradley-Terry model provides the probability distribution for legal moves of each board state.

In the paper of R. Coulom [16], he said that the prediction rate obtained with minorization-maximization and the Bradley- Terry model was the best among those which had been published in academic papers. The work by Stern, Herbrich, and Graepel [73] with more games and pattern shapes was compared to BTMM, at the beginning of the game its performance was better than BTMM, but at the middle game BTMM was better. The reason of this tendency is probably that, their work learned standard opening sequences, and BTMM learned more general rules.

R. Coulom also said "Despite the clever features of this pattern-learning system, selecting the move with the highest probability still produces a terribly weak Go player. It plays some good-looking moves, but also makes huge blunders because it really does not understand the position. Nevertheless, the domain knowledge contained in patterns is very precious to improve a Monte-Carlo program, by providing a good probability distribution for random games, and by helping to shape the search tree.", and he discussed in detail about using patterns in random simulations and progressive widening strategy of MCTS programs.

3.4 For our research problem

The base algorithm of MCTS and how to use an action evaluation function have been introduced in Chapter 2, and how to learn an action evaluation function has been introduced in this Chapter 3. We employ standard MCTS framework, standard two usages of action evaluation function using patterns, and standard BTMM algorithm for learning the weights of patterns. Then, originality and contribution of our work is the feature selection. In other words, our problem is an optimization of pattern shapes. The selection of features, removing redundant and irrelevant features, are important to improve the quality of general machine learning methods, and BTMM algorithm is not an exception. From that, the computational cost will be reduced, and the over-fitting problem will be eased.

Pattern shapes are optimized by two approaches, evolution and evaluation. The evolution of pattern shapes is introduced in chapter 5, and evaluation of a pattern shape is introduced in chapter 6. As the optimization method of pattern shapes, evolutionary algorithm (EA) is considered as a background in our research, then introduced in the next chapter.

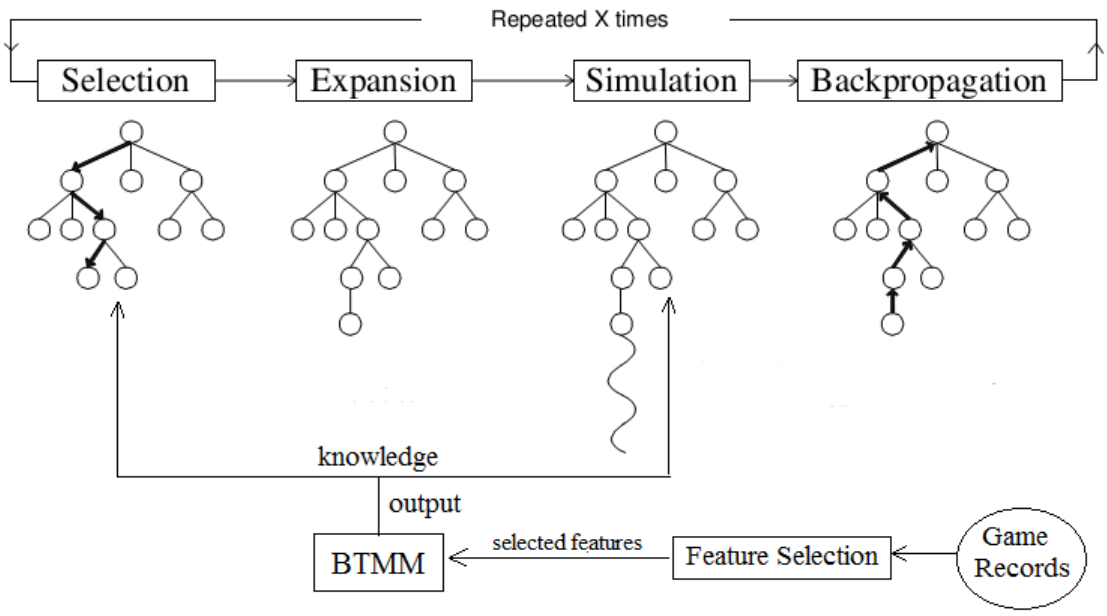


Figure 3.7: Selecting the good features before learning.

Figure 3.7 shows the role of BTMM, a machine learning algorithm, in learning the action evaluation function. This function is used in Selection phase and Simulation phase of MCTS. And, Feature Selection is used to select the relevant features before learning.

Chapter 4

STOCHASTIC OPTIMIZATION AND EVOLUTIONARY COMPUTATION

Our goal is to make strong computer players of board games, in particular of Reversi. For this goal, we employ Monte Carlo Tree Search (MCTS) supported by an action evaluation function, as discussed in Chapter 2. The action evaluation function is learned by Bradley-Terry Minorization-Maximization (BTMM) algorithm, where patterns are used as the input features, as discussed in Chapter 3. There are many possible pattern shapes, and the selection of them strongly affects the performance of BTMM as well as MCTS. In this thesis, we propose to optimize the pattern shapes with a stochastic optimization algorithm.

This chapter shows a general introduction to stochastic optimization and its sub area, evolutionary computation.

4.1 Introduction

Considering a parameter $x \in X$ and a function $f(x) \in R$, an optimization task is to find the best parameter x^* among the legal parameters $X' \subset X$, so that $f(x^*)$ is maximized (or minimized sometimes). There are many optimization methods such as gradient method, linear programming, tabu search, simulated annealing and genetic algorithm.

Evolutionary computation is a category of such optimization algorithms, including genetic algorithm. In the real world, limited resources such as food and space make the individuals such as animals compete for those resources. This will select the individuals that are better adapted to the environment, only the surviving individuals can generate new individuals by recombination and mutation. Again, the new individuals are evaluated

by a fitness function for survival. Repeating such process, individuals change their forms and characteristics gradually so that the fitness function is maximized, this process is called evolution. Evolutionary computations imitate such evolution on a computer, in order to find good parameters in an optimization problem.

Evolutionary computation [81, 27, 26, 47] is considered as a kind of "generate and test" algorithms, and usually they are stochastic, and keep some individuals (solutions) as a population for diversity. The recombination and mutation operators create the new candidate solutions by using the existing good solutions, and only the good solutions evaluated by the given fitness function are selected. Evolutionary computation consists of some elements, population, mutation, reproduction (crossover) and selection (alternation).

The general structure of simple evolutionary algorithms is depicted in algorithm 4.1. and Figure 4.1.

Algorithm 4.1: Evolutionary Algorithm
Initial population (several individuals, solutions) is generated
while termination condition not met **do**
    Individuals are evaluated
    Parent individuals are selected
    New individuals are generated by using the parents and evolution operators
    Survivors to the next generation are selected
end while
return best individual in the population

[**Representation of evolved variables**] The target variables to be optimized depends on the target problem, sometimes just a real value, sometimes a vector, sometimes a picture image, or sometimes a program code. Many types of representation of individuals can be dealt in evolutionary algorithm, and the type of evolutionary operators is selected regarding to the type of representation.

For example, if solutions are represented in a binary string, a genetic algorithm is normally used. If the solutions are represented in LISP trees, a genetic programming is used. The real-value vectors representation is usually used for evolution strategies, and the finite state machines representation is usually used for evolutionary programming.

In our case, the target variables are pattern shapes, there is no standard method for such variables.

[**Selection**] How to select the parents, and how to select the surviving solutions, are called "selection", and usually they are independent from the representation, opposite to the case of evolutionary operators. The simplest case is to select two solutions as a parent randomly, to let the best-n solutions survive. There are many other methods for example Roulette-wheel, tournament, reward-based, stochastic universal sampling, or rank-based

selection, and each method has advantages and disadvantages. We can choose one of them according to the purpose and resources.

In such selection, it is important to regard not only the fitness function but also the diversity. If almost all solutions are the same, it will be difficult to find better solutions apart from the solutions. Such problem is called "premature convergence to a local optimum", wide diversity can reduce the risk of it.

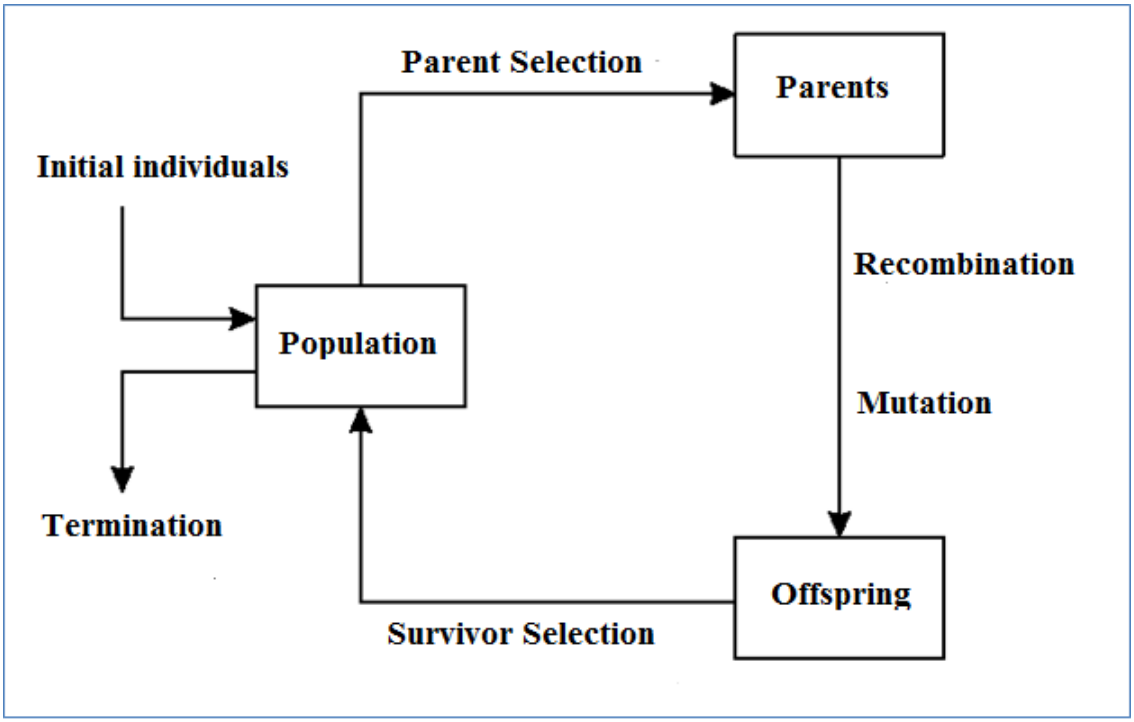


Figure 4.1: Process of evolutionary computation [21].

The genetic operators are used to generate new candidate solutions, by using information of existing solutions. Here, it should be noted that completely new solutions are not generated randomly, because the average quality of such solutions would be lower than that of existing, surviving solutions. Operators can be divided into two types according to their number of input solutions. If the number of inputs is 1, it is called "mutation" operator. If the number of inputs is greater than 1, it is called "recombination" operator.

Mutation delivers a new solution from an input solution by changing a part of its parameters, then reduces the risk of premature convergence to a local optimum. For example, in the case of representing X as a binary string, some bits are flipped. In the case X is an Euclidean space, variables are moved in a radius. In the case X is a decision tree, a new node is added, an existing node is removed, the condition of a node is changed, or the value of a leaf node is changed. In our case, X is a pattern shape, in other words a set of board cells, mutation adds a new cell, removes an existing cell, or change the cell.

On the other hand, the recombination (crossover) is used for two or more inputs

(parents). It merges the information of parent solutions in a way. For example, in the case of binary strings, a part of the string of the new solution is selected from one parent, and the other part is selected from the other parent. There has been much debate about the importance between recombination and mutation, but usually they are both used in most programs.

4.2 Evolutionary Computation and Feature Selection

For feature selection, evolutionary algorithms are often used [22]. In this section an outline of them and related works are shown.

4.2.1 Feature Selection

As discussed in the previous chapters, feature selection is effective as the preparation of machine learning [13], to reduce the cost and the risk of over-fitting. Let $X = f_1 \times f_i \times f_n$ be the input space of given data, it means that each instance $x \in X$ has n attributes, in other words variables, features or dimensions. The feature selection problem is to find a subset $X' \subset X$ that maximizes the performance of works where such features are used, for example machine learning [6]. Usually, redundant and irrelevant features are removed, and a limited number of relevant features are kept

Feature selection differs from feature extraction, feature extraction consists in reducing the data dimensions *and* transforming the data (Figure 4.2 right), while feature selection consists in reducing the number of variables of data without transforming the data (Figure 4.2 left).

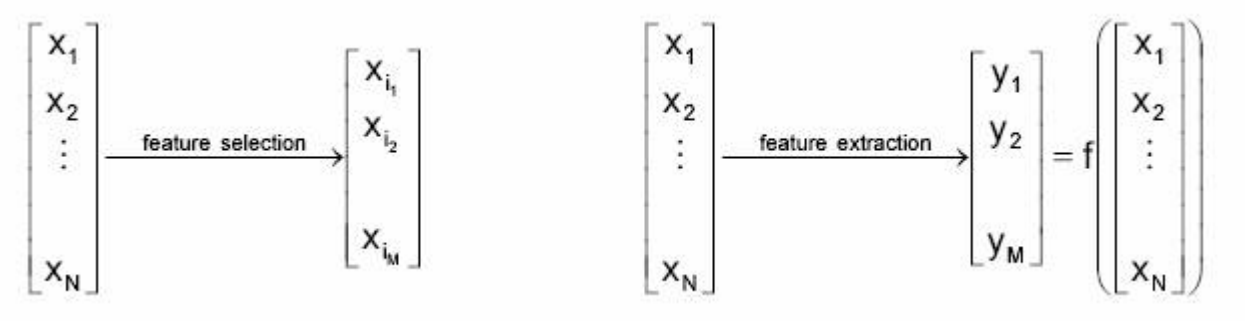


Figure 4.2: Feature Selection vs Feature Extraction.

If the number of possible features is small such as 4, it is easy to evaluate all the $(2^4 - 1)$ combinations of such features. But usually there are more possible features and a lot of combinations of them, so it is impossible to evaluate all of them. Thus, optimization methods such as genetic algorithm with trials and errors are frequently used.

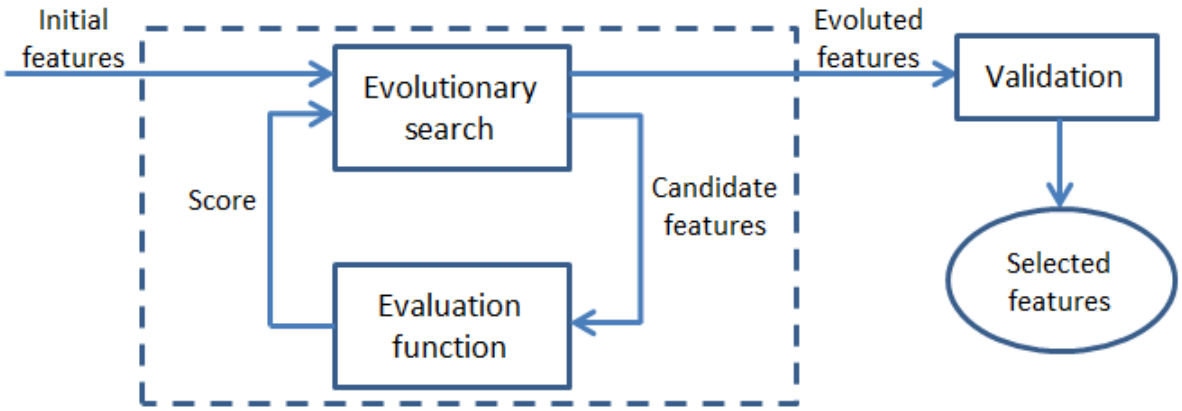


Figure 4.3: Evolutionary search for Feature Selection [59].

Figure 4.3 shows the outline of optimization of features using evolutionary algorithm. At first, optimized variables (search space) is decided [43, 33], and initial solutions (sets of features) are generated. According to the evolutionary algorithm, new candidates (new sets of features) are generated, and tested by an evaluation function. After the evolution, the best solution (set of features) is retrieved, validated and used.

Here, how to evaluate each set of features, in other words the design of evaluation function, is a difficult problem. This problem has been widely discussed, and many approaches are proposed [62, 60, 59]. Mainly, it is important to balance the cost for evaluation and the accuracy of the evaluation. In our case, the most straightforward approach to evaluate a set of pattern shapes is as follows:

1. A set of pattern shapes is given.
2. BTMM is executed to optimize the weights of all patterns, then an action evaluation function is gained.
3. MCTS program enhanced by the function is prepared, and several games are played against a fixed opponent.
4. The winning ratio of the MCTS program (using the set of pattern shapes) is returned.

Our purpose is to make a strong MCTS player, then this approach is accurate. However, it includes BTMM learning and playing many games, then the cost is quite expensive. So, other approaches with lower cost are often employed, though the accuracy is also lower. This trade-off is our main question to solve.

4.2.2 Feature selection in some fields

Feature Selection has been studied not only for games but also for many fields such as speech recognition, natural language processing, computer vision, robot control, or computational biology. And machine learning methods applied for such fields are not only supervised learning, but also unsupervised learning or reinforcement learning.

In Speech Recognition [82], the data is a speech signal, it is segmented into a series of speech frames. Number of frames in each second is calculated by the frame length and frame shift, this is the most cutting edge of speech recognition. The Mutual Information and its extensions are usually used to measure the relevant of feature with the particular classification problem. Some kinds of features such as Syllable, Articulatory-feature, stress-accent are selected from the speech signal.

In Natural Language Processing [83, 51], the data is text information, and the features are evaluated by some methods such as F-score, pointwise mutual information, information gain. Besides, the meaning of text data depends on the part of speech (POS), and the order of words in some languages. Feature selection is removing the terms that are the least informative according to some standards to reduce the dimensionality.

In computer vision [72], there is no any exact definition of feature, it depends on the type of applications. We can consider a feature as an interesting region of an image, and it is used as a starting point for many computer vision algorithms. There are many type of features in computer vision such as Edges, Corner, interest points, Blobs, and Ridges. Feature selection in computer vision basically reduces the semantic gap by adding the feature that best represents the query and removing redundant ones. The evaluation functions are usually the mutual information, entropy, or the inconsistency from the relevance feedback.

In Robot Control [54], we can see some kinds of controls such as Deliberative control, Reactive control, Hybrid control, Behavior-Based control. These kinds of controls are based on the thinking and reaction properties. Robot control is the collection of many machine learning approaches. For this reason, the features for Robot control are also multiform such as signal feature, spatial-temporal feature.

In Computational Biology [66], many computational techniques are used to model biological systems at various level of complexity. Working with biology, the top scoring features/genes are selected while the rest are discarded, and the ranking problem are usually used in Computational Biology. Besides, the space search of features is considered, so the optimization algorithms are used to find the relevance features with low cost.

4.2.3 Some Optimization Methods

As evolutionary optimization methods have been already introduced in Section 4.1, this section introduces some other stochastic optimization methods [34] such as Random Search, Hill-climbing, Simulated Annealing and then discusses about their advantages and disadvantages.

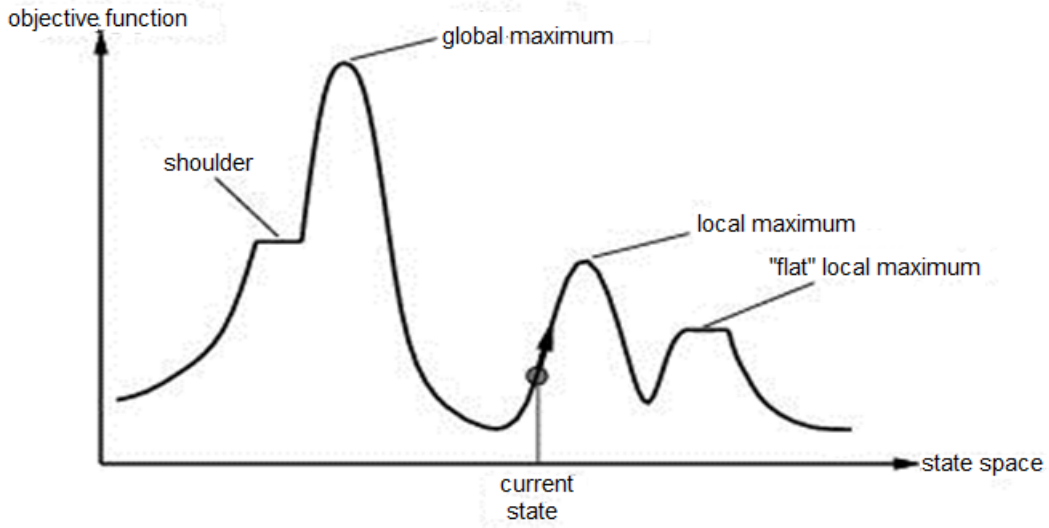


Figure 4.4: "Landscape" of evolutionary search.

[**Hill-climbing**] is an iterative algorithm that starts with a random solution, then tries to find a better solution by changing the solution a bit, step by step. If the change produces a better solution, the new solution is replaced, and such steps are repeated until no further improvements can be found.

Hill climbing is called a greedy optimization method [25]. When local maximum exist (tops of center and right mountains, Figure 4.4), hill climbing can finish the search at one of them. It means that the global optimum (left) is not guaranteed to be found. Hill climbing is easy to implement, and it is used widely in artificial intelligence. If the amount of time available to perform a search is very limited, Hill climbing may produce better results than other algorithms.

Figure 4.5 is the pseudo code of Hill-climbing.

[**Simulated Annealing**] The greedy optimization methods can get trapped easily at local optimum, the results of these methods might depend on the starting point. The greedy optimization methods always reject a new solution if its evaluation value is worse than the current. However, such new solutions should be sometimes accepted, to overcome the local maximum problem. In simulated annealing (SA), the probability of accepting the new solution is calculated based on a temperature parameter T [19, 55]. Figure 4.6 is the pseudo code of SA.

Hill-Climbing

```
function HILL-CLIMBING( problem) return a state that is a local maximum
    input: problem, a problem
    local variables: current, a node.
                     neighbor, a node.

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest valued successor of current
        if VALUE [neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

Figure 4.5: Hill-climbing pseudo code.

Simulated annealing

```
function SIMULATED-ANNEALING( problem, schedule) return a solution state
    input: problem, a problem
           schedule, a mapping from time to temperature
    local variables: current, a node.
                     next, a node.
                     T, a "temperature" controlling the probability of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE[next] - VALUE[current]
        if ΔE > 0 then current ← next
        else current ← next only with probability e^(ΔE/T)
```

Figure 4.6: Simulated Annealing pseudo code.

In the Simulated Annealing pseudo code, the meaning of δE is important to understand. Suppose that we have 3 available moves, with changes in the objective function of δE_1 = -0.1, δE_2 = 0.5, δE_3 = -5. (Let T = 1), select a move randomly.

- if δE_2 is picked, move there.

- if δE_1 or δE_3 are picked, probability of move = exp(d/T). Thus, probability of move 1 = exp(-0.1) = 0.9, it means that 90% of the time we will accept this move. probability of move 2 = exp(-5) = 0.05, it means that 5% of the time we will accept this move.

The temperature parameter T decreases as the algorithm runs longer. We can easily see that the acceptance probability is higher when T is higher. Thus, locally bad solutions can be accepted in early stage of SA, to avoid the local optimum. Such solutions are

rejected gradually, and the behavior becomes similar to hill-climbing.

It is guaranteed that SA always find the global optimum under a condition, with very slow schedule of decreasing T. In practice, limited time resource is given, then SA also can be trapped in a local optimum. However, SA has been widely used, because it is well known that usually SA is better than simple hill-climbing.

[**Genetic Algorithms - GA**] We have already discussed the elements of evolutionary algorithm in Chapter 4.1, and the pseudo code of genetic algorithm (GA) is shown in Figure 4.7. The main difference of GA from other algorithms is that GA keeps many solutions, both for keeping variety to avoid the local optimum problem, and for generating new solutions by using two or more solutions [70, 36]. By such mechanism, usually, GA is better than hill-climbing or SA, *if* sufficient search resource is given. However, in contrast, if the search resource is limited, GA never works well.

Another advantage of GA is that it is possible to generate "several solutions that are far different from each other". The fitness function of GA can be tuned so that not only the quality of a solution is regarded, but also the distances (difference) of the solution to other solutions are regarded. We published a paper to exploit this advantage, to produce "rival players for Reversi beginners" [44], please see Appendix 1.

Genetic Algorithm

```
function GENETIC_ALGORITHM( population, FITNESS-FN) return an individual
    input: population, a set of individuals
           FITNESS-FN, a function which determines the quality of the individual
    repeat
           new_population ← empty set
           loop for i from 1 to SIZE(population) do
                    x ← RANDOM_SELECTION(population, FITNESS_FN)
                              y ← RANDOM_SELECTION(population, FITNESS_FN)
                    child ← REPRODUCE(x,y)
                    if (small random probability) then child  ← MUTATE(child )
                    add child to new_population
           population  ←  new_population
    until some individual is fit enough or enough time has elapsed
    return the best individual
```

Figure 4.7: Genetic Algorithm pseudo code.

In the next chapter, evolution system of pattern shapes using SA, our main contribution, will be shown.

Chapter 5

EVOLUTION OF PATTERN SHAPES

One of our main contributions, evolution of pattern shapes, is shown in this chapter. One paper titled "Fast Optimization of the Pattern Shapes in Board Games with Simulated Annealing" has been accepted to the conference KSE2014 [40].

5.1 Components of Optimization

In this section, we propose an optimization method of pattern shapes (PSs) to improve the performance of MCTS. Figure 5.1 shows an image of such optimization. First, a very simple set of PSs is given as the initial set, its evaluation value is low, for example 10 (left). Through the optimization procedure, a better set of PSs can be achieved, its evaluation value is better, for example 20 (center). It is frequent that the new candidate of PSs is worse than the current one (right top, value is $17 < 20$), such one is usually rejected. Finally, after the optimization procedure, the set of PSs is optimized (right bottom).

For designing such optimization method, it is necessary to consider the following three components:

- Variables to be optimized
- Objective function for the variables
- Optimization method

There are many candidates for each component, it is an important research question to find which ones we should employ. Mainly, we should take care of the trade-off between the cost and the efficiency.

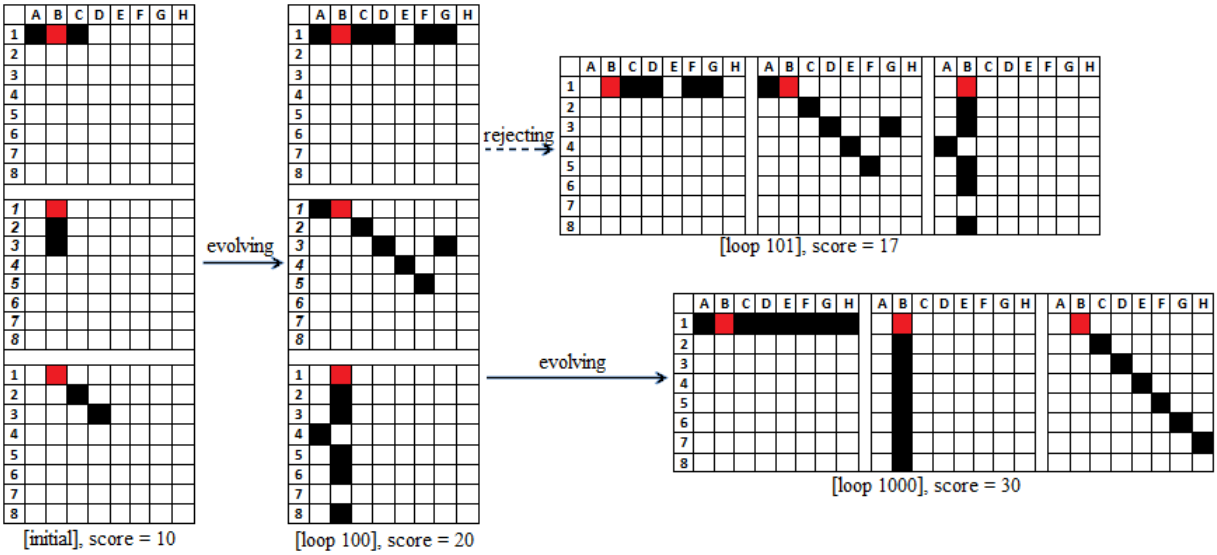


Figure 5.1: An example of evolution.

5.1.1 Variables to be optimized

Figure 5.2 (and Table 5.1 in Section 6.3) shows a set of 25 PSs, that we call "default PSs". This set is manually selected from the PSs used in a strong Reversi program Logistello [11].

Considering the symmetry of the board and initial state of Reversi, we need only 9 sets of PSs, each set is for one cell of {A1, B1, C1, D1, B2, C2, D2, C3, D3}. For example, PSs for A1 can be used also for A8, H1 and H8 by a rotation. It is usual that multiple small PSs are used to evaluate one position instead of a single wide PS. For example, the action evaluation value is decided by using 3 PSs, 7, 8 and 9 shown in Figure 5.2 in the case of default PSs.

For fair comparison, we use the same number of PSs for each position, for example 3 for B1, then totally 25 PSs are optimized. Still, there are some strategies about how many PSs are optimized *at once*.

- 25 pattern shapes are optimized at once, by one optimization procedure. This is a straightforward way, but the variable space is huge.
- one pattern shape is optimized by one procedure, totally 25 optimization procedures are done iteratively. The variable space is smaller, but the dependency of PSs to each other is not considered well.
- the set of PSs for one position such as B1 is optimized at once, totally 9 optimization procedures are done iteratively.

Considering only the expected maximum performance, the first one is the best choice. But usually, cost and difficulty of optimization depends on the size of variable space, and the relationship is not linear. If the number of variables is twice, maybe the cost is over twice, possibly 4-times, 8-times or more. So, we employed the last one in this thesis, considering the balance of the cost and the efficiency.

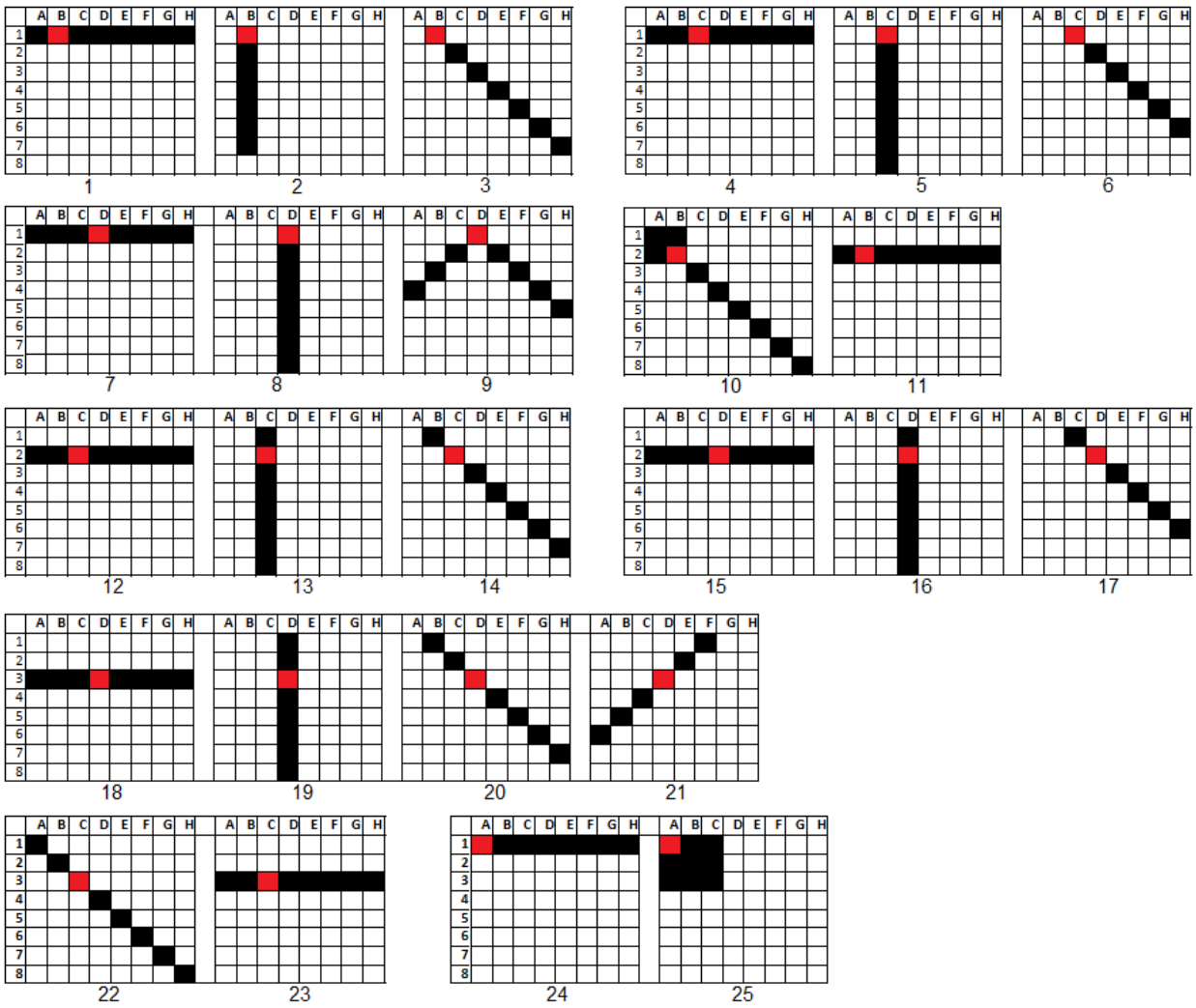


Figure 5.2: The default PSs are selected manually.

5.1.2 Objective function

Also for evaluating the goodness of PSs, there are many possible ways with their advantages and weaknesses.

- Battle-performance against a fixed opponent. As our goal is to make the strong MCTS player, then this evaluation is the most direct and straightforward one. But

this requires BTMM learning before battles, and many games to reduce the cost by random seeds. Then the evaluation cost is huge.

- Heavy-MLE (Mean Log Evidence), the performance of BTMM learning.
- Light-MLE, the performance of BTMM learning, but with smaller training data and shorter MM iterations.
- A direct measurement without any learning [38]. This will be introduced in the next Chapter.

The first one is the straightforward way, but it is almost impossible to employ considering the expected cost. The second one can be used to reduce the cost, because we can assume "learning performance can be used to estimate the strength of MCTS". Further, the third one can be used to reduce the cost more, because we can assume "learning performance with light settings can be used to estimate the true learning performance". Last one is effective if the time is very limited, but the quality is worse than others. So, we employed the third one (Light-MLE by light-BTMM) in this thesis, considering the balance of the cost and the efficiency.

5.1.3 Optimization method

As already discussed in the last Chapter, there are many possible optimization methods.

- Random Search, which use the best solution among some solutions randomly sampled. The cost is the lowest, but the performance will be also the lowest.
- Hill-Climbing or Local Search (LS). The expected quality is better, but the needed cost is bigger.
- Simulated Annealing (SA). The expected quality is better than LS, but the cost is bigger.
- Genetic Algorithm (GA). The expected quality is better than SA, but the cost is bigger.
- Brute Force. The optimal set of PSs can be achieved, but it is impossible to execute by the cost.

Considering the balance of the cost and the efficiency, Random Search and Brute Force cannot be employed. After a series of preliminary experiments, we employed SA as the optimization method in this thesis.

5.2 Problem Description and Methods

This section introduces the problem description including the definition of some symbols and functions, and the whole algorithm of our proposal method.

5.2.1 Problem Description

Let $C_{target} = \{A1, B1, C1, D1, B2, C2, D2, B3, C3\}$ be the target cells of Reversi board, so that several PSs are optimized for each cell.

Let $PS_c = \{p_1, p_2, .., p_m\}$ be a set of pattern shapes for a cell $c \in C_{target}$. Let $D = \{D_{A1}, D_{B1}, .., D_{C3}\}$ be the default pattern shapes shown in Figure 5.2. For example, D_{A1} has two elements, two pattern shapes, {A1, A2, A3, B1, B2, B3, C1, C2, C3} (numbered 1 in Figure 5.2), and {A1, B1, C1, D1, E1, F1, G1, H1} (numbered 2).

The optimization target is a set of PSs for a cell, PS_c, and the evaluation function is the early result of BTMM (MLE of light-BTMM). Let PS'_{A1} be the evolved set of pattern shapes for A1 after optimization. Finally, let $E = \{E_{A1}, E_{B1}, .., E_{C3}\}$ be the whole evolved pattern shapes.

5.2.2 Our Method

As mentioned above, we employ Simulated Annealing as the optimization method, light-BTMM as the evaluation function of the set of PSs, and several pattern shapes for a position is optimized by one SA.

It must be noted that each set of PSs is not optimized independently from other sets. When PS_{A1} is optimized, default PSs $\{D_{B1}, D_{C1}, .., D_{C3}\}$ are used as the features for other positions. After E_{A1} the evolved set of PSs for A1 is obtained, then PS_{B1} is optimized and here $\{\mathbf{E_{A1}}, D_{C1}, .., D_{C3}\}$ are used.

As SA is a stochastic optimization, sometimes the result of SA is worse than the average expected performance. To reduce the risk of such bad result, we can do multiple trials of SA, and select the best one. Figure 5.3 shows the image of whole algorithm. The initial set of PSs is given (1), and 4 SAs are done (2), each SA optimizes several PSs for A1 (3), and the evaluation value is calculated by light-BTMM (4). After 4 SAs are done, 4 sets of PSs for A1 are obtained, then the best one with the highest light-MLE is selected (5). Finally, after 9 iterations, all sets of PSs are optimized (6), here BTMM learning with the regular number of games and loops is done (7), and the result is used in MCTS (8).

Complete Algorithm is as follows:

Complete Algorithm

1. Game records (G) and a set of default pattern shapes D are given.

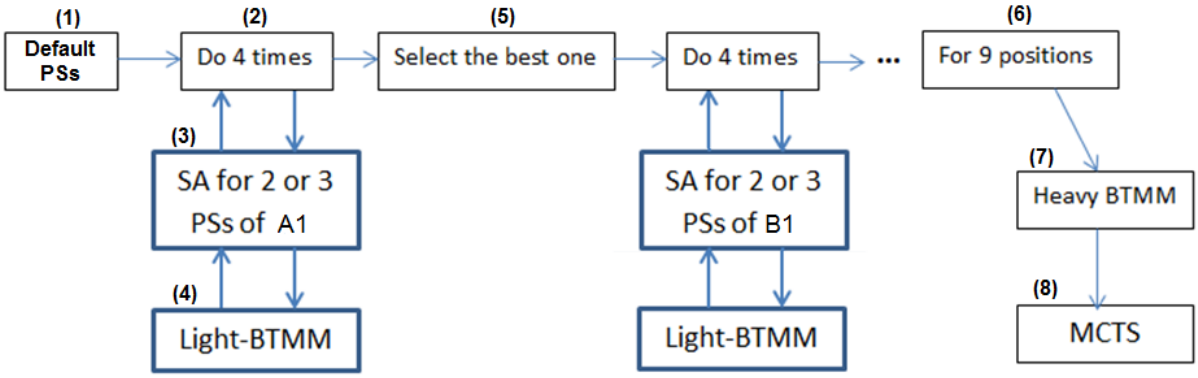


Figure 5.3: The Complete Algorithm for 4 trials of SA.

2. Parameters ($0 < T_{stop} < T_{init}$, $\gamma < 1$, $1 \leq n_{trials}$) are decided.
3. baseSet := D, evolvedSet := $\emptyset$
4. **foreach** cell $c \in C_{target}$
   (a) D_c := the default set of pattern shapes for c
   (b) baseSet := baseSet $\setminus$ D_c
   (c) candidateSet := $\emptyset$
   (d) for i=1 to n_{trials}
      i. $T = T_{init}$
      ii. currentSet := D_c
      iii. currentMLE := lightBTMM(currentSet), MLE value from BTMM by using features baseSet $\cup$ evolvedSet $\cup$ currentSet, and the game records G.
      iv. **while** T > T_{stop}
         A. newSet is generated by one of adding/deleting/changing a cell of a PS of currentSet randomly
         B. newMLE := lightBTMM(newSet)
         C. badness := currentMLE - newMLE
         D. if badness $\leq$ 0 then currentSet := newSet
         E. if badness > 0 then currentSet := newSet with probability $e^{\frac{-badness}{T}}$
         F. if currentSet is replaced, currentMLE := newMLE
         G. $T := T * \gamma$
      v. candidateSet := candidateSet $\cup$ {currentSet}
   (e) E_c is selected from candidateSet, in other words result sets of PSs after n_{trials} SAs.

(f) evolvedSet := evolvedSet $\cup$ E_c

5. evolvedSet is used

There are many loops in the whole algorithm. The top layer is the loop for 9 target positions (Step 4), second layer is the loop for n_{trials} SAs (Step 4(d)), and the last one is the loop of SA iterations (Step 4(d) iv.). In each iteration of SA, a new set of PSs is generated (Step 4(d) iv. A), evaluated by using light-BTMM (B), and accepted at a probability (C-E). Please note that the loop for SA and the loop for the 9 target positions cannot be easily done in parallel, but the loop for n_{trials} SAs can easily be done in parallel using n_{trials} threads.

Figure 5.4 is an example of evolution of PSs using SA. X-axis is the number of iterations, Y-axis is the evaluation value, light-BTMM. There are oscillations of light-BTMM in earlier iterations because the temperature T is higher, then wrong PSs can be easily accepted. It decreases in the middle and converges at the ending iterations. This is normal behavior of SA.

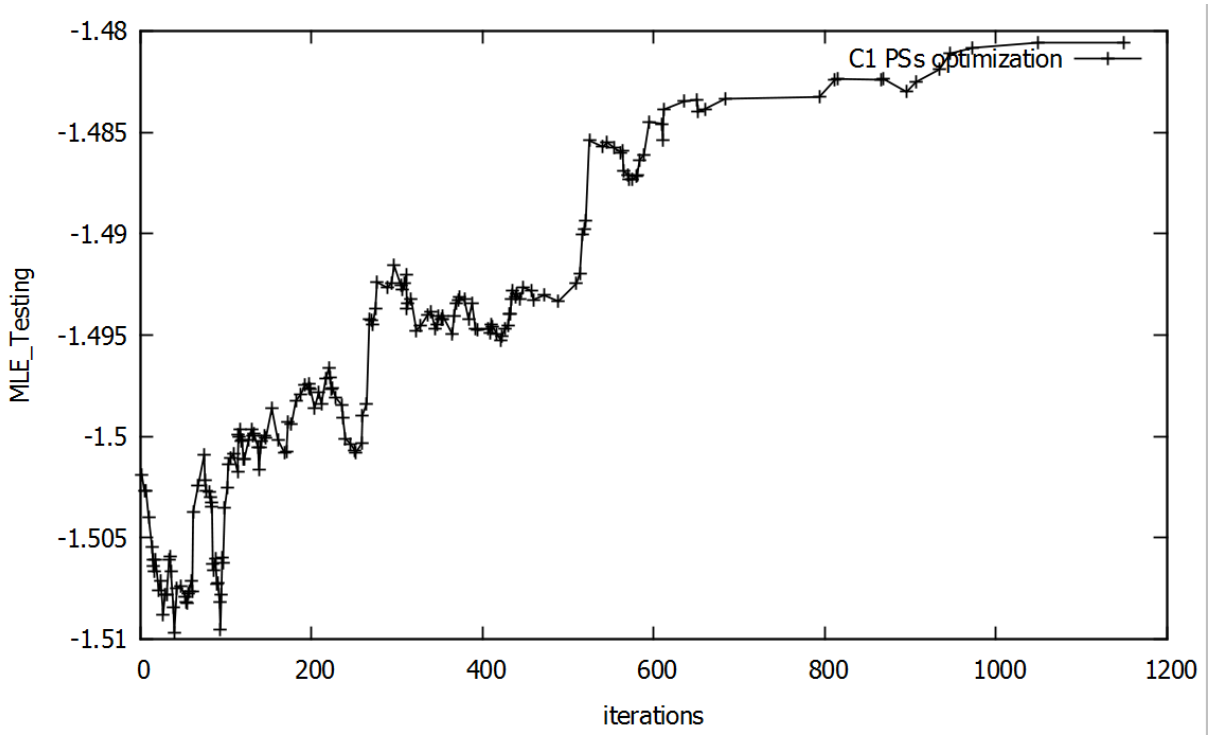


Figure 5.4: An example of SA evolution (The PSs evolution of C1 position).

5.3 Experiment 1

In this section, our algorithm is performed and evaluated. The brief stream of the experiments is as follows:

- Game records used for learning is prepared
- Default pattern shapes D is prepared
- Parameters for optimization are fixed
- Whole algorithm is executed, evolved pattern shapes E is obtained
- MCTS Riversi player with evolved PSs E is made, and compared against that with default PSs D.

5.3.1 Game records

For supervised learning, good examples should be given. In our case, game records played by strong players can be the good examples. 70,000 games are downloaded from the web-site of Michael Buro [11], author of Logistello program, http://skatgame.net/mburo/ggs/game-archive/Othello. These game records have the advantage of being publicly available for free.

For learning and testing of BTMM, *after optimization before MCTS*, 65,000 and 5,000 games are used, respectively. If we can employ more games for learning, the performance of BTMM and then MCTS will be improved, but the cost will be more expensive. The number of loops for BTMM is set to 20, after some preliminary experiments. If we iterate less loops, the risk of underfitting rises, and if we iterate more loops, the risk of overfitting rises and the cost will be more expensive.

5.3.2 Default pattern shapes

Default pattern shapes D must be prepared for three purposes, 1) to use as the initial set of PSs in the optimization, 2) to compare the learning performance between D and evolved PSs, and 3) to compare the MCTS perfomance among them.

25 pattern shapes D have been carefully picked up from the pattern shapes used in Logistello, as shown in Figure 5.2 and Table 5.1. After iterating 20 loops of BTMM using default PSs D and 65,000 games for learning, the performance is measured by using 5,000 games for testing, and the MLE value was -1.47557. If MLE value using a set of PSs is bigger than this value, the set can be considered to be better than D.

5.3.3 Parameters for optimization

In this experiment, rich resources are given to the algorithm and only 1 whole trial was done, to show how good performance can be achieved. Parameters for learning are as follows:

Table 5.1: The default pattern shapes

Number	Position	Pattern shapes
1	A1	A1, A2, A3, B1, B2, B3, C1, C2, C3
2		A1, B1, C1, D1, E1, F1, G1, H1
3	B2	A1, B2, C3, D4, E5, F6, G7, H8, A2, B1
4		A2, B2, C2, D2, E2, F2, G2, H2
5	C3	A3, B3, C3, D3, E3, F3, G3, H3
6		A1, B2, C3, D4, E5, F6, G7, H8
7	B1	A1, B1, C1, D1, E1, F1, G1, H1
8		B1, B2, B3, B4, B5, B6, B7, B8
9		B1, C2, D3, E4, F5, G6, H7
10	C1	A1, B1, C1, D1, E1, F1, G1, H1
11		C1, C2, C3, C4, C5, C6, C7, C8
12		C1, D2, E3, F4, G5, H6
13	D1	A1, B1, C1, D1, E1, F1, G1, H1
14		D1, D2, D3, D4, D5, D6, D7, D8
15		A4, B3, C2, D1, E2, F3, G4, H5
16	C2	A2, B2, C2, D2, E2, F2, G2, H2
17		C1, C2, C3, C4, C5, C6, C7, C8
18		B1, C2, D3, E4, F5, G6, H7
19	D2	A2, B2, C2, D2, E2, F2, G2, H2
20		D1, D2, D3, D4, D5, D6, D7, D8
21		C1, D2, E3, F4, G5, H6
22	D3	A3, B3, C3, D3, E3, F3, G3, H3
23		D1, D2, D3, D4, D5, D6, D7, D8
24		B1, C2, D3, E4, F5, G6, H7
25		A6, B5, C4, D3, E2, F1

- The numbers of games used for learning and test of light-BTMM are 14,000 and 1,000, respectively.
- The number of loops of light-BTMM is 4.
- The number of trials n_{trials} of SAs is 4.
- The initial temperature of SA T_{init} is 0.001
- The final temperature of SA T_{stop} is 0.0001
- The decay parameter of temperature γ is 0.999

By the three parameter T_{init}, T_{stop} and γ, the number of iterations for a SA is about 2300. We need $9 \times 4 \times 4 \times 2300 = 331200$ MM loops for a whole optimization, and for each loop light-BTMM with $14000 + 1000$ games is done. By our implementation in C# and a computer, it took about 2 weeks.

Please note that though light-BTMM (14,000 games for learning, 1,000 games for testing, 4 loops) is used in optimization, heavy-BTMM (65,000 games for learning, 5,000 games for testing, 20 loops) is used to compare the performance of the sets of PSs.

5.3.4 Evolution

This section has 9 subsections, each subsection shows the result of 4 SAs for a position (Algorithm step 4(a) to 4(e)).

Evolution 1, for B1

At first, the set of PSs for B1 is targeted. The baseSet (initialized at Algorithm step 3) is $\{D_{A1}, D_{B1}, D_{C1}, D_{D1}, D_{B2}, D_{C2}, D_{D2}, D_{C3}, D_{D3}\}$, and its MLE value (after heavy-BTMM) is -1.47557.

After evolution 1, we have improved pattern shapes E_{B1} as in table 5.2, and its MLE value is -1.45754. Then, the gain is (-1.45754) - (-1.47557) = 0.01803.

Table 5.2: Pattern shapes of evolution 1, B1

Number	D_{B1}	E_{B1} (the improved pattern shapes)
1	A1, B1, C1, D1, E1, F1, G1, H1	A1, B1, C1, D1, E1, F1, G1, H1, B2
2	B1, B2, B3, B4, B5, B6, B7, B8	A1, B1, C1, D1, A2, B2, C2, A3
3	B1, C2, D3, E4, F5, G6, H7	A1, B1, F1, C2, G2, B3, G7

Evolution 2, for C1

Next, the set of PSs for C1 is targeted. The baseSet is changed to $\{D_{A1}, \mathbf{E_{B1}}, D_{C1}, D_{D1}, D_{B2}, D_{C2}, D_{D2}, D_{C3}, D_{D3}\}$, and its MLE value is -1.45754.

After evolution 2, we have improved pattern shapes E_{C1} as in table 5.3, and its MLE value is -1.44368. Then, the gain is 0.01386.

Table 5.3: Pattern shapes of evolution 2, C1

Number	D_{C1}	E_{C1} (the improved pattern shapes)
1	A1, B1, C1, D1, E1, F1, G1, H1	A1, B1, C1, D1, E1, F1, H1, B2, C2, E2, E3
2	C1, C2, C3, C4, C5, C6, C7, C8	C1, C2, D1, D2, F2, E3, F4, B3
3	C1, D2, E3, F4, G5, H6	C1, A1, B2, C2, D2, A3, B1, C3, C6

Evolution 3, for D1

Next, the set of PSs for C1 is targeted. The baseSet is changed to $\{D_{A1}, \mathbf{E_{B1}}, \mathbf{E_{C1}}, D_{D1}, D_{B2}, D_{C2}, D_{D2}, D_{C3}, D_{D3}\}$, and its MLE value is -1.44368.

After evolution 3, we have improved pattern shapes E_{D1} as in table 5.4, and its MLE value is -1.43955. Then, the gain is 0.00413.

Table 5.4: Pattern shapes of evolution 3, D1

Number	D_{D1}	E_{D1} (the improved pattern shapes)
1	A1, B1, C1, D1, E1, F1, G1, H1	D1, C1, E1, F1, H1, D5, F2
2	D1, D2, D3, D4, D5, D6, D7, D8	D1, A1, F1, G1, H1, D2, C2, E2, F2, F3
3	A4, B3, C2, D1, E2, F3, G4, H5	D1, C1, E1, C2, D2, E2, C3, D3, E4

Evolution 4, for B2

Next, the set of PSs for B2 is targeted. The baseSet is changed to $\{D_{A1}$, $\mathbf{E_{B1}}$, $\mathbf{E_{C1}}$, $\mathbf{E_{D1}}$, D_{B2}, D_{C2}, D_{D2}, D_{C3}, $D_{D3}\}$, and its MLE value is -1.43955.

After evolution 4, we have improved pattern shapes E_{B2} as in table 5.5, and its MLE value is -1.43288. Then, the gain is 0.0066.

Table 5.5: Pattern shapes of evolution 4, B2

Number	D_{B2}	E_{B2} (the improved pattern shapes)
1	A1, B2, C3, D4, E5, F6, G7, H8, A2, B1	B2, A1, C1, A3, B1, A2, F6
2	A2, B2, C2, D2, E2, F2, G2, H2	B2, A7, B7, A1, B6, B3, B5, H1

Evolution 5, for C2

Next, the set of PSs for C2 is targeted. The baseSet is changed to $\{D_{A1}$, $\mathbf{E_{B1}}$, $\mathbf{E_{C1}}$, $\mathbf{E_{D1}}$, $\mathbf{E_{B2}}$, D_{C2}, D_{D2}, D_{C3}, $D_{D3}\}$, and its MLE value is -1.43288.

After evolution 5, we have improved pattern shapes E_{C2} as in table 5.6, and its MLE value is -1.42504. Then, the gain is 0.00784.

Table 5.6: Pattern shapes of evolution 5, C2

Number	D_{C2}	E_{C2} (the improved pattern shapes)
1	A2, B2, C2, D2, E2, F2, G2, H2	C2, A4, B3, C3, C4, B4, F4, D3
2	C1, C2, C3, C4, C5, C6, C7, C8	C2, B4, C3, D2, D4, C6, D3, A2
3	B1, C2, D3, E4, F5, G6, H7	C2, E3, C4, F6, D2, E2, D6, C1

Evolution 6, for D2

Next, the set of PSs for D2 is targeted. The baseSet is changed to $\{D_{A1}$, $\mathbf{E_{B1}}$, $\mathbf{E_{C1}}$, $\mathbf{E_{D1}}$, $\mathbf{E_{B2}}$, $\mathbf{E_{C2}}$, D_{D2}, D_{C3}, $D_{D3}\}$, and its MLE value is -1.42504.

After evolution 6, we have improved pattern shapes E_{D2} as in table 5.11, and its MLE value is -1.40159. Then, the gain is 0.02345.

Table 5.7: Pattern shapes of evolution 6, D2

Number	D_{D2}	E_{D2} (the improved pattern shapes)
1	A2, B2, C2, D2, E2, F2, G2, H2	D2, C2, E2, C4, E3, C3, D3, F4, A5, B2, A1
2	D1, D2, D3, D4, D5, D6, D7, D8	D2, E2, C2, E3, C6, F2, D3, A2, H1
3	C1, D2, E3, F4, G5, H6	D2, E3, C3, C5, F6, E5, F2

Evolution 7, for D3

Next, the set of PSs for D3 is targeted. The baseSet is changed to $\{D_{A1}$, $\mathbf{E_{B1}}$, $\mathbf{E_{C1}}$, $\mathbf{E_{D1}}$, $\mathbf{E_{B2}}$, $\mathbf{E_{C2}}$, $\mathbf{E_{D2}}$, D_{C3}, $D_{D3}\}$, and its MLE value is -1.40159.

After evolution 7, we have improved pattern shapes E_{D3} as in table 5.8, and its MLE value is -1.39340. Then, the gain is 0.0082.

Table 5.8: Pattern shapes of evolution 7, D3

Number	D_{D3}	E_{D3} (the improved pattern shapes)
1	A3, B3, C3, D3, E3, F3, G3, H3	D3, C3, F4, C5, C4, F2, E4, F3, C1
2	D1, D2, D3, D4, D5, D6, D7, D8	D3, D7, E3, F4, A5, E6, C3, D6, B8
3	B1, C2, D3, E4, F5, G6, H7	D3, F4, F5, C6, D6, G6, C3, E6, E2, D4
4	A6, B5, C4, D3, E2, F1	D3, E3, D4, C3, B5

Evolution 8, for C3

Next, the set of PSs for C3 is targeted. The baseSet is changed to $\{D_{A1}$, $\mathbf{E_{B1}}$, $\mathbf{E_{C1}}$, $\mathbf{E_{D1}}$, $\mathbf{E_{B2}}$, $\mathbf{E_{C2}}$, $\mathbf{E_{D2}}$, D_{C3}, $\mathbf{E_{D3}}\}$, and its MLE value is -1.39340.

After evolution 8, we have improved pattern shapes E_{C3} as in table 5.9, and its MLE value is -1.38901. Then, the gain is 0.00439.

Table 5.9: Pattern shapes of evolution 8, C3

Number	D_{C3}	E_{C3} (the improved pattern shapes)
1	A1, B2, C3, D4, E5, F6, G7, H8	C3, F6, F4, D6, C5, E3, H3, C8, H1, A8
2	A3, B3, C3, D3, E3, F3, G3, H3	C3, C4, C5, C6, C7, E5, D4, B7, D3, A1

Evolution 9, for A1

Finally, the set of PSs for A1 is targeted. The baseSet is changed to $\{D_{A1}$, $\mathbf{E_{B1}}$, $\mathbf{E_{C1}}$, $\mathbf{E_{D1}}$, $\mathbf{E_{B2}}$, $\mathbf{E_{C2}}$, $\mathbf{E_{D2}}$, $\mathbf{E_{C3}}$, $\mathbf{E_{D3}}\}$, and its MLE value is -1.38901.

After evolution 8, we have improved pattern shapes E_{C3} as in table 5.10, and its MLE value is -1.38994. Then, the gain is -0.00093. The reason why no improvement was achieved is that A1 is always an attractive position, almost independent from the patterns.

Table 5.10: Pattern shapes of evolution 9, A1

Number	D_{A1}	E_{A1} (the improved pattern shapes)
1	A1, B1, C1, D1, E1, F1, G1, H1	A1, B1, C1, D1, G1, H1
2	A1, A2, A3, B1, B2, B3, C1, C2, C3	A1, A2, A3, B1, B2, C1, D1, A4

5.3.5 MCTS performance

Our goal is to make a strong MCTS Reversi player, instead of good MLE. We assume that an action evaluation function with good MLE also makes the MCTS player stronger, but to confirm that, we conducted a series of games.

As mentioned in Chapter 3, an action evaluation function can be used both for search part and simulation part of MCTS. In the selection phase of MCTS, our program used the following formula of UCB (equation 5.1) [42].

$$UCB_{Bias} = \frac{w_j}{n_j} + C \times \sqrt{\frac{\ln n}{n_j}} + \alpha \times \sqrt{\frac{K}{n+K}} \times P(m_j) \tag{5.1}$$

First term and second term of equation 5.1 are usual UCB, and third term is the bias, to enhance the search of moves with good evaluation value $P(m_j)$. α and K are the parameters, 0.5 and 5000 are used in our experiments. These parameters are tuned through preliminary experiments using default pattern shapes D instead of evolved PSs, to keep the fairness of comparison.

Against an alpha-beta program

At first, we employed a program as the opponent, Riversi in C#, open source Reversi program with alpha-beta tree search. This program is much weaker than Logistello, but its expert level is stronger than almost all human players. It can be download from the link http://www.codeproject.com/Articles/4672/Reversi-in-C.

For each move, 3 seconds are given to search program. Over 2000 games were played between MCTS with default pattern shapes D and Riversi in C#, the winning ratio of MCTS was 54.7%, in other words these two programs have almost the same strength.

After the 7^{th} evolution, we had a set of PSs $\{D_{A1}$, $\mathbf{E_{B1}}$, E_{C1}, E_{D1}, D_{B2}, D_{C2}, D_{D2}, D_{C3}, $D_{D3}\}$, and its MLE value is -1.39340, 0.08217 better than the default PSs. Over 3000 games were played between MCTS with this set of PSs and Riversi in C#, the winning ratio of MCTS was 94.8%. It is clear that the MCTS with evolved PSs is stronger not only than Riversi in C# but also than MCTS with default PSs.

Self play experiments

Table 5.11 shows the progress of MLE and winning ratio against the MCTS with the default PSs, over evolutions. MLE increased gradually step by step from -1.47557 to -1.38994, and also the winning ratio increased gradually from 50% to 83.45%.

Table 5.11: Progress of MLE and strength with 95% confidence interval, over evolutions

Number	Evolution	MLE	MCTS
1	Default pattern shapes	-1.47557	50%
2	evolution 1 vs Default	-1.45754	52.72% [48.06% - 57.38%]
3	evolution 2 vs Default	-1.44368	55.84% [53.84% - 58.64%]
4	evolution 3 vs Default	-1.43955	58.85% [56.07% - 61.63%]
5	evolution 4 vs Default	-1.43288	62.12% [59.47% - 64.77%]
6	evolution 5 vs Default	-1.42504	67.71% [65.51% - 69.91%]
7	evolution 6 vs Default	-1.40159	74.48% [72.41% - 76.55%]
8	evolution 7 vs Default	-1.39340	79.56% [77.09% - 82.03%]
9	evolution 8 vs Default	-1.38901	81.76% [79.58% - 83.94%]
10	evolution 9 vs Default	-1.38994	83.45% [81.19% - 85.71%]

Figure 5.5 shows the relationship between MLE (X-axis) and winning ratio (Y-axis). It is clear that the two performances have a positive correlation. This result supports our assumption that "learning performance can be used to estimate the strength of MCTS", and then supports our decision to use learning performance to evaluate a set of PSs.

5.3.6 Obtained pattern shapes

Figure 5.6 shows the set of pattern shapes obtained by the whole process of proposed optimization. We can see that they are very different from the default pattern shapes (Figure 5.2). Further, they seem to be very *strange*. It is almost impossible to believe that these PSs are better than the default ones. But as shown above, the performance of these PSs are surely and significantly better than the default ones, not only about BTMM learning but also about MCTS playing. This fact supports that automatic pattern shape optimization is promising, because it will be very difficult to design such strange shapes by hand.

5.4 Experiment 2

In the last experiment, only 1 trial was done by using rich resources, to show that very good performance *can* be achieved. However, it is also valuable to show the behaviors of many trials by using poorer resources. Then, here we conducted another series of experiments using the following parameters.

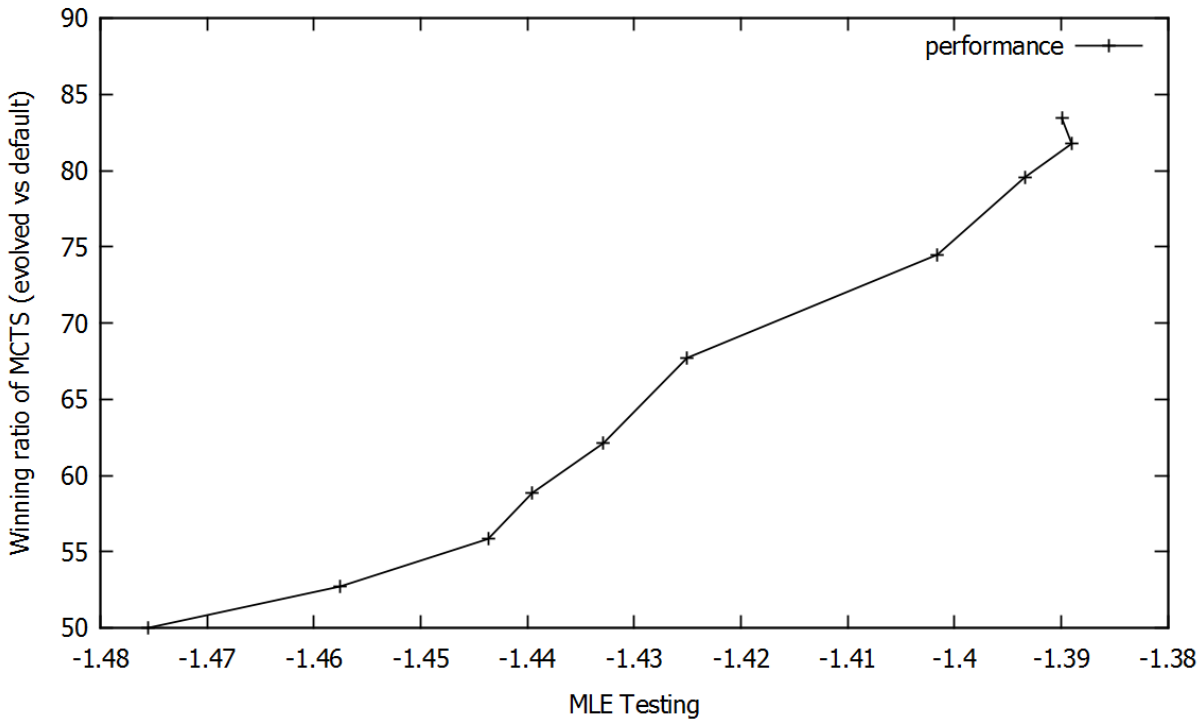


Figure 5.5: Performance of MCTS program after each evolution.

- The numbers of games used for learning and test of light-BTMM are 10,000 and 1,000, respectively.
- The number of loops of light-BTMM is 3.
- The number of trials n_trials of SAs is 1.
- The initial temperature of SA T_{init} is 0.001
- The final temperature of SA T_{stop} is 0.0001
- The decay parameter of temperature γ is 0.995

Here the number of iterations for one SA is about 460 instead of 2300. The total number of iterations is $9 \times 3 \times 1 \times 460 = 12420$ instead of 331200, then the cost for 1 trial is about 3% of the last experiment, considering the number of games.

Figure 5.7 shows the result, where X-axis is the number of optimized positions, and Y-axis is the MLE value. Compared to the progress of MLE values in the case of Table 5.11, it is clear that the evolutions under the poorer condition are varied and not robust. Sometimes MLE value deteriorates after the evolution, and finally the gap between the best case (-1.408) and the worst case (-1.442) is very big. It is usual that stochastic optimization such as SA can produce varied performance, especially under poorer resource

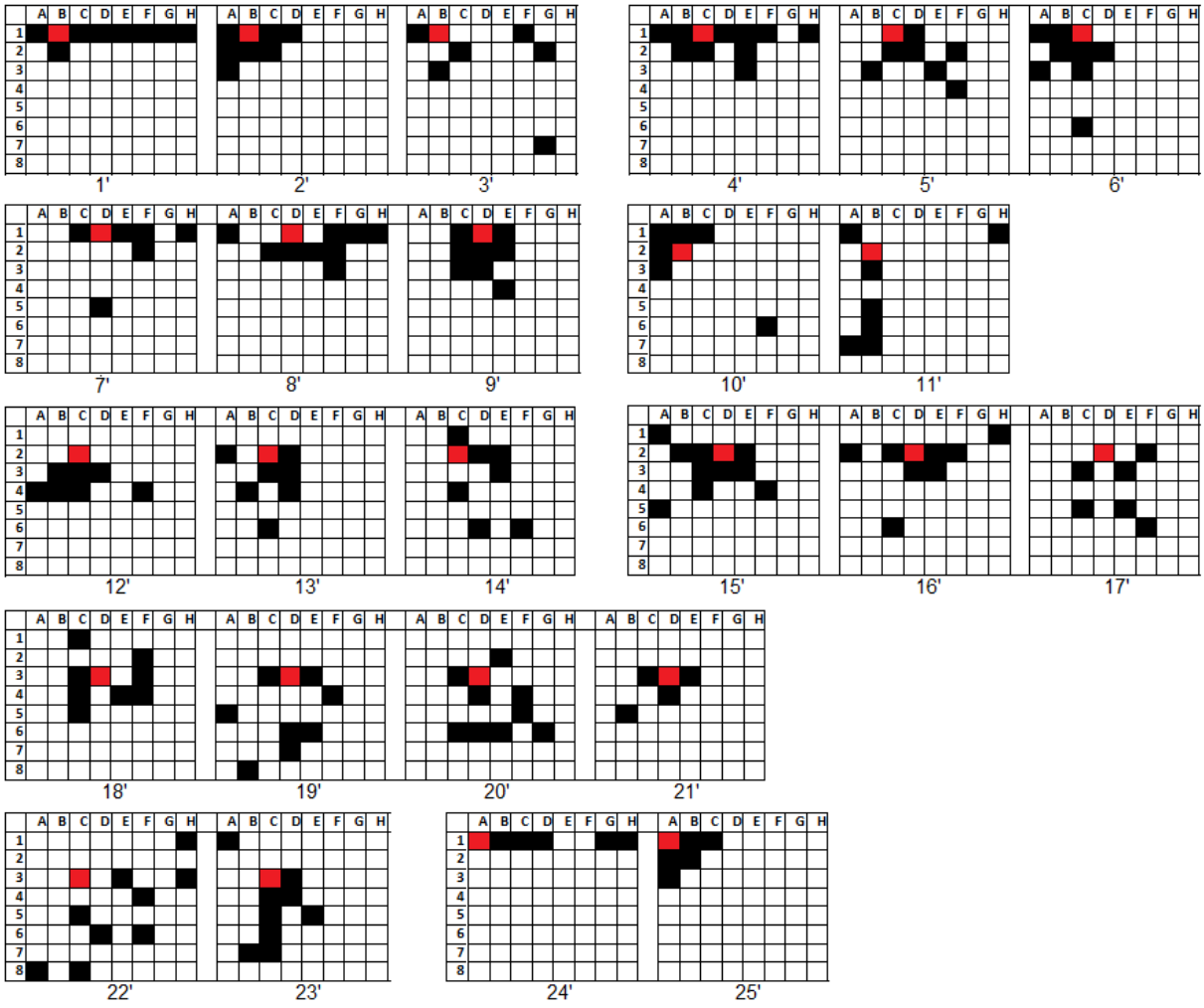


Figure 5.6: 25 pattern shapes after the evolution

conditions. So, we believe that our approach "to select the best result among the multiple (n_{SA}) trials after each step" is a good way to obtain a robust result, without increasing the optimization time if they are done in parallel.

5.5 Conclusion

For optimizing the set of pattern shapes, three components should be carefully selected. In our case, simulated annealing was selected as the optimization method, light-MLE, the early result of BTMM, was selected as the evaluation function, and several PSs for a target cell were selected as the variables to be optimized at once.

Through some experiments, we showed the selections were reasonable, and evolution of pattern shapes is really effective not only to improve the learning performance, but also to improve the strength of MCTS.

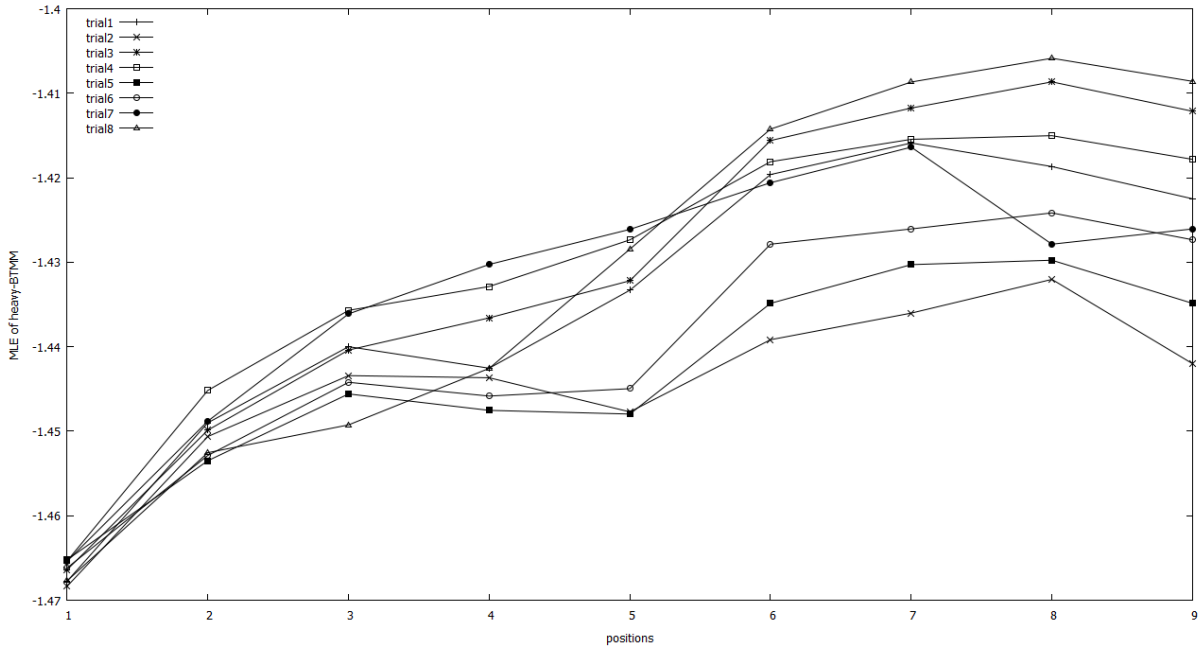


Figure 5.7: Evolutions of pattern shapes under poorer condition, 8 trials

Chapter 6

EVALUATION OF PATTERN SHAPES BEFORE MACHINE LEARNING

In the optimization shown in Chapter 5, *a set of* pattern shapes is evaluated, *using* learning. In this chapter, we introduce a measurement to evaluate a **single** pattern shape, **without** learning. The goodness of a pattern shape is estimated using a kind of statistics over game records. This is one of our main contributions, one conference paper and one journal paper titled "Extracting Important Patterns for Building State-Action Evaluation Function in Riversi" [39] and "Evaluation of pattern shapes in board games before machine learning" [38] were accepted to TAAI2012 and IJEE respectively.

6.1 Introduction

It has been shown that we can estimate the goodness of pattern shapes (PSs) applied to MCTS, by the performance of BTMM. Light-BTMM was used for evaluating a set of PSs, but still this is expensive. In the phase of simulated annealing, only one PS is changed by adding/removing/changing a cell, so a more rapid method to evaluate one PS is desired, to discard unpromising candidates.

In this section, some statistic values related to a PS and given game records are introduced, and measurement to estimate the goodness of the PS is proposed.

6.1.1 Selection Ratio

Let P be a pattern shape. Let X_i be a possible pattern, $1 \leq i \leq n$, n is the number of possible patterns of P.

Let G be a set of game records. Each game has only the sequence of played moves, but the set of other legal moves can be easily extracted from it. $G = (T, m, L)$ where,

- T is the number of moves played in G,
- $m = (m_1^G, .., m_T^G)$ is the selected moves in G.
- $L = (L_0^G, .., L_{T-1}^G)$ is the sets of legal moves. L_{t-1}^G is the set of legal moves just before the move m_t^G is played, and so $m_t^G \in L_{t-1}^G$.

If a pattern appears in the selected moves frequently through games, the pattern will be attractive to play. On the other hand, if a pattern hardly appears in the selected moves despite frequently appearing in the legal moves, the pattern is not attractive to play. So, the "selection ratio" defined in the following equation is the important information.

$$sel\_rat(X_i, G) = \frac{\text{The number of appearances of pattern } X_i \text{ in the set of selected move m}}{\text{The number of appearances of pattern } X_i \text{ in the set of legal moves L}}$$

For example, two patterns A and B using a pattern shape $\{A1, B1, ..., H1\}$ are shown in Figure 6.1. In some game records, $sel\_rat(A, G) = \frac{120}{130}$, in other words B1 is very attractive for White player at pattern A. On the other hand, $sel\_rat(B, G) = \frac{10}{150}$, in other words B1 is not attractive for White player at pattern B. Selection ratios using this PS varies from low value to high value, it means that this PS is valuable to refer.

On the other hand, we can see two patterns C and D which use another pattern shape $\{B1, B2, D5, E6, F6, E7, F7\}$ shown in Figure 6.2. In the same game records, $sel\_rat(C, G) = \frac{30}{100}$, and $sel\_rat(D, G) = \frac{60}{200}$. Selection ratios using this PS is almost the same, independent from patterns, this means that this PS is not valuable to refer.

Considering these examples, we can assume that if sel_rat values among patterns varied from low to high, the PS is valuable.

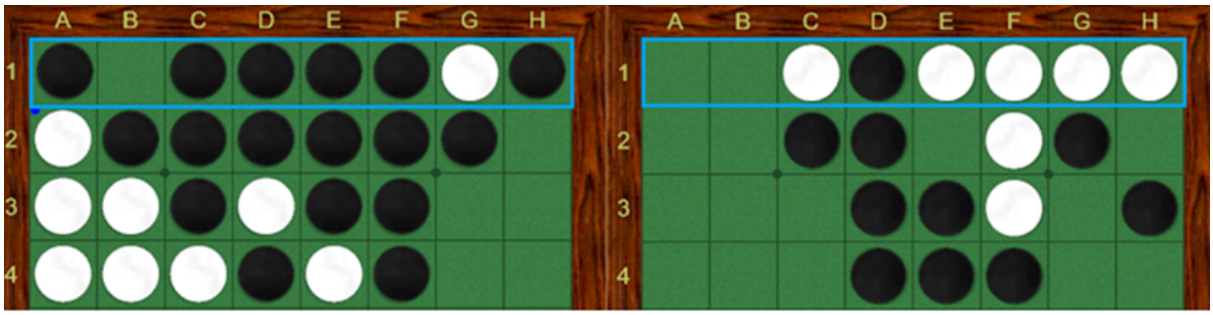


Figure 6.1: Two example patterns for B1, referring a good PS, the next player is White.

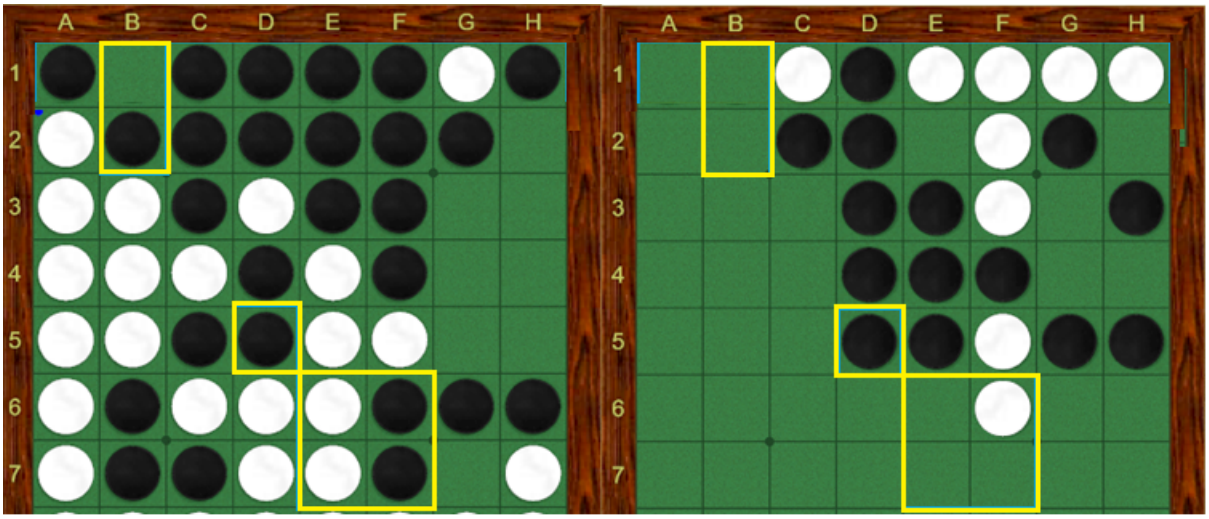


Figure 6.2: Two example patterns for B1, referring a bad PS, the next player is White.

6.1.2 measurement 1: standard deviation of selection ratio distribution

Assume we want to evaluate the importance of a pattern shape, and let $P_{\tau_{app}}$ be the set of patterns which appear in the game records G at least τ_{app} times. Patterns which appear only few times are not considered, because their statistics are not reliable.

Let m be the size of $P_{\tau_{app}}$. Here, we can calculate the average selection ratio ω by equation 6.1, and the standard deviation $std\_dev(P, G)$ by equation 6.2. As mentioned above, we believe that the pattern shape is important if $std\_dev(P, G)$ is high.

$$\omega = \frac{\sum_{i=1}^{m} sel\_rat(X_i)}{m} \tag{6.1}$$

$$std\_dev(P, G) = \sqrt{\frac{1}{m}(\omega - \sum_{i=1}^{m}(sel\_rat(X_i))^2} \tag{6.2}$$

division of patterns

Here we divide the set of patterns $P_{\tau_{app}}$ into (k+2) groups, where $k = \lceil \log_2 m \rceil$, according to their selection ratios.

- $P_{\tau_{app}} = \{P_{[0]}, P_{[0,1/k]}, P_{[1/k,2/k]}, .., P_{[(k-1)/k,1]}, P_{[1]}\}$
- $P_{[0]}$ is the subset of $P_{\tau_{app}}$, and $sel\_rat(X_i) = 0$ for all X_i
- $P_{[0,1/k]}$ is the subset of $P_{\tau_{app}}$, and $0 < sel\_rat(X_i) \leq 1/k$ is satisfied for all X_i

By this definition, distributions of selection ratios can be drawn as a histogram.

example

Figure 6.3 shows the typical examples of such a histogram of selection ratio. For simplicity, here we consider 10 patterns.

- In figure 6.3(A), there are 5 patterns for which the selection ratios are 0, and the selection ratios of the others are 1. In other words, $P_{[0]} = \{X_1, .., X_5\}$, $P_{[1]} = \{X_6, .., X_{10}\}$. In this case, the pattern shape is absolutely important because we can almost perfectly predict whether a move is selected or not if this pattern shape appears.
- In figure 6.3(B), each pattern i has a selection ratio in the interval of $[i/10, (i+1)/10]$, i is from 0 to 9.
- In figure 6.3(C), there are 5 patterns for which the selection ratios are in $[0.4, 0.5]$, and the selection ratios of the others are in $[0.5, 0.6]$. In this case, the pattern shape will be not important.

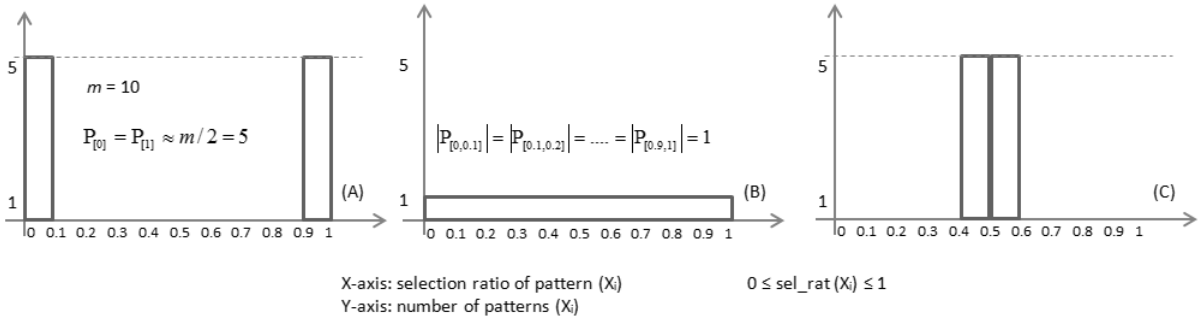


Figure 6.3: Three typical distributions of selection ratios, (A) (B) are desirable cases and (C) is undesirable case.

We use equation 6.1 to evaluate the standard deviation of the patterns in the pattern shape of figure 6.3(A) = 0.5, figure 6.3(B) = 0.316, and figure 6.3(C) = 0.057. It shows that a pattern shape is more important when the standard deviation of the pattern shape is bigger.

6.1.3 measurement 2: scattering factor of the histogram

Except the extreme cases such as figure 6.3(A), it will be better that the distribution of selection ratios is more flat and continuous such as figure 6.3(B), rather than the

case of spiny or tall mountains such as figure 6.3(C). Then, we introduce an additional measurement of pattern shape, "scattering" by equation 6.3. It means the rate of "non-empty" subsets of $P_{\tau_{app}} = \{P_{[0]}, P_{[0,1/k]}, P_{[1/k,2/k]}, .., P_{[(k-1)/k,1]}, P_{[1]}\}$, and it is expected that "a pattern shape is better when there is more scattering".

$$scattering(P, G) = \frac{\sum_{j=0}^{k+1} count\{t_j \mid t_j > 0\}}{k+2}, \tag{6.3}$$

where $t_j = |P_{[\frac{j}{k}, \frac{j+1}{k}]}|$

6.1.4 measurement 3: appearance ratio of patterns

The $app\_ratio(P, G)$ of a pattern shape P in G is calculated by $\frac{|P_{\tau_{app}}|}{|P|}$. In other words, how many patterns in P appear at least τ_{app} times in G.

The appearance ratio is useful to avoid the risk of over-fitting. Considering two pattern shapes shown in Figure 6.4, PS1 (8 cells) has maximum 2×3^7 patterns, PS2 (24 cells) has maximum 2×3^{23} patterns. It is clear that PS2 is better than PS1 about "learning performance", but bad about "generalized performance" because of the over-fitting. In this case, $app\_ratio(PS2, G)$ will be very small if the number of games in G is not so huge, then such PS2 can be rejected, by preferring pattern shapes with high app_ratio values.

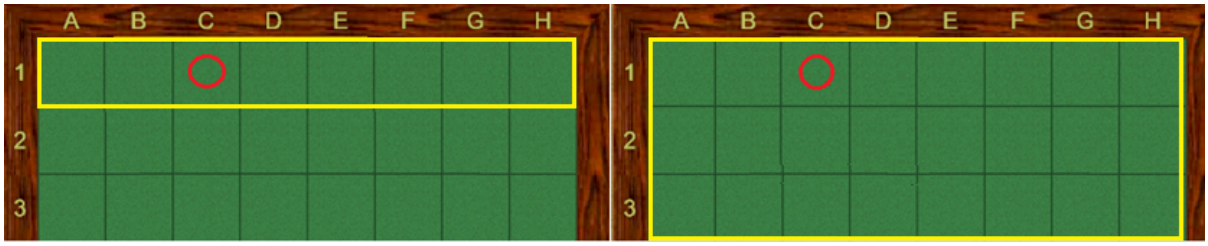


Figure 6.4: Pattern shape with reasonable size of cells (left) and with too many cells (right)

6.1.5 weighted measurements: importance

By using the three measurements described above, we propose the "importance" estimator of a pattern shape as follows:

$$importance(P, G) = std\_dev(P, G) + w_{scat} \times scattering(P, G) + w_{appr} \times app\_ratio(P, G) \tag{6.4}$$

Where w_{scat} is the weight of scattering, w_{appr} is the weight of appearance ratio. The weight of standard deviation w_{std} is 1.0 as the default value, because this property is surely useful.

6.1.6 similarity of two pattern shapes

All measurements and importance above are calculated for a single pattern shape. But it is usual to use several PSs for a position, and in this case "similar" pattern shapes are frequently useless. For example in the case of Riversi, it is clear that "one vertical PS and one horizontal PS" is better than "two vertical PSs". So, here we introduce an extra measurement "similarity" between two pattern shapes as follows:

$$sim(P_1, P_2) = \frac{(\text{the number of common cells of } P_1 \text{ and } P_2)}{min(P_1.length, P_2.length) - 1} \tag{6.5}$$

For example, P1 has {A1, A2, A3, A4} and P2 has {A3, A4, A5}, $sim(P_1, P_2)$ = (2-1) / (3-1) = 0.5.

6.2 Experiments and Evaluation

In this section, we confirm through experiments that pattern shapes with higher importance are really better than that with lower importance. The brief stream of the experiments is as follows:

1. The game records are prepared, and the parameters are fixed.
2. The candidate pattern shapes are prepared.
3. They are separated into 3 (better/middle/worse) groups, by the importance.
4. BTMM learning is executed for each group, and the results are compared
5. The performance of MCTS Riversi players are compared

6.2.1 Step 1. Game records and parameters

300,000 games records were prepared as experiment 1 in Chapter 5. By preliminary experiments, parameters are set as $w_{scat} = 0.2$, $w_{appr} = 0.2$, and $\tau_{sim} = 0.45$. τ_{sim} is used for preventing similar PSs. The minimum length of PS is set to 4, and the maximum length is to 10, respectively.

6.2.2 Step 2. Candidate pattern shapes

The purpose of this experiment is to know the ability of "importance" measure for evaluating the goodness of PSs, then some good PSs and bad PSs should be prepared. Based on the PSs used in Logistello, we prepared 60 PSs in total, by adding some modifications (adding/changing cells) to the original PSs. 9 PSs are prepared for each of $B2$ and $C2$, 6 PSs are prepared for each of the other 7 positions.

6.2.3 Step 3. Separation

60 candidate PSs are separated to 3 groups, better group, middle group and worse group, according to their importances and similarity. Each group has 3 PSs for each of $B2$ and $C2$, 2 PSs for each of the other 7 positions, then 20 PSs totally.

At first, the better group (listed in Table 6.1) is generated as follows:

- There are 6 candidate PSs for $A1$, then the best one in importance is selected. In this case $\{A1, B1, C1, A2, B2, C2, A3, B3, C3\}$ with importance 0.463 is selected.
- The second best one is selected, if the similarity to the best one is under the threshold τ_{sim}. In this case $\{A1, A2, A3, A4, A5, A6, A7, A8, A1, A2\}$ with importance 0.453 is selected, because similarity is 0.37, under 0.45.
- In the same way, 2 PSs for $B1$ are selected, and so on.

Next, the middle group (listed in Table 6.2) is generated from the rest 40 PSs in the same manner. Finally, the worse group (listed in Table 6.3) is the group of the rest 20 PSs. The value of "Sim" column in these tables shows the maximum similarity of each PS among the selected PSs for the position. In the worse group, some of them exceed the threshold $\tau_{sim} = 0.45$ because they are the rest PSs.

6.2.4 Step 4. BTMM learning

Three groups of PSs are prepared, according to the proposed measurements. Here BTMM learnings using these groups are employed, and the results (MLEs) are compared. The number of game records for learning is 97,000, that for testing is 3,000, and the number of loops is 20.

Usually only the testing result is important, but to see the degree of over-fitting, two curves, learing-curve and testing-curve are shown in the graphs. Figure 6.5 is the result, X-axis shows the number of loops, Y-axis shows the MLE value. It is clear that BTMM using better group of PSs is the best, and BTMM using worse group of PSs is the worst. In addition, the gap between learning-curve and testing-curve is the biggest in the case of worse group, this implies that the effect of over-fitting is also the biggest.

Table 6.1: Pattern shapes of better group.

No	Pos	Better pattern shapes	Length	Std Dev	Scat	App ratio	Imp	Sim
1	A1	{A1, B1, C1, A2, B2, C2, A3, B3, C3}	9	0.235	1	0.139	0.463	0.37
2		{A1, A2, A3, A4, A5, A6, A7, A8, B1, B2}	10	0.233	1	0.098	0.453	0.37
3	B1	{A1, B1, C1, A2, B2, C2, A3, B3, C3}	9	0.229	1	0.167	0.463	0.37
4		{A1, B1, C1, D1, E1, F1, G1, H1, A2, B2}	10	0.238	1	0.175	0.473	0.37
5	C1	{A1, B1, C1, D1, A2, B2, C2, D2, E2, F2}	10	0.233	1	0.102	0.453	0.44
6		{A1, B1, C1, D1, E1, F1, G1, H1, B2, G2}	10	0.252	1	0.118	0.476	0.44
7	D1	{A1, B1, C1, D1, E1, F1, G1, H1, B2, G2}	10	0.248	1	0.108	0.470	0.33
8		{A1, B1, C1, D1, A2, B2, C2, D2, E2, F2}	10	0.217	1	0.099	0.437	0.33
9	B2	{A1, B1, C1, D1, E1, F1, A2, B2}	8	0.199	1	0.442	0.488	0.37
10		{A1, B1, A2, B2, C2, D2, B3, C3, B4}	9	0.226	1	0.165	0.459	0.37
11		{A1, B2, C3, D4, E5, F6, G7, H8, A2, B1}	10	0.246	1	0.121	0.470	0.37
12	C2	{A2, B2, C2, D2, E2, F2, G2, H2, B3, C3}	10	0.197	1	0.109	0.419	0.44
13		{B1, C2, D3, E4, F5, G6, H7, C1, B2}	9	0.201	1	0.125	0.426	0.37
14		{A1, B1, C1, D1, E1, A2, B2, C2, D2, E2}	10	0.216	1	0.049	0.426	0.44
15	D2	{A1, B1, C1, D1, E1, A2, B2, C2, D2, E2}	10	0.219	1	0.033	0.426	0.44
16		{A2, B2, C2, D2, E2, F2, G2, H2, C3, D3}	10	0.207	1	0.070	0.421	0.44
17	C3	{C3, D3, E3, F3, G3, H3, C4, C5, C6, C7}	10	0.215	1	0.130	0.441	0.22
18		{A3, B3, C3, D3, C1, C2, D2, B4, C4, D4}	10	0.199	1	0.043	0.408	0.22
19	D3	{C3, D3, E3, F3, G3, D4, D5, D6, D7, D8}	10	0.214	1	0.069	0.428	0.44
20		{D3, E3, F3, G3, H3, D4, E4, F4, G4, H4}	10	0.211	1	0.046	0.420	0.44

Table 6.2: Pattern shapes of middle group.

No	Pos	Middle pattern shapes	Length	Std Dev	Scat	App ratio	Imp	Sim
1	A1	{A1, B1, C1, D1, E1, A2, B2, C2, D2}	9	0.222	1	0.152	0.452	0.25
2		{A1, A2, A3, A4, A5, A6, A7, A8, B2, B7}	10	0.225	1	0.095	0.444	0.25
3	B1	{A1, A2, A3, A4, A5, B1, B2, B3, B4}	9	0.193	1	0.210	0.434	0.25
4		{A2, B2, C2, D2, E2, F2, G2, H2, B1, G1}	10	0.202	1	0.193	0.441	0.25
5	C1	{A1, B1, C1, D1, C2, C3, C4, C5, C6, C7}	10	0.210	1	0.120	0.434	0.33
6		{B1, C1, D1, E1, F1, B2, C2, D2, E2, F2}	10	0.221	1	0.121	0.446	0.33
7	D1	{B2, C2, D2, E2, F2, G2, H2, B1, C1, D1}	10	0.211	1	0.119	0.435	0.11
8		{D1, E1, F1, G1, H1, D2, D3, D4, D5, D6}	10	0.202	1	0.102	0.422	0.11
9	B2	{B1, B2, B3, B4, B5, B6, B7, B8}	8	0.117	0.929	0.706	0.444	0.14
10		{B1, C1, D1, B2, C2, D2, B3, C3, D3}	9	0.103	0.714	0.200	0.286	0.37
11		{A2, B2, C2, D2, E2, F2, G2, H2, B1, G1}	10	0.162	1	0.267	0.416	0.37
12	C2	{A2, B2, C2, D2, E2, F2, G2, H2, B3, C3}	10	0.197	1	0.109	0.419	0.37
13		{C1, D1, E1, C2, D2, E2, C3, D3, E3}	9	0.170	1	0.181	0.407	0.37
14		{A1, B1, C1, C2, C3, C4, C5, C6, C7, C8}	10	0.189	1	0.082	0.405	0.25
15	D2	{D1, E1, F1, G1, H1, D2, E2, F2, G2, H2}	10	0.201	1	0.048	0.410	0.11
16		{A2, B2, C2, D2, E2, A3, B3, C3, D3, E3}	10	0.186	1	0.077	0.401	0.11
17	C3	{A2, B2, C2, D2, E2, A3, B3, C3, D3, E3}	10	0.181	1	0.033	0.388	0.22
18		{C1, C2, C3, C4, C5, C6, C7, C8, B1, B2}	10	0.188	1	0.037	0.395	0.22
19	D3	{A2, B2, C2, D2, E2, A3, B3, C3, D3, E3}	10	0.186	1	0.018	0.390	0.33
20		{C1, C2, C3, C4, C5, D1, D2, D3, D4, D5}	10	0.194	1	0.018	0.398	0.33

6.2.5 Step 5. MCTS performance

Usually, it is expected that the performance in BTMM affects positively the performance of MCTS, as shown in Chapter 5. Here, we conducted a series of games with MCTS players using the learned values for each group of PSs. In this experiment, learned values are used only for the simulation part, so the effect is smaller than in the case of Chapter 5.

Table 6.3 shows the battle results of MCTS players against a program Reversi in C# which is downloaded from the link http://www.codeproject.com/Articles/4672/Reversi-in-C. The winning ratio of MCTS with better group is 62.6% with a 95% confidence interval of $\pm 3.5\%$. This is significantly better than $54.7\% \pm 3.6\%$, the case of MCTS with

Table 6.3: Pattern shapes of worse group.

No	Pos	Worse pattern shapes	Length	Std Dev	Scat	App ratio	Imp	Sim
1	A1	{A1, B1, C2, D3, E4, F5, G6, H7, H8}	9	0.171	1	0.174	0.405	0.12
2		{A1, A2, A3, A4, B4, C4, D4, E4, F4, G4}	10	0.173	1	0.080	0.389	0.12
3	B1	{C1, B1, B2, B3, B4, B5, B6, B7, B8}	9	0.147	1	0.404	0.428	0.12
4		{B1, C1, C2, C3, C4, C5, C6, C7, C8, D8}	10	0.140	1	0.179	0.376	0.12
5	C1	{D1, C1, C2, C3, C4, C5, C6, C7, C8, B8}	10	0.147	1	0.168	0.380	0.55
6		{C1, C2, C3, C4, C5, D1, D2, D3, D4, D5}	10	0.162	1	0.094	0.381	0.55
7	D1	{C1, C2, C3, C4, C5, D1, D2, D3, D4, D5}	10	0.138	1	0.120	0.362	0.44
8		{E1, D1, D2, D3, D4, D5, D6, D7, D8, C8}	10	0.133	0.933	0.150	0.350	0.44
9	B2	{B1, C1, D1, E1, B2, C2, D2, E2}	8	0.105	0.846	0.436	0.362	0.42
10		{B2, C2, D2, B3, C3, D3, B4, C4, D4}	9	0.047	0.571	0.231	0.207	0.37
11		{B1, B2, B3, B4, B5, C1, C2, C3, C4, C5}	10	0.093	0.933	0.173	0.314	0.37
12	C2	{C2, C3, C4, C5, C6, D2, D3, D4, D5, D6}	10	0.153	1	0.123	0.377	0.5
13		{C2, C3, C4, C5, C6, C7, C8, B8, D8}	9	0.117	1	0.262	0.369	0.5
14		{C2, C3, C4, C5, C6, B2, B3, B4, B5, B6}	10	0.140	1	0.138	0.368	0.5
15	D2	{D2, D3, D4, D5, D6, D7, D8, A8, B8, C8}	10	0.139	1	0.059	0.351	0.44
16		{E2, E3, E4, E5, E6, D2, D3, D4, D5, D6}	10	0.163	0.929	0.087	0.366	0.44
17	C3	{C3, C4, C5, C6, C7, A8, B8, C8, D8, E8}	10	0.157	0.929	0.057	0.354	0
18		{A1, B2, C3, D4, E5, F6, G7, H8, H7, G8}	10	0.148	0.833	0.016	0.318	0
19	D3	{D3, D4, D5, D6, D7, D8, E8, F8, G8, H8}	10	0.159	0.923	0.027	0.349	0.22
20		{B1, C2, D3, E4, F5, G6, D7, E6, G4, H3}	10	0.163	0.923	0.036	0.355	0.22

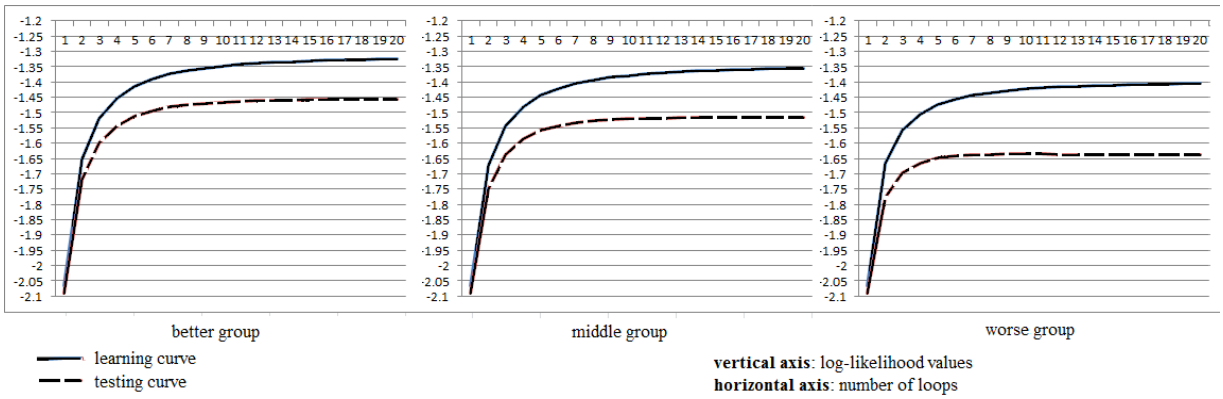


Figure 6.5: Learning and testing curves of three groups of pattern shapes.

middle group. Also the winning ratio of MCTS with worse group is significantly worse than the case of middle group.

From these results, it is confirmed that our measurements can select good pattern shapes, and selected ones are really good, not only for BTMM learning but also for MCTS player.

Table 6.4: performance of three MCTSs against Riversi in C#

Pattern shapes	Better group	Middle group	Worse group
MCTS Riversi (4 sec per move)	445 win games	392	435
Riversi (expert level)	272 win games	324	672
Result (MCTS Riversi / total games)	62.6 ± 3.5%	54.7 ± 3.6%	39.3 ± 2.7%

6.2.6 Additional Experiment

In the last experiment, each of better/middle/worse groups consist of 20 PSs. If the number of PSs is limited by some reasons, we should use better group. But if the number of PSs is unlimited, maybe it is more effective to use 30, 40 or more PSs, because the estimation model becomes more rich. Then, here 3 groups "better" group (20 PSs), "better + middle" group (40 PSs), and "better + middle + worse" group (60 PSs) are compared, by using BTMM performance and MCTS performance.

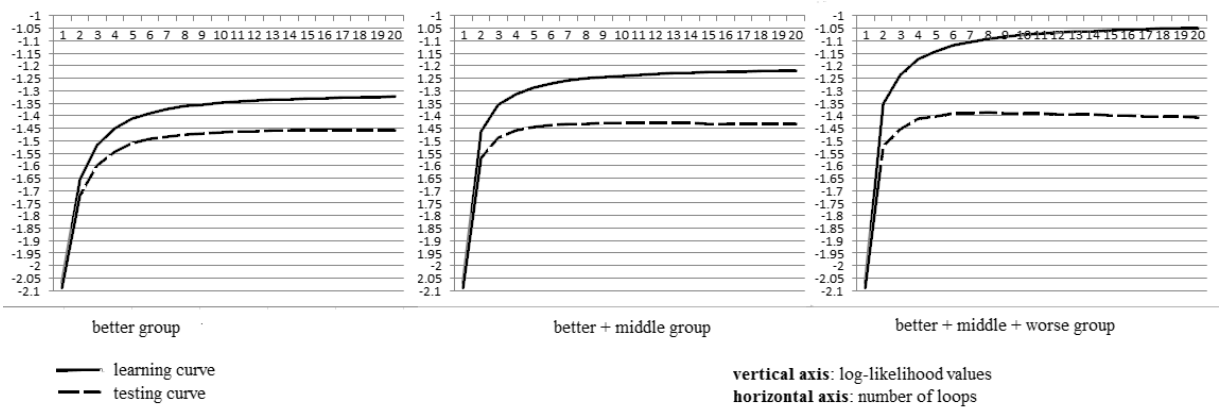


Figure 6.6: Learning and testing curves of three groups after combination.

Figure 6.6 is the result of BTMM learning, as Figure 6.5. About the learning-curves, "better + middle + worse" group is the best, because the model is the richest. However, about the testing-curves, the difference is very small though "better + middle + worse" is still the best. The big gap between the two curves and also the decrease of the testing-curve in the case of "better + middle + worse" are both signs of over-fitting.

Table 6.5 is the battle result of MCTS, as Table 7.2. Though "better + middle + worse" group is the best for the BTMM performance, the "better" group is the best for MCTS performance, because of the cost of computation. In the case of only using "better" group, 9,800 simulations can be done in 4 seconds. On the other hand, only 4,400 simulations can be done if using "better + middle + worse" group, because much more patterns are hit, then much more calculations are needed. It is well known that the number of simulations directly affects the performance of MCTS, then the balance of cost and accuracy is important to fix the number of features used.

Table 6.5: performance of three (light/middle/heavy) MCTSs against Riversi in C#

Pattern shapes	Better group	Better + Middle group	Better + Middle + Worse group
MCTS Riversi	456 (9800 simulations)	315 (5000 simulations)	396 (4400 simulations)
Riversi (expert)	272 win games	285	306
MCTS Riversi / total	62.6 ± 3.5%	52.5 ± 4%	56.4 ± 3.7%

6.3 Conclusion

In this Chapter, we introduced several measurements to evaluate a single pattern shape, using some statistics without learning. Three measurements, "selection ratio", "scattering", "appearance ratio" were proposed, and integrated to "importance" by using some weights. The effectiveness of "importance" was shown through experiments. A set of pattern shapes with higher "importance" is really better than others, not only in the BTMM performance, but also in the MCTS performance.

The cost for computing these values is much cheaper than that for executing light-BTMM proposed in Chapter 5, then it will be promising to use "importance" to discard unpromising candidates in evolution.

Chapter 7

CONCLUSION

This chapter summarizes the main ideas of the thesis: the goal, motivation, purpose, backgrounds, and contributions. The contributions are reviewed, and the future works are discussed.

The goal of our research is to improve the performance of Monte Carlo Tree Search (MCTS) programs for board games. Pattern shapes is a kind of feature that is useful for making an action evaluation function which supports and enhances MCTS. Thus, finding out the important pattern shapes in board games is the motivation and purpose of our research. Figure 7.1 shows the relationship between the motivation and the goal of our research.

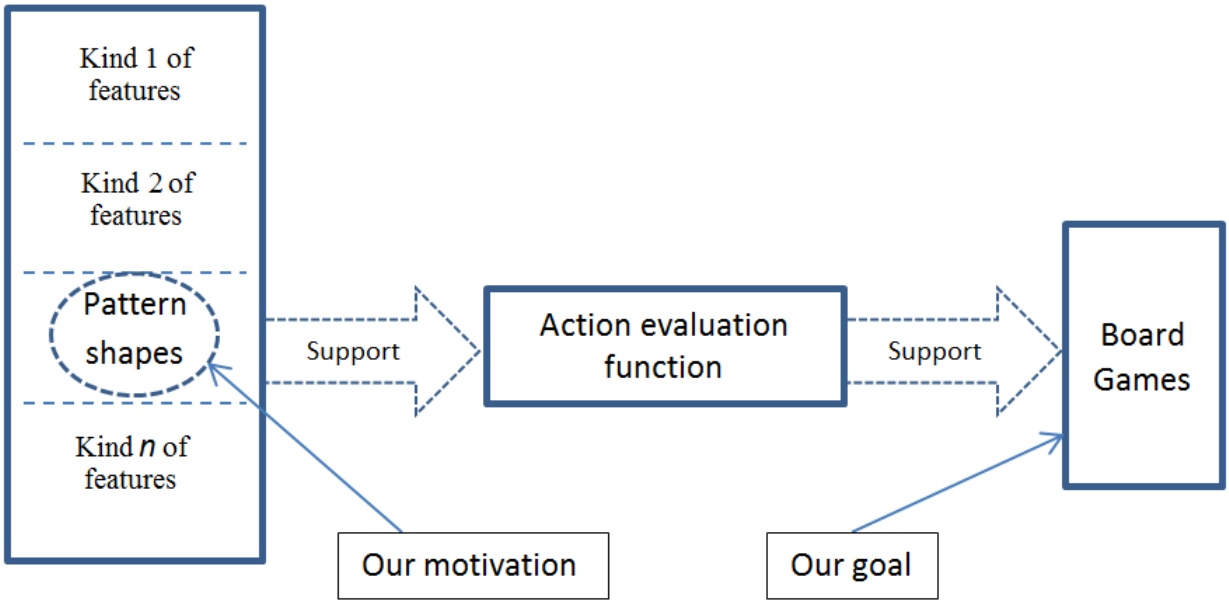


Figure 7.1: Motivation and Goal of research.

Based on the backgrounds such as game tree search, supervised learning and stochas-

tic optimization, we build up the methods for finding out the important pattern shapes. Figure 7.2 shows the relationships between the purpose, the backgrounds (presented in chapters 2, 3, 4), and our contributions (chapters 5 and 6). Chapter 2 presented an overview of Monte Carlo Tree Search. Chapter 3 presented an overview of machine learning and BTMM, a recent supervised learning algorithm for board games. Chapter 4 presented an overview of stochastic optimization and evolutionary algorithms, which forms the background of our approach in Feature Selection.

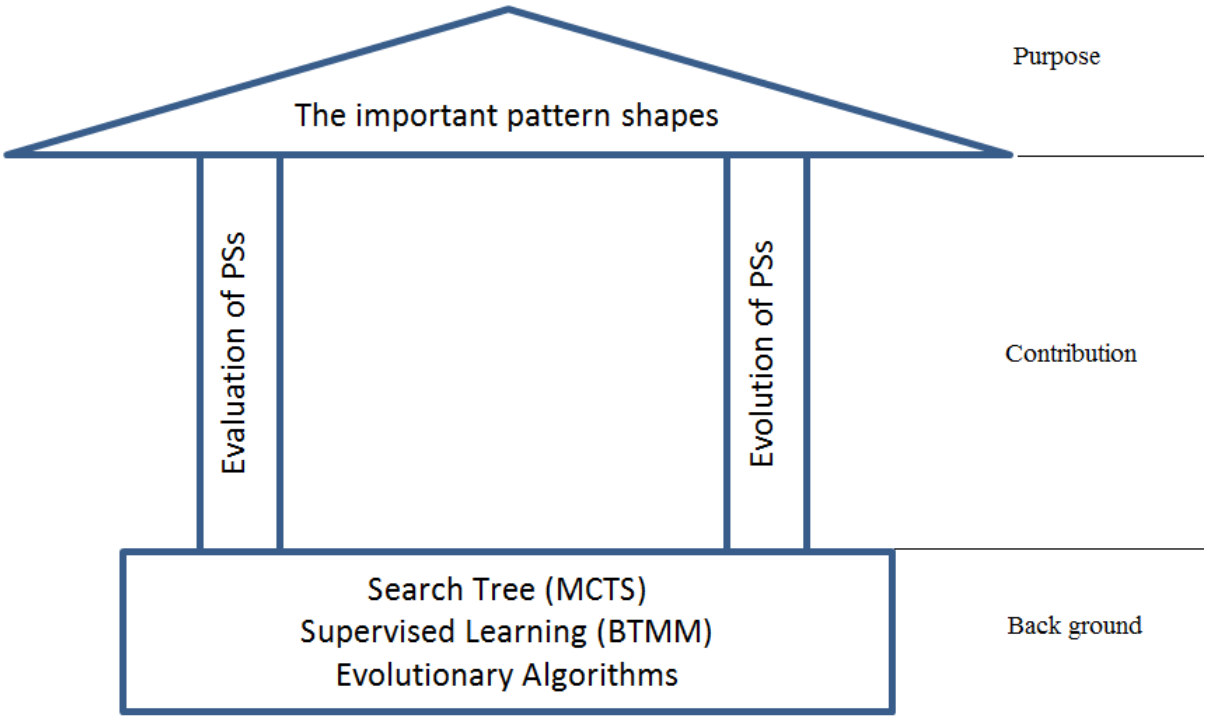


Figure 7.2: Purpose, contributions, and backgrounds.

Our contribution is the selection and evolution of pattern features for supervised learning of an action evaluation function in board games. This is also **the answer to the research question** in chapter 1. Chapter 5 presents the selection of pattern features, this work is done successfully with one accepted paper. Chapter 6 presents the evolution of pattern features, this work is also done with two published papers.

The evolution of pattern features is done by using BTMM performance with smaller number of training data and smaller number of loops, instead of the MCTS performance itself. Simulated annealing with this light evaluation function is used to optimize the pattern shapes. The input data is the default pattern shapes related to the cells of the game board, and the output is the improved pattern shapes. Then, these improved pattern shapes are applied in an MCTS program, and the experiments show that MCTS performance is improved by improving MCTS performance. One more interesting result is

the strangeness of the obtained pattern shapes, it supports that automated optimization of pattern shape is promising, because it will be very difficult to optimize them by hand.

For the selection of pattern features, we proposed measurements to evaluate pattern shapes such as the appearance ratio, the scattering, the standard deviation, the importance, and the similarity. The experiments showed that our method can evaluate and select good pattern shapes without learning. This method is particularly useful to reject unpromising candidates in the step of optimization.

We have a plan to expand our research for future work. Many kinds of options can be employed as the machine learning used, as the search algorithm used, and as the target game. For example, support vector machine or neural network, alpha-beta search, Connect-6, Go or Amazon can be employed. Also, we are going to establish a method to use rapid evaluation (in chapter 6) for rejecting unpromising candidates in evolution (in chapter 5).

Appendix

Adaptation of Game AIs

This section introduces one application that we used the Genetic Algorithm to generate "a group of rival AIs" having a suitable strength for the average human player. This application was published in SCIS-ISIS 2012 conference with title "Adaptation of Game AIs using Genetic Algorithm: Keeping Variety and Suitable Strength" [44].

Introduction of Our Application

We want to build the AIs that are weakened for average players. But these AIs must satisfy the following three requirements:

1. Strength of the AIs is almost the same as the target player.
2. Human players feel that the moves of AIs are natural.
3. AIs use various features as well as various strategies.

The system enables users to play with AIs of the same level, and furthermore it builds several AIs with different strategies, so that players can continue to play with high motivation. There are three components in our system.

1. A method to measure the strength of player from game records, and imitate him.
2. An Input-Output model that removes unnatural moves from legal moves.
3. An algorithm that generates various AIs whose strength are suitable to the player.

The proposed system can be applied widely to most games, but we used the game of Riversi as an experiment example. Besides, the rules of Riversi are simple, a lot of people know them, strong AIs is already available, but this kind of systems have not been researched well.

Figure 7.3 shows the proposed system, it mainly consists of two parts:

1. Game records of player is evaluated by a tester AI, and an agent AI which has the same strength is built.

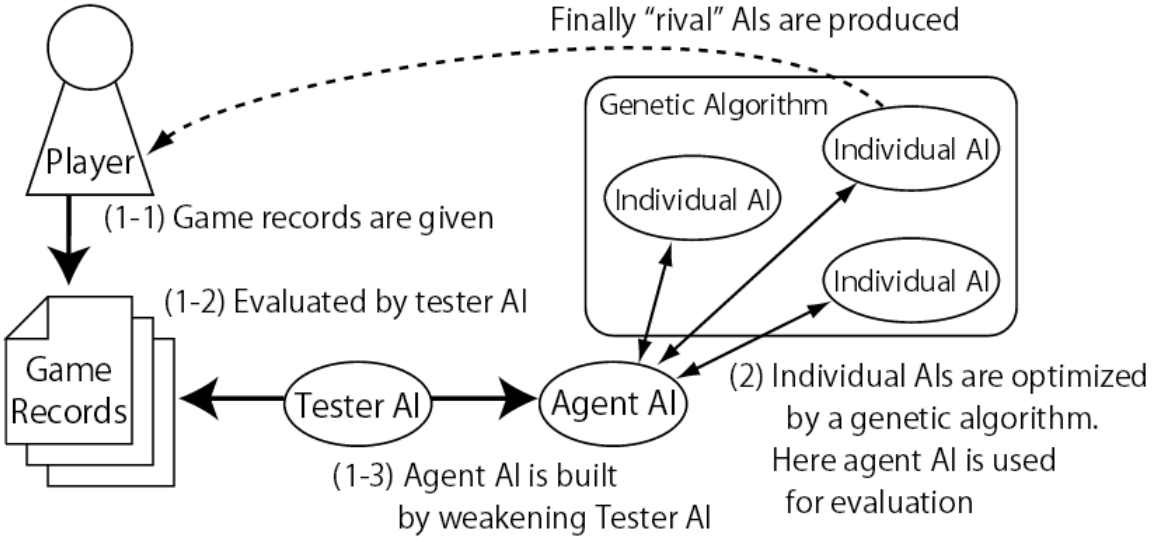


Figure 7.3: Our "rival AIs" generating system.

2. Individual AIs are optimized by using a genetic algorithm (GA) whose evaluation function is calculated from both the *winning ratio* against the agent AI and *varieties* of parameters.

Building Agent AI

This part is to measure the strength of player from game records, and to imitate him.

[**Evaluation and imitation of players**] The term "weakness" is defined by the averaged difference of evaluation values between the best move and the actual move, and it is utilized for evaluating and imitating the strength of the target player. The weakness μ is measured by a sufficiently strong AI, called the **tester AI**, and then the weakness is imitated by a controlled AI, called the **agent AI**. The main components of the agent AI are same as the tester AI, but the agent AI select a bad move so that the weakness is imitated. The concrete procedures are as follows.

1. At move t in the records of players, the best move a_t^* is computed by the Tester AI.

2. Both the best move a_t^* and the actual move a_t of the player are evaluated, and the difference of evaluation values $v(a_t^*) - v(a_t)$ is calculated.

3. Weakness is defined by the average of these differences, from the first move to 20^{th} move, $\mu = \frac{1}{20}\sum_{t=1}^{20} v(a_t^*) - v(a_t)$. When two or more games are available, μ is defined by the average of them.

4. After μ is computed, the agent AI is controlled so that its weakness is almost μ.

(a) For example, assume that μ is 20.

(b) All the legal moves are evaluated, and the differences (badness) of the evaluation values to the best move are calculated, for example {0,5,40,50}. If this is the first move, the move with badness 5 (nearest to 20) is selected.

(c) Assume that the differences of the next move is {0,15,30,50}. In this case, the move with badness 30 is selected instead of 15, because the average with the previously selected difference (17.5 instead of 10) is nearest to the target $\mu = 20$.

[**Configuration of Tester AI**] The following AI is used as the tester AI (and also as the agent AI by weakening):

- Search algorithm: minimax tree search without $\alpha\beta$ for attaining accurate value.
- Depth of search: 5
- Parameters of the evaluation function: parameters of Expert level of Riversi-in-C#

This tester AI is clearly stronger than standard beginners. In a preliminary experiment, we obtained 20 wins per 20 games.

The purpose of building an agent AI is to imitate the strength of a given player. In order to validate the robustness of the proposed method, we employ various kinds of AIs in the place of various human players, and build an agent AI for each of them. The experiment is successful if a target player and its corresponding agent AI as target players:

- $\alpha\beta_4$: Riversi in C# intermediate, depth 4 $\alpha\beta$ search
- $\alpha\beta_3$: Riversi in C# intermediate, depth 3 $\alpha\beta$ search
- UCT_1: MCTS with UCT, 1 second per move
- UCT_{10}: MCTS with UCT, 10 second per move
- UCT+: MCTS with UCT+, 10 second per move

200 games were done for each pair, between a target player and its agent AI. Table 7.1 shows that the agent AI and the target have almost the same strength in each case.

Optimization Using a Genetic Algorithm

[Input-Output model and tactical parameters] The input-output model of individual AI used $\alpha\beta$ method with a fixed depth and some heuristics. The evaluation function of a

Table 7.1: Comparison between the Agent AI with the target

Target player	weakness μ	Win ratio of agent AI
$\alpha\beta_4$	25.12	111-85 (0.57)
$\alpha\beta_3$	28.98	96-100 (0.49)
UCT_1	36.68	114-87 (0.57)
UCT_{10}	26.51	118-80 (0.59)
UCT+	21.47	97-99 (0.49)

board state is constructed from a linear sum of difference of "number of stones", "openness","number of legal moves" and "number of defined stones (the stones cannot be taken by opponents)". These concepts are relevant to the game of Riversi, and used in Reversi in C#.

Each individual of GA has its own weight for each term of the sum, except the weight of "number of stones" is fixed to 100 for all individuals. It implies that each individual has 3 free parameters.

Two simple heuristics that prevent unnatural moves from legal moves are also introduced:"if the AI can take the corner, then always take" and "prevent the opponent from taking the corner at the next turn if possible". These heuristics, specific to Riversi, were carefully selected by interviewing 10 beginner players.

[The genetic algorithm] The GA employed in this research is MGG-best2. The procedure of MGG-best2 is as follows.

1. A population of N_{pop} individuals (with 3-dimensional parameters) is generated. Each parameter is randomly selected from the range [1, 10000].

2. Two individuals are selected randomly from the population as parents.

3. A crossover operator is applied to the parents. N_{child} individuals are obtained as children. Details of this crossover are introduced in $BLX-\alpha$ method.

4. The fitness values of parents and the children are calculated. Details of the fitness calculation are introduced in $Fitness$.

5. The two best individuals of the family are selected and returned to the group, in place of the parent individuals.

6. Repeat N_{cycle} times from step 2.

[$BLX-\alpha$ **method**] $BLX-\alpha$[24] is employed as crossover operator. In $BLX-\alpha$, a child is generated by refering to parents chromosome coordinates and parameter α. Figure 7.4 shows an example of this crossover on 2-dimensions.

When the parents chromosome coordinates are (x_1, y_1) and (x_2, y_2), then the values $max_{cx}, min_{cx}, max_{cy}, min_{cy}$ can be defined by the following equations.

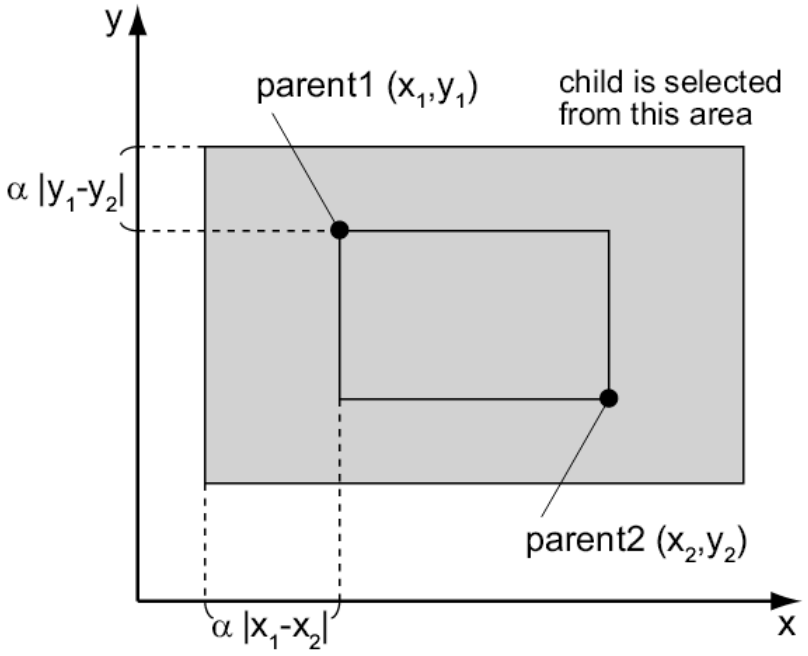


Figure 7.4: $BLX - \alpha$ generates child in (hyper-)rectangle.

- $max_{cx} = max(x_1, x_2) + \alpha \mid x_1 - x_2 \mid$
- $min_{cx} = min(x_1, x_2) - \alpha \mid x_1 - x_2 \mid$
- $max_{cy} = max(y_1, y_2) + \alpha \mid y_1 - y_2 \mid$
- $min_{cy} = min(y_1, y_2) - \alpha \mid y_1 - y_2 \mid$

The child chromosome coordinates x is selected randomly from $[min_{cx}, max_{cx}]$, and y is selected randomly from $[min_{cy}, max_{cy}]$. The actual crossover used in this application is a variant of this procedure, where the crossover is computed on the logarithm of coordinates instead of directly on the coordinates themselves. First, the logarithm of the coordinates is computed, then the crossover is computed, and finally the children coordinates are obtained by exponentiation. In practice, it leads to better results of the GA.

If α is 0 then all children are inside their parents coordinates, but if α is larger than 0 then a child can be outside of its parents coordinates. It means this method holds both crossover and mutation at once. In the experiment, we fixed $\alpha = 0.5$.

[*Fitness*] The design of the fitness function is an essential part to achieve the goal. In our case, the natural moves are already achieved by two heuristics, then measurement about the strength closeness and about strategy variety must be included in the fitness function.

$$f = f_{wr} - \beta f_{sim} \tag{7.1}$$

Equation 7.1 calculates the fitness of each AI. f_{wr} is a fitness concerning the winning ratio, f_{sim} is a penalty concerning the parameters variety, and $\beta(> 0)$ is a fixed constant weight to balance f_{wr} and f_{sim}. The AI becomes better with bigger f_{wr} and smaller f_{sim}.

The fitness concerning the winning ratio is given by equation 7.2. It shows how similar the strength of an individual AI is to the agent AI. The best value of f_{wr} is 1.0 when $wr = 0.5$, and the worst value of f_{wr} is 0.0 when $wr = 0.0$ or $wr = 1.0$.

$$f_{wr} = 4 \times (0.5 - \mid 0.5 - wr \mid)^2 \tag{7.2}$$

On the other hand, the penalty concerning the parameters variety is given by equation 7.3. n is the total number of different individuals in the population and (x_i, y_i, Z_i) are the parameters of individual i. The penalty for an individual i gets bigger when its parameters are closer to other individuals.

$$f_{sim} = \frac{1}{n} \sum_{j \neq i} \frac{1}{d^2(x_i, x_j) + d^2(y_i, y_j) + d^2(z_i, z_j)} \tag{7.3}$$

where $d^2(w_i, w_j) = (log_{10} w_i - log_{10} w_j)^2$

Conclusion of our application

The performance of the proposed method was evaluated through several experiments. The effectiveness of the heuristics for the natural moves was clear. The strength control was successful on the average, but not sufficient for weaker players, because their strength decreases after playing too many games. some dynamic information such as the advantage of the current situation could be used to control the strength more carefully. The variety of strategies used by the AIs was correctly identified by human players, but less significant than our expectation. One possible explanation is that the selected three features were too advanced for beginner/intermediate players. Features easier to identify by such players could be introduced such as “a tendency to play on the edge of the board”.

Some additional experiments for the importance measure

This section introduces some additional experiments to show the robustness of the importance measure.

A comparison with Logistello pattern shapes

This experiment confirms that pattern shapes of the better group are better than the pattern shapes selected by experts. The stream is also the same the main experiment. However, the experiment is implemented in 2 groups as Table 7. The mark "x" in Table 7 indicates that two pattern shapes corresponding to two groups are the same.

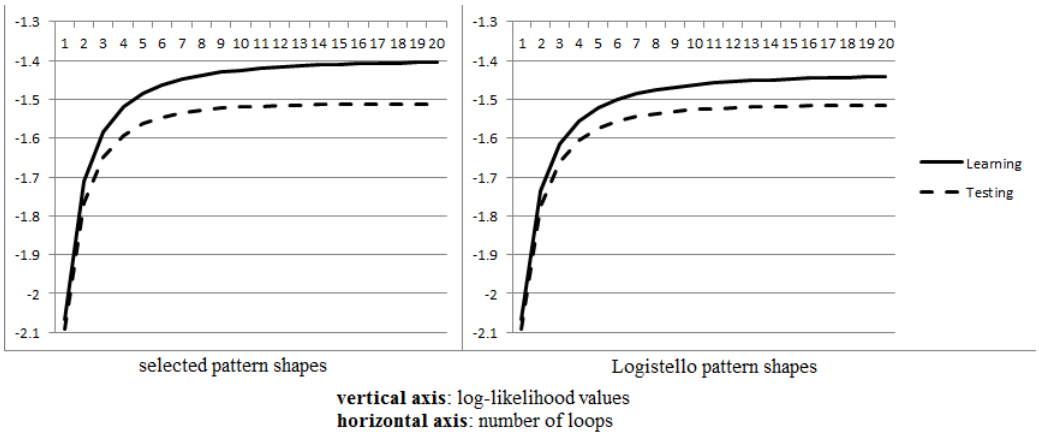


Figure 7.5: Comparison between selected pattern shapes and Logistello.

Logistello is the famous strong Riversi program, M. Buro used some pattern shapes which were selected manually by expert knowledge. We can refer these pattern shapes in [11], in this paper, he used the term "pattern" to define such pattern shapes.

After learning the game records by using BTMM, we can see in figure 7.5 that the selected pattern shapes are not worse than that Logistello pattern shapes. It means that the selected pattern shapes according to the importance are important ones.

Table 7.2: Selected pattern shapes and Logistello.

No	Pos	Selected pattern shapes (1)	Imp (1)	Imp (2)	Logistello pattern shapes (2)
1	A1	{A1, B1, C1, A2, B2, C2, A3, B3, C3}	x	x	{A1, B1, C1, A2, B2, C2, A3, B3, C3}
2		{A1, A2, A3, A4, A5, A6, A7, A8, B1, B2}	0.444	0.444	{A1, B1, C1, D1, E1, F1, G1, H1, B2, G2}
3	B1	{A1, B1, C1, A2, B2, C2, A3, B3, C3}	x	x	{A1, B1, C1, A2, B2, C2, A3, B3, C3}
4		{A1, B1, C1, D1, E1, F1, G1, H1, A2, B2}	0.473	0.473	{A1, B1, C1, D1, E1, F1, G1, H1, B2, G2}
5	C1	{A1, B1, C1, D1, A2, B2, C2, D2, E2, F2}	0.453	0.463	{A1, B1, C1, D1, E1, A2, B2, C2, D2, E2}
6		{A1, B1, C1, D1, E1, F1, G1, H1, B2, G2}	x	x	{A1, B1, C1, D1, E1, F1, G1, H1, B2, G2}
7	D1	{A1, B1, C1, D1, E1, F1, G1, H1, B2, G2}	x	x	{A1, B1, C1, D1, E1, F1, G1, H1, B2, G2}
8		{A1, B1, C1, D1, A2, B2, C2, D2, E2, F2}	0.437	0.463	{A1, B1, C1, D1, E1, A2, B2, C2, D2, E2}
9	B2	{A1, B1, A2, B2, C2, D2, B3, C3, B4}	0.459	0.433	{A1, B1, C1, A2, B2, C2, A3, B3, C3}
10		{A1, B2, C3, D4, E5, F6, G7, H8, A2, B1}	0.470	0.406	{A1, B1, C1, D1, E1, A2, B2, C2, D2, E2}
11	C2	{B1, C2, D3, E4, F5, G6, H7, C1, B2}	0.426	0.420	{A1, B1, C1, A2, B2, C2, A3, B3, C3}
12		{A1, B1, C1, D1, E1, A2, B2, C2, D2, E2}	x	x	{A1, B1, C1, D1, E1, A2, B2, C2, D2, E2}
13	D2	{A1, B1, C1, D1, E1, A2, B2, C2, D2, E2}	x	x	{A1, B1, C1, D1, E1, A2, B2, C2, D2, E2}
14		{A2, B2, C2, D2, E2, F2, G2, H2}	x	x	{A2, B2, C2, D2, E2, F2, G2, H2}
15	C3	{C3, D3, E3, F3, G3, H3, C4, C5, C6}	0.460	0.376	{A1, B1, C1, A2, B2, C2, A3, B3, C3}
16		{A3, B3, C3, D3, C1, C2, C4, D4}	0.424	0.303	{A1, B2, C3, D4, E5, F6, G7, H8}
17	D3	{C3, D3, E3, F3, G3, D4, D5, D6}	0.441	0.429	{A3, B3, C3, D3, E3, F3, G3, H3}
18		{D1, D2, D3, D4, D5, D6, D7, D8}	x	x	{D1, D2, D3, D4, D5, D6, D7, D8}

The necessity of w_{appr} parameter

This experiment confirms the necessity of the *app_ratio* weighting w_{appr}. This parameter can be tuned so that the performance of learning is the best. Thus, the shape of testing curve will be the mountain shape by the variants of w_{appr} parameter.

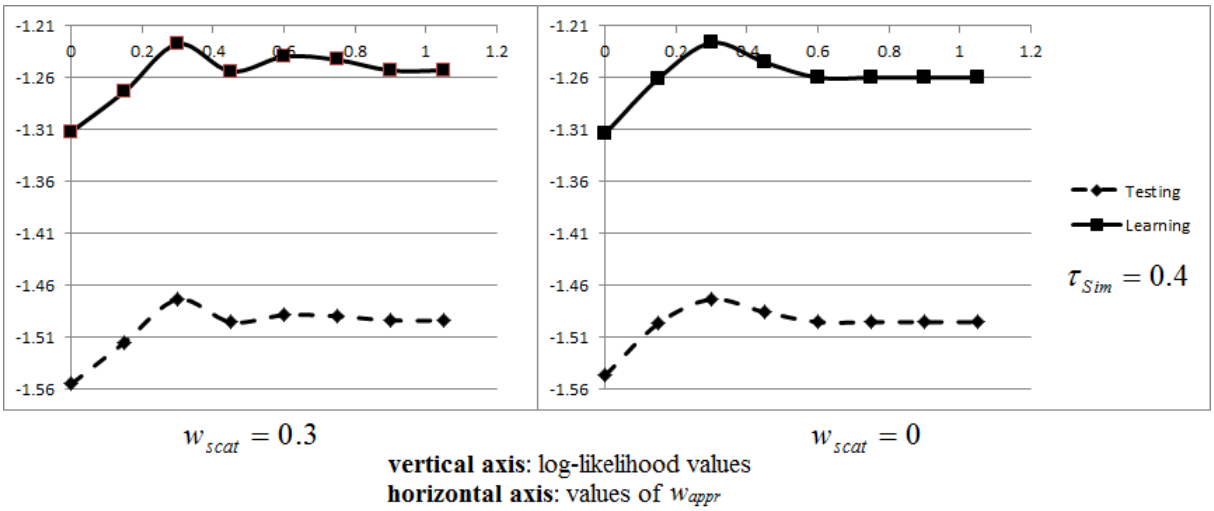


Figure 7.6: Evidence of the necessity of the w_{appr} parameter.

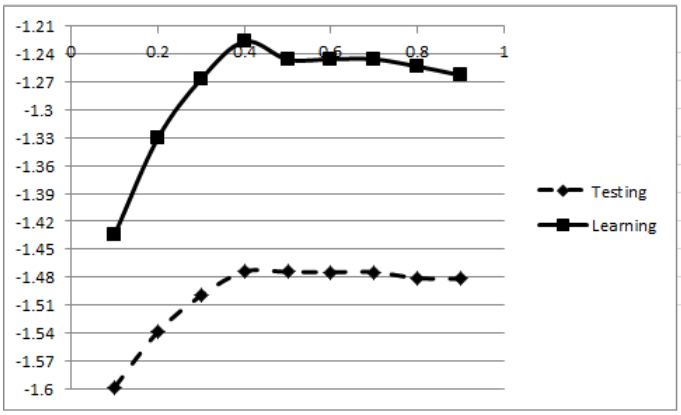
The pattern shapes are extracted from following conditions: The data set was made

of the 100 000 games, $\tau_{sim} = 0.4$, $\tau_{Lim} = 3$. The experiment is run for three scattering weighting $w_{scat} = 0.3$, and 0 as you can see in figure 7.6. Each value of w_{scat}, the weighting of app_ratio is tuned from 0 to 1.

Learning the game records by using BTMM is implemented from a data set of 30000 game records, and test in 3000 game records. Each BTMM optimization runs 20 loops. The experiment result reaches our expectation because following reasons: The result must be converged to good point if the best parameter is selected, and the mountain curve is proved the necessity of w_{appr} parameter. If the value of w_{appr} is too high, maybe many pattern shapes having bad standard deviations are selected. If the value of w_{appr} is too low, maybe many pattern shapes having low appearances are selected. The two last cases make the method fail.

The necessity of the similarity

This experiment confirms the necessity of the similarity (τ_{sim}). The experiment conditions like as the experiment for necessity of w_{appr} parameter, but the $w_{appr} = 0.3$, the $w_{scat} = 0$, the τ_{sim} is from 0.1 to 0.9, $\tau_{Lim} = 3$. The experiment result also reaches our expectation (see figure 7.7). The lower the value of τ_{sim} is, the smaller the number of pattern shapes will be because the selection condition is too strict in this case. For the very high of τ_{sim} values, although the importance values are higher, but too many similar pattern shapes are selected, maybe.



τ_{Sim}	Testing	Learning
0.1	-1.5983	-1.4337
0.2	-1.5388	-1.3302
0.3	-1.4998	-1.2668
0.4	-1.4739	-1.2264
0.5	-1.4744	-1.2458
0.6	-1.4746	-1.2454
0.7	-1.4746	-1.2454
0.8	-1.481	-1.2532
0.9	-1.4816	-1.2627

$\tau_{Lim} = 3$

vertical axis: log-likelihood values **horizontal axis**: values of τ_{Sim}

Figure 7.7: Evidence of the necessity of the τ_{sim} parameter.

Bibliography

[1] Arneson, B., Hayward, R., Henderson, P.: Mohex wins hex tournament. In: Int. Comp. Games Assoc. J., vol. 32, no. 2. pp. 114–116. IEEE (2009)

[2] Baier, H., Winands, M.H.: Monte-carlo tree search and minimax hybrids. In: Proceedings of CIG2013. pp. 129 – 137 (2013)

[3] Bouzy, B., Cazenave, T.: Computer go: An ai oriented survey. Artificial Intelligence 132, 39–103 (2001)

[4] Bouzy, B., Chaslot, G.: Bayesian generation and integration of k-nearest-neighbor patterns for 19x19 go. In: IEEE 2005 Symposium on Computational Intelligence in Games. pp. 1019–1025 (2005)

[5] Bradski, G., Kaehler, A.: Learning open cv. In: Learning Open CV, Chapter 13. O'Reilly Media (2008)

[6] Brown, G., Pocock, A., Zhao, M.J., Lujan, M.: Conditional likelihood maximisation: A unifying framework for information theoretic feature selection. Journal of Machine Learning Research 13, 27–66 (2012)

[7] Browne, C., Powley, Whitehouse, Lucas, Cowling, Tavener, Perez, Samothrakis, Colton: A survey of monte carlo tree search methods. IEEE transactions on computational intelligence and AI in games 4, 1 – 43 (2012)

[8] Brugmann, B.: Monte carlo go. In: AAAI Symposium on Games (1993)

[9] Buro, M.: Experiments with multi-probcut and a new high-quality evaluation function for othello. In: Games in AI Research, Maastricht, The Netherlands (2000)

[10] Buro, M.: Improving mini-max search by supervised learning. Artificial Intelligence 134, 85–99 (2002)

[11] Buro, M.: The evolution of strong othello programs. In: The International Federation for Information Processing Volume 112. pp. 81 – 88 (2003)

[12] Campbell, M., Hoane, J., Hsu, F.H.: Deep blue. Artificial Intelligence 134, 57–83 (2002)

[13] Chandrashekar, G., Sahin, F.: A survey on feature selection methods. Computers and Electrical Engineering 40, 16–28 (2014)

[14] Chaslot, G., Winands, M., Herik, H., Uiterwijk, J., Bouzy, B.: Progressive strategies for monte-carlo tree search. In: New Math. Nat. Comput., vol. 4, no. 3. p. 343 357. WorldScientific (2008)

[15] Coquelin, P.A., Munos, R.: Bandit algorithms for tree search. CoRR abs/cs/0703062 (2007)

[16] Coulom, R.: Computing elo ratings of move patterns in the game of go. ICGA Journal 30, 198–208 (2007)

[17] Coulom, R.: Efficient selectivity and backup operators in monte-carlo tree search. In: Computers and Games. pp. 72–83 (2007)

[18] Dang, T.T., Shirai, K.: Machine learning approaches for mood classification of songs toward music search engine. In: International Conference on Knowledge and Systems Engineering. pp. 144–149 (2009)

[19] Debuse, J.C., Rayward-Smith, V.J.: Feature subset selection within a simulated annealing data mining algorithm. Journal of Intelligent Information Systems 9, 57–81 (1997)

[20] Drake, P., Uurtarno, S.: Move ordering vs heavy playouts: Where should heuristics be applied in monte carlo go. In: Proc. 3rd North Amer. Game-On Conf., Gainesville, Florida. pp. 35–42 (2007)

[21] Eiben, A., Smith, J.: Introduction to evolutionary computing. In: Natural Computing Series. Springer, Berlin, Heidelberg, New York (2003)

[22] Emmanouilidis, C., Hunter, A., MacIntyre., J.: A multiobjective evolutionary setting for feature selection and a commonality-based crossover operator. In: Congress on Evolutionary Computing. pp. 309–316 (2000)

[23] Enzenberger, M., Müller, M., Arneson, B., Segal, R.: Fuego - an open-source framework for board games and go engine based on monte-carlo tree search. In: IEEE Trans. Comp. Intell. AI Games, vol. 2, no. 4. pp. 259–270. IEEE (2010)

[24] Eshelman, L.J., Schaffer, J.D.: Real-coded genetic algorithms and interval-schemata. In: FOGA (1992)

[25] Farmer, M., Bapna, S., Jain, A.: Large scale feature selection using modified random mutation hill climbing. In: Pattern Recognition, ICPR 2004. pp. 1051–4651 (2004)

[26] Fleming, P., Purshouse, R.: Evolutionary algorithms in control systems engineering: a survey. Control Engineering Practice 10, 1223–1241 (2002)

[27] Fonseca, C., Fleming, P.: An overview of evolutionary algorithms in multiobjective optimization. Evolutionary Computation 3, 1–16 (1995)

[28] Gauci, J., Stanley, K.O.: Indirect encoding of neural networks for scalable go. In: Proceedings of the 11th International Conference on Parallel Problem Solving From Nature. pp. 354–363 (2010)

[29] Gelly, S., Silver, D.: Combining online and offline knowledge in uct. In: ICML 07: Proceedings of the 24th Internatinal Conference on Machine Learning. pp. 273–280. ACM (2007)

[30] Gelly, S., Silver, D.: Achieving master level play in 9×9 computer go. In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008). pp. 1537 – 1540. AAAI (2008)

[31] Gelly, S., Wang, Y.: Exploration exploitation in go: Uct for monte-carlo go. In: Proc. Adv. Neur. Inform. Process. Syst (2006)

[32] Georgios, Yannakakis, N., Hallam, J.: Evolving opponents for interesting interactive computer games (2004)

[33] Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. Journal of Machine Learning Research 3, 1157–1182 (2003)

[34] Harada, K., Ikeda, K., Kobayashi, S.: Hybridization of genetic algorithm and local search in multiobjective function optimization: recommendation of ga then ls. In: GECCO 2006. pp. 667–674 (2006)

[35] Hoki, K., Kaneko, T.: The global landscape of objective functions for the optimization of shogi piece values with a game-tree search. LNCS 7168, 184–195 (2012)

[36] Huang, C.L., Wang, C.J.: A ga-based feature selection and parameters optimization for support vector machines. Expert Systems with Application 31, 231–240 (2006)

[37] Huang, S.C.: New heuristics for monte carlo tree search applied to the game of go. In: PhD Thesis. National Taiwan Normal University (2011)

[38] Huy, N.Q., Ikeda, K.: Evaluation of pattern shapes in board games before machine learning. International Journal Of Electical Engineering 20, 39 – 49 (2013)

[39] Huy, N.Q., Le, B., Ikeda, K.: Extracting important patterns for building state-action evaluation function in othello. In: Proceedings of the Technologies and Applications of Artificial Intelligence (TAAI). pp. 278–283 (2012)

[40] Huy, N.Q., Viennot, S., Ikeda, K.: Fast optimization of the pattern shapes in board games with simulated annealing (2014)

[41] Iida, H., Sakuta, M., Rollason, J.: Computer shogi. Artificial Intelligence 134, 121–144 (2002)

[42] Ikeda, K., Viennot, S.: Efficiency of static knowledge bias in monte-carlo tree search. In: Computers and Games 2013 (2013)

[43] Ikeda, K.: Exemplar-based direct policy search with evolutionary optimization. In: Congress on Evolutionary Computing. pp. 2357–2364 (2005)

[44] Ikeda, K., Tanaka, Y., Viennot, S., Quoc, N.H., Ueda, Y.: Adaptation of game ais using genetic algorithm: Keeping variety and suitable strength. In: Proceedings of SCIS 2012. pp. 945 – 951. IEEE (2012)

[45] Inagaki, K., Nakano, R.: Learning evaluation functions of shogi positions from different sets of games. LNCS 4694, 210–217 (2007)

[46] Inagaki, K., Nakano, R.: Large-scale optimization for evaluation functions with minimax search. Journal of Artificial Intelligence Research 4694, 527–568 (2014)

[47] Kaji, H., Ikeda, K., Kita, H.: Uncertainty of constraint function in evolutionary multi-objective optimization. In: Congress on Evolutionary Computing. pp. 1621–1628 (2009)

[48] Kloetzer, J., Iida, H., Bouzy, B.: The monte-carlo approach in amazons. In: Computer Game Workshop, Amsterdam, The Netherland (2007)

[49] Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: ECML-06. Number 4212 in LNCS. pp. 282 – 293. Springer (2006)

[50] Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2. pp. 1137–1143. IJCAI'95, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1995), http://dl.acm.org/citation.cfm?id=1643031.1643047

[51] Kool, A., Zavrel, J., Daelemans, W.: Simultaneous feature selection and parameter optimization for memory-based natural language processing. pp. 93–100 (2000)

[52] Lee, C., Wang, M., Chaslot, G., Hong, T.P.: The computational intelligence of mogo revealed in taiwans computer go tournaments. In: IEEE Trans. Comp. Intell. AI Games vol. 1, no. 1. pp. 73–89. IEEE (2009)

[53] Mandziuk, J., Osman, D.: Alpha-beta search enhancements with a real-value game-state evaluation function. ICGA Journal 27, 38–43 (2004)

[54] Marocco, D., Floreano, D.: Active vision and feature selection in evolutionary behavioral systems. In: Proceedings of the Seventh International Conference on Simulation of Adaptive Behavior on From Animals to Animats. pp. 247–255. ICSAB, MIT Press, Cambridge, MA, USA (2002)

[55] Meiri, R., Zahavi, J.: Using simulated annealing to optimize the feature selection problem in marketing applications. European Journal of Operational Research 171, 842–858 (2006)

[56] Mller, F., Spth, C., Geier, T., Biundo, S.: Exploiting expert knowledge in factored pomdps. In: ECAI'12. pp. 606–611 (2012)

[57] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. CoRR abs/1312.5602 (2013)

[58] Mycielsk, J.: Games with perfect information. In: Handbook of Game Theory. Elsevier (1992)

[59] Navot, A.: On the role of feature selection in machine learning. In: PhD Thesis. the Senate of the Hebrew University (2006)

[60] Nguyen, H., Franke, K., Petrovic, S.: Optimizing a class of feature selection measures. In: NIPS 2009 Workshop on Discrete Optimization in Machine Learning. pp. 667–674 (2009)

[61] Nguyen, M.L., Shimazu, A.: A semi supervised learning model for mapping sentences to logical forms with ambiguous supervision. Data and Knowledge Engineering 90, 1–12 (2014)

[62] Peng, H., Long, F., , Ding, C.: Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions on Pattern Analysis and Machine Intelligence 27, 1226–1238 (2005)

[63] Perez, D. ; Rohlfshagen, P..L.S.: Monte carlo tree search: Long-term versus short-term planning. In: Computational Intelligence and Games (CIG). pp. 219 – 226 (2012)

[64] Saito, J.T., Winands, M.H., Uiterwijk, J.W., van den Herik, H.J.: Grouping nodes for monte-carlo tree search. In: Computer Game Workshop, Amsterdam, The Netherland (2007)

[65] Schaeffer, J., van den Herik, H.J.: Games, computers, and artificial intelligence. Artificial Intelligence 134, 1–7 (2002)

[66] de Schaetzen, V., Molter, C., Coletta, A., Steenhoff, D., Meganck, S., Taminau, J., Lazar, C., Duque, R., Bersini, H., Nowe, A.: A survey on filter techniques for feature selection in gene expression microarray analysis. IEEE/ACM Transactions on Computational Biology and Bioinformatics 9(4), 1106–1119 (2012)

[67] Schaul, T., Schmidhuber, J.: Scalable neural networks for board games. LNCS 5768, 1005–1014 (2009)

[68] Shay, C., Eytan, R., Gideon, D.: Feature selection based on the shapley value. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence. pp. 665–670. IJCAI'05, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005), http://dl.acm.org/citation.cfm?id=1642293.1642400

[69] Shih-Chieh, H., Rémi, C., Shun-Shii, L.: Monte-carlo simulation balancing in practice. In: Proceedings of the 7th International Conference on Computers and Games. pp. 81 – 92. CG'10, Springer-Verlag, Berlin, Heidelberg (2011), http://dl.acm.org/citation.cfm?id=1950322.1950330

[70] Silva, R.G.O., de Souza Ribeiro, M.W., do Amaral, L.R.: Building high level knowledge from high dimensionality biological dataset (nci60) using genetic algorithms and feature selection strategies. In: Congress on Evolutionary Computing. pp. 578 – 583 (2013)

[71] Silver, D., Muller, M.: Reinforment learning of local shape in the game of go. In: Proceedings of the 20th international joint conference on Artifical intelligence. pp. 1053 – 1058. Springer (2007)

[72] Singh, V., Pathak, S.: Feature selection using classifier in high dimensional data. CoRR abs/1401.0898 (2014)

[73] Stern, D., Herbrich: Bayesian pattern ranking for move prediction in the game of go. In: Proceedings of the 23rd international conference on Machine learning. pp. 873 – 880 (2006)

[74] Tesauro, G.: Programming backgammon using self-teaching neural nets. Artificial Intelligence 134, 181–199 (2002)

[75] Tesfatsion, L.: Agent-based computational economics: A constructive approach to economic theory. Handbook of Computational Economics 2, 831–880 (2006)

[76] To, H.V., Ichise, R., Le, H.B.: An adaptive machine learning framework with user interaction for ontology matching. In: The IJCAI 2009 Workshop on Information Integration on the Web. pp. 35–40 (2009)

[77] Tom, D., Mller, M.: A study of uct and its enhancements in an artificial game. Advances in Computer Games 6048, 55–64 (2010)

[78] Tsuruoka, Y., Yokoyama, D., Chikayama, T.: Game-tree search algorithm based on realization probability. ICGA Journal 25, 146–153 (2002)

[79] Vilkas, E.: Axiomatic definition of the value of a matrix game. In: Theory Probabl. Appl. (1963)

[80] Wang, Y., Gelly, S.: Modifications of uct and sequence-like simulations for monte-carlo go. In: Proc. IEEE Symp. Comput. Intell. Games, Honolulu, Hawaii. pp. 175–182. IEEE (2007)

[81] Whitley, D.: An overview of evolution algorithms: practical issues and common pitfalls. Information and Software Technology 43, 817–831 (2001)

[82] Wu, T.: Feature selection in speech and speaker recognition. In: PhD Thesis (2009)

[83] Xuan, H.N.T., Le, A.C., Nguyen, M.L.: Linguistic features for subjectivity classification. In: International Conference on Asian Language Processing. pp. 17–20 (2012)

[84] Zhang, R., Liu, C., Wang, C.: Research on connect 6 programming based on mtd(f) and deeper-always transposition table. In: Proceedings of Cloud Computing and Intelligent Systems (CCIS) 2012. pp. 206 – 208. IEEE (2012)

Publications

Journal articles

N.Q.Huy, K.Ikeda: Evaluation of pattern shapes in board games before machine learning. In: International Journal Of Electical Engineering, Vol 20, No. 2. pp. 39-49. IJEE (2013)

International conferences

N.Q.Huy, K. Ikeda, L.H. Bac: Extracting Important Patterns for Building State-Action Evaluation Function in Othello. In: Proceedings of the 2012 Conference on Technologies and Applications of Artificial Intelligence, pp. 278-283, (TAAI 2012).

K. Ikeda, Y. Tanaka, S. Viennot, N.Q. Huy, Y. Ueda: Adaptation of Game AIs using Genetic Algorithm: Keeping Variety and Suitable Strength. In: Proceedings of SCIS 2012, pp. 945-951, (2012).

Huy Nguyen, Simon Viennot, Kokolo Ikeda: Fast Optimization of the Pattern Shapes in Board Games with Simulated Annealing, submited in KSE 2014 (accepted).

Other publications (not in thesis)

L.H. Bac, N.Huy, B. Vo: An efficient strategy for mining high utility itemsets. In: International Journal of Intelligent Information and Database Systems, Vol 5, No 2, pp. 143 - 163 (2011).

H.Iida, T.Nakagawa, N.Huy, S.Hasai, A.Husan, A.Muangkasem, S.Sone, T.Ishitobi. Game information dynamics and its applications in Congkak and Othello. In: Proceedings of the International Conference on Information Society (i-Society2012), pp. 415 - 422, (2012).

L.H. Bac, N.Huy, B. Vo. Efficient Algorithms for Mining Frequent Weighted Itemsets from Weighted Items Databases. In: Proceedings of RIVF 2010, pp. 1 - 6 (2010).

B. Vo, N.Huy, L.H. Bac. Mining High Utility Itemsets from Vertical Distributed Databases. In: Proceedings of RIVF 2009, pp. 1 - 4 (2009).

B. Vo, N.Huy, T.B. Ho, L.H. Bac. Parallel Method for Mining High Utility Itemsets from Vertically Partitioned Distributed Databases. In: Proceedings of KES 2009, pp. 251 - 260, (2009).

L.H. Bac, N.Huy, T.A. Cao, B. Vo. A Novel Algorithm for Mining High Utility Itemsets. In: Proceedings of ACIIDS 2009, pp. 13 - 17, (2009)