國立交通大學

資訊科學與工程研究所

博 士 論 文

隨機型完全資訊遊戲對局程式之強度分析研究

On Strength Analyses of Computer Programs for Stochastic Games with Perfect Information

研 究 生：薛筑軒

指導教授：吳毅成　教授

中 華 民 國 一零八 年 三 月

隨機型完全資訊遊戲對局程式之強度分析研究
On Strength Analyses of Computer Programs for Stochastic Games with Perfect Information

研 究 生：薛筑軒　　Student：Chu-Hsuan Hsueh

指導教授：吳毅成　　Advisor：I-Chen Wu

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

博 士 論 文

A Dissertation

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

March 2019

Hsinchu, Taiwan

中華民國 一零八 年 三 月

隨機型完全資訊遊戲對局程式之強度分析研究

研究生：薛筑軒　　指導教授：吳毅成 博士

國立交通大學資訊科學與工程研究所博士班

摘要

對於人工智慧（Artificial Intelligence）的研究，電腦遊戲是一個重要的領域。遊戲可以依據所牽涉的機率事件（Chance）兩種不同的作用分為：決定型（Deterministic）對比隨機型（Stochastic）以及完全資訊（Perfect Information）對比不完全資訊（Imperfect Information）。由於許多真實世界的問題會牽涉到不確定性，隨機型遊戲以及不完全資訊遊戲是非常值得研究的。本論文以隨機型完全資訊遊戲為研究主題，因為此類型遊戲相較於不完全資訊遊戲更容易模擬。暗棋以及一個簡化且已解的變種 2×4 暗棋是其中兩個此類型的遊戲，也是本論文主要研究的遊戲。

本論文首先使用數個現存之結合額外知識的技術強化一支基於蒙地卡羅樹搜尋（Monte-Carlo Tree Search）的暗棋程式。結合於此暗棋程式中的額外知識是人為設定的，並且被併入以下四個技術：提早模擬結束（Early Playout Terminations）、隱性最小極大更新（Implicit Minimax Backups）、基於品質的獎勵值（Quality-Based Rewards）、以及先進偏差（Progressive Bias）。進一步結合這四個技術的程式對原程式達到 84.75%（±1.90%）的勝率。

此外，本論文透過 2×4 暗棋研究三種強度分析度量（Strength Analysis Metric），分別為對其他玩家的勝率、對專家走步（Expert Action）的預測率（Prediction Rate）、以及對盤面價值（Value of Position）的均方差（Mean Squared Error）。實驗結果顯示，

勝率的確是衡量程式強度一種好的指標。另外兩種度量雖然不如勝率，但也是衡量程式強度不錯的指標。另一個在 2×4 暗棋上進行研究的是阿法零演算法（AlphaZero Algorithm）。阿法零演算法是一種強化學習（Reinforcement Learning）演算法，已經在西洋棋、日本將棋、以及圍棋上達到超越人類的水準。實驗結果顯示，即使是隨機型遊戲，此演算法也能夠習得盤面的理論值（Theoretical Value）以及最佳下法（Optimal Play）。

最後，此論文也研究另外兩個隨機型完全資訊遊戲：愛因斯坦棋（EinStein Würfelt Nicht!）以及 2048 類型遊戲。另一種強化學習演算法時序差異學習（Temporal Difference Learning）被應用到這兩個遊戲。在愛因斯坦棋中，結合下列三種使用習得知識之技術的程式對原程式有 62.25%（±2.12%）的勝率。此三種技術分別為先進偏差、先備知識（Prior Knowledge）、以及 ε−貪心模擬（Epsilon-Greedy Playouts）。對於 2048 類型遊戲，一個多階段（Multistage）的時序差異學習變種則用來增強學習到的知識。

On Strength Analyses of Computer Programs for Stochastic Games with Perfect Information

Student：Chu-Hsuan Hsueh　　Advisor：Dr. I-Chen Wu

Institute of Computer Science and Engineering
National Chiao Tung University

Abstract

The field of computer games is important to the researches in artificial intelligence. According to two different roles of the elements of chance involved, games can be classified as deterministic vs. stochastic and perfect information vs. imperfect information. Since many real-world problems involve uncertainty, stochastic games and imperfect information games are worthy to study. This thesis targets at stochastic games with perfect information since the games in this category is easier to model than imperfect information games. Chinese dark chess (CDC) and a reduced and solved variant, 2×4 CDC, are two games of this category which this thesis mainly focuses on.

This thesis first enhances a game-playing program for CDC based on Monte-Carlo tree search (MCTS) by several existing techniques that combine additional knowledge. The additional knowledge is manually designed, and is incorporated into four techniques including early playout terminations, implicit minimax backups, quality-based rewards, and progressive bias. By combining all, the win rate is 84.75% (±1.90%) against the original program.

In addition, this thesis investigates three strength analysis metrics on 2×4 CDC, including win rates playing against other players, prediction rates to expert actions, and mean squared errors to values of positions. Experiments show that win rates are indeed good indicators of programs' strengths. The other two metrics are also good indicators, though not as good as win rates. Another analysis performed on 2×4 CDC is applying the AlphaZero algorithm, which is a kind of reinforcement learning algorithm achieved superhuman levels of plays in chess, shogi, and Go. Experiments show that the algorithm can learn the theoretical values and optimal plays even in stochastic games.

Finally, this thesis studies two more stochastic games with perfect information, which are EinStein Würfelt Nicht! (EWN) and 2048-like games. Another kind of reinforcement learning algorithm, temporal difference learning, is applied to EWN and 2048-like games. For EWN, a program combining three techniques using the learned knowledge, including progressive bias, prior knowledge, and epsilon-greedy playouts, has a win rate of 62.25% (±2.12%) against the original program. For 2048-like games, a multistage variant of temporal difference learning improves the learned knowledge.

致謝

經過多年的研究及努力，終於完成本篇博士論文。首先要感謝指導教授吳毅成老師從大學專題以來的指導與照顧，在學術研究方面可以說是獲益良多，除了大學專題獲得系內專題競賽優等外，進實驗室後也已經發表了六篇期刊論文和六篇國際會議論文，特別開心也特別感謝的是投稿到 TAAI 2018 會議的論文獲得國際組最佳論文獎的殊榮。學術研究以外，老師也常常以自己的經驗分享很多人生道理，讓我學會用不同的角度思考事情；此外，也很感謝老師替我畢業後的出路設想很多，安排我參加棋類以外的研究題目，拓展我的研究視野，還有介紹一個日本教職的機會，讓我在畢業前就找到一個不錯的工作。

感謝陳志昌教授在暗棋研究上的指導與建議，還有提供我對未來規劃的一些分析與看法；感謝徐讚昇教授及其研究團隊提供 2×4 暗棋的資料庫做為我的研究題目之一，也感謝徐老師在 2×4 暗棋的研究上提供許多寶貴的意見。

感謝口試委員朱正忠教授、吳毅成教授、林順喜教授、陳志昌教授、徐讚昇教授、許舜欽教授、黃俊龍教授、蔡孟宗教授以及顏士淨教授（以上依姓名筆劃排列）提出論文與報告的不足之處及改善方法，讓我的論文與表達都更進步。

感謝待了將近八年半的交通大學，大學四年加上研究所四年半（碩一結束逕博、博班三年半）的時間過得很充實，也認識了很多朋友。感謝資工系在實作和理論上紮實的訓練；感謝大學專題的隊友詹翊和瑞琳一起研究殺手數獨，算是我正式開始研究電腦對局的起點；感謝實驗室的包子學長擔任我們專題的助教，在我大四下進實驗室後也很照顧我；感謝傑哥學長提供他開發的暗棋程式作為我進實驗室後的第一個研究題目，基於這支程式的延伸除了發表兩篇期刊論文和一篇國際會議論文外，也獲得國內外暗棋比賽多面獎牌；感謝 Ting 學長抽空幫幾篇我要投稿的論文潤稿，從學長的修改及建議中，

我的英文寫作技巧也有所提升；感謝我在實驗室期間的歷屆實驗室系統組阿水學長、家銓、宏君、承倫、九州、源灝管理以及維護實驗室機器以及設備，讓我可以順利完成論文所需的實驗；感謝助理 Cindy 以及我在實驗室期間的歷屆實驗室總務立楷學長、江翰、小莊、筱茜、安仁、詠嘉協助安排參與會議和比賽以及張羅實驗室的各種資源；感謝一起發表過論文的實驗室成員（按論文發表先後排列）包子學長、傑哥學長、小康學姊、阿駶、家銓、朝欽學長、江翰、詠嘉、源灝、維元、筑苓；感謝在我進實驗室的這五年期間的所有成員，從大家的研究題目以及 seminar 報告中我也學到很多。最後，也要感謝浩然圖書館豐富的藏書，除了提供專業知識的相關補充資料外，勵志、溫馨的小說也總能適時為我的心靈充滿正面能量。

感謝大學同學詹翊、昀臻、老魏、詩凡、瑞琳、文馨、乾乾、默默、亞格、秋樺、毛瑩，每次一起吃飯、聊天都很開心，特別是在大家都畢業各奔東西之後，能夠聚在一起的每個機會都很難得。

最後，最重要的是感謝家人爸爸、媽媽、哥哥、弟弟對於我攻讀博士班的全力支持與鼓勵，也感謝所有長輩、親戚的關心與支持。讀博士班的期間並不是一帆風順，也曾經長達好幾個月的時間因為研究上的挫折而沮喪，但正是因為有家人、朋友、老師的開導與鼓勵讓我能夠獲得更多繼續堅持下去的動力到現在即將完成博士學位。謹以此論文獻給我最摯愛的家人與所有的親朋好友。

薛筑軒

2019 年 3 月 5 日

Contents

List of Figures

List of Tables

Chapter 1 Introduction

The field of computer games is important to the researches in artificial intelligence. Computer games are relatively simple compared to many real-world problems. Moreover, it is easier to define *success* in computer games (i.e., whether the program can win). A famous quote by Schaeffer and van den Herik [114] was “chess is to AI as the fruit fly is to genetics”. An earlier milestone in 1997 was that the computer program Deep Blue won against the chess champion Garry Kasparov by 3½-2½ in a six-game match [20]. Nearly twenty years later, significant milestones have been reached in a more sophisticated game, Go. The program AlphaGo won against a 9-dan professional player Lee Sedol by 4-1 in 2016 [121]. A much stronger version called AlphaGo Master won against the professional player ranked number one at present, Ke Jie, by 3-0 in 2017 [123].

Games can be classified according to the numbers of players as *single-player* games (or puzzles), *two-player* games, and *multi-player* games. According to two different roles of the elements of chance involved, games can also be classified as *deterministic* vs. *stochastic* and *perfect/full information* vs. *imperfect/partial information* [99, 114, 118, 150]. Games with and without chance from outcome uncertainty (e.g., rolling dice or dealing cards) are classified as stochastic and deterministic respectively. Games with and without chance from state uncertainty (e.g., invisible opponent positions or actions) are classified as perfect and imperfect information respectively. Examples of the two kinds of classification are listed in Table 1 .

	Perfect Information	Imperfect Information
Deterministic	Checkers, Chess, Chinese Chess, Connect6, Go, Hex, Shogi	Bridge, Kriegspiel, Phantom Go, Stratego
Stochastic	Backgammon, Chinese dark chess, EinStein Würfelt Nicht!, 2048	Mahjong, Poker, Rummikub, Scrabble

Table 1: Examples of games classified according to the elements of chance involved.

Many of the researches on computer games, including the milestones, focused on deterministic games with perfect information [6, 8, 16, 20, 26-28, 37, 42, 45-50, 52, 58, 63, 67, 74-76, 82, 83, 85-87, 96, 97, 102, 104, 109, 112, 121-123, 135, 136, 138, 142, 151, 152, 156]. However, since many real-world problems involve uncertainty, stochastic games and imperfect information games are worthy to study [114]. This thesis targets at stochastic games with perfect information. In this category, the uncertainty only comes from obtainable probability distributions. For example, in backgammon, the uncertainty comes from rolling a fair six-sided dice, where the probability of rolling a specific number is 1/6. Thus, it is easier to model the games in this category than imperfect information games, which may need to consider opponent modeling [35, 38, 105, 106, 114].

Research topics for computer games include solving games [22, 24, 47, 52, 72, 102, 112, 137, 145], creating strong game-playing programs [6, 8, 9, 13, 14, 16, 20, 26-28, 34, 38, 42, 48-50, 56-58, 63, 67, 68, 74-76, 82-88, 97, 98, 100, 104, 109, 113, 120-123, 128, 131, 134-136, 138, 139, 142, 147-149, 151-153, 155, 156, 159, 162, 164], and serving for education or entertainment purpose [7, 39, 59, 62, 64, 65, 90, 101, 116, 119, 154, 158, 165]. This thesis focuses on the second topic, creating strong game-playing programs. However, defining the *strengths* of game-playing programs is already a non-trivial task. Usually, researchers make

their programs play against some other players and consider the win rates as indicators of the strengths [6-9, 12, 13, 16, 17, 19-21, 27-30, 34, 35, 37, 38, 42, 45, 48-50, 56-58, 63, 67, 70, 71, 74-76, 82, 87, 96, 97, 100, 104, 109, 120-123, 126, 128, 134-136, 138, 142, 146-149, 151-153, 156, 162]. To clarify whether the commonly used metrics such as win rates are good indicators of the playing strengths, this thesis also investigates three strength analysis metrics.

More specifically, this thesis mainly focuses on the game of Chinese dark chess (CDC) [30, 55-57, 162] and a reduced and solved variant, 2×4 CDC [22, 24, 25, 54], which are stochastic games with perfect information. CDC is a two-player game widely played in Taiwan, and also a game played in Computer Olympiad [53, 61, 94, 141, 144, 160, 161]. Although 2×4 CDC was not proposed for humans to play, it is an excellent test domain for analyses since the theoretical values of all the positions have been derived [22, 24]. This thesis first enhances an award winning game-playing program for CDC, and then investigates three strength analysis metrics and a reinforcement learning algorithm called AlphaZero [122] on 2×4 CDC. Two more stochastic games with perfect information are also studied, which are EinStein Würfelt Nicht! (EWN) [1, 34, 84, 100, 128] and 2048-like games [36, 68, 88, 98, 125, 131, 155, 159]. EWN is a two-player game played in Computer Olympiad [2, 60, 61, 94, 143, 144], while 2048-like games are a kind of single-player tile-sliding games. Another kind of reinforcement algorithm called temporal difference learning [129, 130] is applied to these two games to create strong game-playing programs.

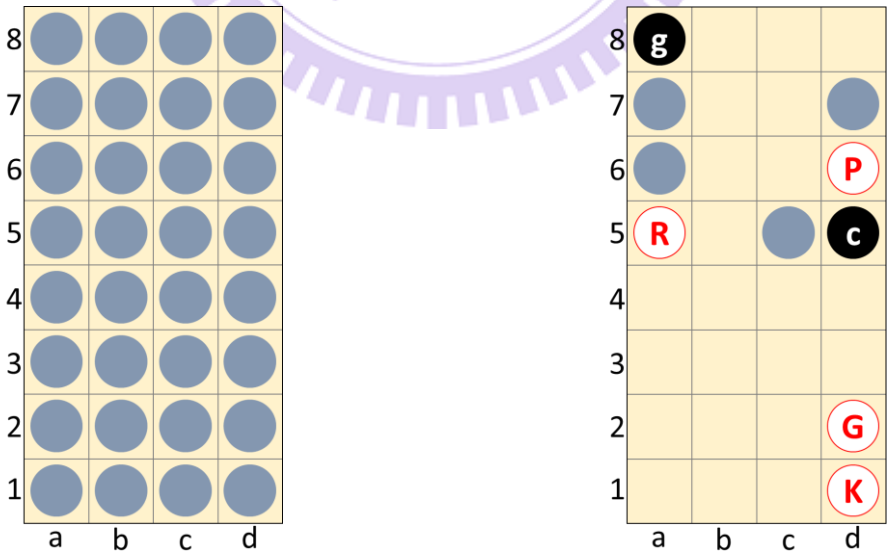
The rest of this chapter is organized as follows. Section 1.1 introduces the main test domain in this thesis, the games of CDC and 2×4 CDC, which are investigated in Chapter 2 to Chapter 4 (the other two games, EWN and 2048-like games, are introduces in Chapter 5). Section 1.2 then illustrates the motivations and goals of this thesis. Finally, Section 1.3 describes the organization of this thesis.

1.1 Chinese Dark Chess and a Reduced Variant

This section introduces the game of Chinese dark chess in Subsection 1.1.1. A reduced and solved version is then introduced in Subsection 1.1.2.

1.1.1 Chinese dark chess (CDC)

Chinese dark chess (CDC) [30, 55-57, 162], widely played in Taiwan, is a two-player zero-sum stochastic game with perfect information played on a 4×8 square board as illustrated in Figure 1 (a) and (b). It is also a game played in Computer Olympiad since 2010 [53, 61, 94, 141, 144, 160, 161]. The two players, *Red* and *Black*, respectively own identical sets of sixteen pieces with different colors. The set of pieces includes one king (K/k), two guards (G/g), two ministers (M/m), two rooks (R/r), two knights (N/n), two cannons (C/c), and five pawns (P/p), as listed in Table 2. Each piece has two faces, one showing the piece type and the other showing the piece cover which is identical for all pieces. When the piece type faces up, the type is *revealed* and known by both players. When it faces down, it is *unrevealed* and unknown.



(a) (b)

Figure 1: (a) Initial board and (b) an example of board for CDC.

Piece Types			#Pieces	Rank
Name	**Red**	**Black**		
King	K	k	1	1
Guard	G	g	2	2
Minister	N	m	2	3
Rook	R	r	2	4
Knight	N	n	2	5
Cannon	C	c	2	6
Pawn	P	p	5	7

Table 2: Pieces in CDC.

Two kinds of actions are allowed in CDC: *flipping* and *moving*. Each flipping action flips a piece, namely making the piece type revealed, and is denoted by S(?) where S is the flipped square. For example, flipping the unrevealed piece at c5 in Figure 1 (b) is denoted by c5(?). Each moving action is to move a revealed piece by the player owning the piece, denoted by S-D where S and D are source and destination squares respectively. All pieces can be moved to empty neighboring squares (with one square up, down, left or right). Pieces except cannons can be moved to neighboring squares with capturing the opponent pieces that have equal or lower ranks shown in Table 2 with the following exceptions: Pawns can capture the opponent king, but not the other way around. A piece that can be captured by another piece p as above is said to be capturable by p in this thesis. Cannons have a different capturing rule called one-step-jump [30, 162] in which any opponent pieces in the same row/column can be captured by jumping over exactly one piece in that row/column. For example, in Figure 1 (b), the black cannon (c) at d5 can capture the red rook (R) at a5 or the red king (K) at d1.

Initially, all 32 pieces are unrevealed and placed randomly, as shown in Figure 1 (a). Thus, the probability distribution of unrevealed pieces is all equal. The first player flips one of the 32 pieces and then owns the set of pieces of the revealed color, while the second player owns the other set. A game is played by the two players in turn. As flipping actions performed, the number

of unrevealed pieces decreases. When there remains at most one unrevealed piece type, the game becomes deterministic.

Finally, a player wins by capturing all of the opponent pieces or making the opponent have no legal actions. The game also ends with a draw when both players play without any capturing and flipping within 40 plies, or when the same position appears three times.

1.1.2 2×4 Chinese Dark Chess (2×4 CDC)

A reduced version of CDC, 2×4 CDC, has been devised and solved by Chang *et al*. [22, 24]. The major difference between 2×4 CDC and the original version lies in the board size and the set of used pieces. In addition to the board size, the used pieces in 2×4 CDC only considered 24 fair and equivalent material combinations (combinations of pieces) as listed in Table 3. By fair, both players have the same set of pieces at the beginning of a game. For example, "KGGM vs. kggr" is not listed in Table 3 since two players have different sets of pieces. By equivalent [32], material combinations with the same relations between pieces are only counted once. For example, "KGGR vs. kggr" is not listed in Table 3 since it has the same piece relations as "KGGM vs. kggm."

KGGM vs. kggm	KGGC vs. kggc	KGGP vs. kggp	KGMM vs. kgmm
KGMR vs. kgmr	KGMC vs. kgmc	KGMP vs. kgmp	KGCC vs. kgcc
KGCP vs. kgcp	KCCP vs. kccp	KCPP vs. kcpp	KPPP vs. kppp
GGMM vs. ggmm	GGMR vs. ggmr	GGMC vs. ggmc	GGCC vs. ggcc
GGCP vs. ggcp	GMCP vs. gmcp	GCCP vs. gccp	GCPP vs. gcpp
GPPP vs. gppp	CCPP vs. ccpp	CPPP vs. cppp	PPPP vs. pppp

Table 3: 24 fair and equivalent material combinations in 2×4 CDC.

All legal positions were assigned theoretical values from retrograde analysis [137] in the expectiminimax manner [22, 24]. A position was defined by the set of remaining pieces, the

configuration of the board, and the player to act. Chang *et al*. [22, 24] used expected win rates as theoretical values. If a position does not contain any unrevealed piece or only one unrevealed piece type remains, the game becomes deterministic. Thus, the theoretical value is 1, 0, or -1 which represents a win, a draw, or a loss respectively from the view point of the player to act. An example position is shown in Figure 2 (a), where the result is a win for the black player.

(a) (b)

Figure 2: Examples of positions in 2×4 CDC from a game of "KGCP vs. kgcp" (a) without and (b) with chance events.

Since CDC is actually a stochastic game – a flipping action with different piece types may result in different game outcomes – for each position, the theoretical value ranges from -1 to 1 where -1 and 1 were 100% losses and 100% wins respectively from the view point of the player to act. For example, assuming that it is the red player's turn in the position in Figure 2 (b), the theoretical value is 0.0958 according to Chang *et al*. [22, 24], which shows a slight advantage toward the red player. Note that a theoretical value of 0 does not represent a necessary draw, it may also represent the situation with 50% wins and 50% losses.

1.2 Motivations and Goals

This thesis aims to create strong programs for stochastic games with perfect information, and to investigate strength analysis metrics for game-playing programs. More specifically, this thesis first applies several existing techniques to Monte-Carlo tree search (MCTS) [26, 73] to further enhance the strengths of the game-playing programs for CDC. MCTS is a best-first

search algorithm which is able to work without domain-specific knowledge. However, the search process may require an infeasible amount of time to converge to better (or even the best) actions. Usually, combining additional knowledge, which can be manually designed or learned (semi)-automatically, can lead MCTS to converge faster. The combined knowledge for CDC is manually designed. The investigated techniques include early playout terminations [8, 45, 76, 83, 85, 86, 151, 152], implicit minimax backups [76], quality-based rewards [104], and progressive bias [26, 28].

As mentioned earlier, it is non-trivial to define the strengths of game-playing programs. A commonly used metric is the win rates playing against other players [6-8, 12, 13, 16, 17, 19-21, 27-30, 34, 35, 37, 38, 42, 45, 48-50, 56-58, 63, 67, 70, 71, 74-76, 82, 87, 96, 97, 100, 104, 109, 120-123, 126, 128, 134-136, 138, 142, 146-149, 151-153, 156, 162]. Sometimes, prediction rates to expert actions [25, 37, 46, 75, 82, 109, 121, 123, 138, 146, 156] and mean squared errors to values of positions [48, 121, 123, 156] are also used to measure the strengths of game-playing programs. This thesis aims to clarify how well these strength analysis metrics are, which is investigated on 2×4 CDC.

This thesis also applies reinforcement learning algorithms to learn domain-specific knowledge for games. The AlphaZero algorithm [122], which was proposed by Silver *et al*. and achieved superhuman levels of plays in chess, shogi, and Go, is applied to 2×4 CDC for analyses. The work aims to investigate whether the AlphaZero algorithm can learn the theoretical values of positions and optimal plays even in stochastic games. Another kind of reinforcement learning algorithm, temporal difference learning [129, 130], is applied to EWN and 2048-like games. The learnt knowledge for EWN is further incorporated into three techniques to improve the playing strength, which include progressive bias [26, 28], prior knowledge [48, 49], and epsilon-greedy playouts [126]. For 2048-like games, a multistage variant of temporal difference learning is presented for further enhancement.

1.3 Organization

The rest of this thesis is organized as follows. Chapter 2, mainly based on [56, 57], enhances and analyzes the strength improvements of an MCTS-based CDC program. Chapter 3, mainly based on [55], then investigates three strength analysis metrics on 2×4 CDC. Chapter 4, mainly based on [54], studies the AlphaZero algorithm through 2×4 CDC. Chapter 5, mainly based on [34, 159], contains the investigation on two more stochastic games, EinStein Würfelt Nicht! and 2048-like games. Finally, Chapter 6 concludes this thesis and discusses future research directions. In addition, Appendix A presents the rules to determine early playout terminations for CDC in MCTS for Subsection 2.4.1. Appendix B includes the detailed experiment results of the investigated strength analysis metrics in Chapter 3. Appendix C includes the detailed experiment results of the AlphaZero algorithm on 2×4 CDC in Chapter 4.

Chapter 2 Strength Improvements for CDC Programs

Monte-Carlo tree search (MCTS) [18, 73] has been successfully applied to many games, such as Go [26-28, 37, 42, 48-50, 58, 63], General Game Playing (GGP) [16, 45, 104], backgammon [146] and phantom Go [17, 148, 149]. Many techniques were proposed to further improve the playing strength of MCTS-based programs. In this chapter, four techniques are studied and analyzed through an MCTS-based CDC game-playing program named DARKKNIGHT. The program has won two CDC tournaments in 2013 [140, 160], including the 17th Computer Olympiad.

1. *Early playout terminations* [8, 45, 76, 83, 85, 86, 151, 152]: Terminate a playout much earlier when it is very likely to win, lose, or draw.
2. *Implicit minimax backups* [76]: Guide MCTS selections by using minimax scores from heuristic evaluations of positions together with the original simulated win rates.
3. *Quality-based rewards* [104]: Use simulation length and terminal state quality to adjust the rewards returning from simulations.
4. *Progressive bias* [26-28, 58, 63, 96]: Use heuristic scores of actions to differentiate newly generated nodes in selections.

This chapter is organized as follows. Sections 2.1 and 2.2 review MCTS and previous work on computer Chinese dark chess respectively. Section 2.3 introduces the above four techniques, and the incorporation into DARKKNIGHT is described in Section 2.4. The experiment results are shown in Section 2.5. Some fine tunings for tournaments and the results are included in Section 2.6. Finally, Section 2.7 makes concluding remarks of this chapter.

2.1 Monte-Carlo Tree Search

Monte-Carlo tree search (MCTS) [18, 73] is a best-first search algorithm on top of a search tree, named UCT [73], using Monte-Carlo simulations as state evaluations. To generate an action, MCTS usually consists of several iterations (or called simulations). Each iteration contains four phases:

1. Selection. A path is traversed from the root to one of the leaves following a selection policy. A popular selection policy is to select the children with the maximum values of the upper confidence bounds (UCB) function [4]:

$$\underset{a}{\operatorname{argmax}}\left\{Q(s,a)+C\cdot\sqrt{\frac{\ln(\sum_b N(s,b))}{N(s,a)}}\right\} \tag{1}$$

   where $Q(s,a)$ is an estimator for the value of action a for state s, $N(s,a)$ the visit count, and C a constant representing the weight of exploration. Commonly, $Q(s,a)$ is a win rate, $W(s,a)/N(s,a)$, or a mean value for rewards, where $W(s,a)$ is the win count of a for s.
2. Expansion. One or more children are expanded from the selected leaf.
3. Playout. Following a playout policy (e.g., randomly select a legal action), the playout is played from the position corresponding to the selected leaf to some terminal state of the game.
4. Backpropagation. The result of the playout, e.g., rewards, is updated from terminal state back to all its ancestors in the selected path during selection.

After the search finishes, the action a with the highest $N(s_0,a)$ at the root s_0 is chosen as the best action to play.

2.2 Computer CDC

B.-N. Chen *et al.* [30] estimated the state-space complexity of the game CDC to be 10^{37} between those of Draughts and chess, while the game tree complexity without considering *chance nodes* [110] was estimated to be 10^{135} between those of chess and Chinese chess. Yen *et al.* [162] estimated the game tree complexity with chance nodes to be 10^{207} between those of Chinese chess and Shogi.

Many of the CDC programs were developed using *alpha-beta search* [23, 30, 31, 161]. B.-N. Chen *et al.* [30] published a work about alpha-beta search in their CDC program. Some more research work for CDC included opening books [29], endgame databases [23, 31-33, 111], solving smaller CDC [22, 24, 25] and game variations [69, 79].

MCTS has also been incorporated into CDC game-playing programs [70, 71, 160, 162]. Yen *et al.* [162] developed an MCTS-based CDC program, named DIABLO, which used n*on-deterministic nodes*, so-called in their article, to contain all *inner nodes* for all the possible outcomes of the flipping actions. The rest of this thesis uses chance nodes instead of non-deterministic nodes since they are conceptually similar. Thus, inner nodes are simply children of chance nodes. The win counts and visit counts of chance nodes were the sums from all their children. In the selection phase, roulette wheel selection was applied to chance nodes to select one of children according to the probability distribution of unrevealed pieces.

In the playout phase, the program DIABLO [162] used piece weights, as listed in Table 4, when capturing pieces. Another MCTS-based CDC program, named DARKKNIGHT [160], was developed independently, but it used almost the same weights as Yen *et al.* [162], except for reducing them by a factor of 100. DIABLO won four tournaments in 2011 and 2012 [81, 127, 157, 163], while DARKKNIGHT won two CDC tournaments in 2013 [140, 160], including the 17th Computer Olympiad. DARKKNIGHT serves as the baseline program in this chapter.

Piece type	**K/k**	**G/g**	**M/m**	**R/r**	**N/n**	**C/c**	**P/p**
Weights by Yen *et al.* [162]	5500	5000	2500	1000	800	3000	800
Weights in This Thesis	55	50	25	10	8	30	8

Table 4: Pieces weights for CDC.

2.3 Incorporated Techniques

In this section, the four techniques, including early playout terminations, implicit minimax backups, quality-based rewards, and progressive bias, are reviewed in Subsections 2.3.1 to 2.3.4 respectively.

2.3.1 Early Playout Terminations

The idea of *early playout terminations*, i.e., to terminate playouts early before the ends of games, was proposed by several researchers with different forms and names. *Early Cutoffs* was proposed by Finnsson [45] to collect information online, and terminate playouts when the game obviously favors some player from the collected information. It was used to save the time for more simulations for GGP.

Some other researches were to return results after making a fixed number of actions in playouts. For Amazons [83, 85], Breakthrough [85, 86], and Havannah [85], Lorentz and Horey used evaluation functions to determine the winners who the evaluated scores favor. Baier and Winands [8] performed a shallow minimax search with fixed depths and then returned the evaluated scores directly. Their method, called *MCTS with Informed Cutoffs*, was tested for Othello, Breakthrough and Catch the Lion.

Winands *et al*. [151, 152] proposed a method for Lines of Action (LOA) to check the scores by evaluation function every three actions, and then return the results in the following cases. If the scores exceeded some threshold, the results were wins. On the contrary, if the scores were

below some threshold, the results were losses. Checking evaluation functions every three actions was for the sake of performance.

2.3.2 Implicit Minimax Backups

Lanctot *et al.* [76] proposed *implicit minimax backups* which incorporated into MCTS the minimax scores based on *heuristic evaluations* of positions. The minimax scores were used together with the estimator $Q(s,a)$ to guide the selections of MCTS. Namely, $Q(s,a)$ in Formula (1) was replaced by the following:

$$(1 - w_{IMB}) \cdot Q(s,a) + w_{IMB} \cdot M(s,a) \quad (2)$$

where $M(s,a)$ is the minimax score of action a for state s and w_{IMB} the weight of the minimax score.

In each leaf of UCT, a heuristic evaluation function was used to calculate the score of the position corresponding to the leaf, scaled to $[-1,1]$ through a sigmoid function [51]. The scaled score was then backed up as classical minimax search. For consistency, $Q(s,a)$ was also ranged in $[-1,1]$.

They improved the playing strength of the three games, Kalah, Breakthrough, and LOA, and obtained win rates of around 60% to 70% in different settings of the self-play games. Their work also showed that even simple evaluation functions can lead to better performance. However, they found that this technique did not work on every game. They ran experiments on Chinese checkers and the card game Heart but obtained no significant improvement.

2.3.3 Quality-Based Rewards

Pepels *et al.* [104] proposed *quality-based rewards* to adjust the rewards obtained from simulations. Simulation length and terminal state quality were used as *quality assessments* of simulations to adjust the rewards from wins and losses.

Simulation length, a domain independent quality assessment, was defined as the length of simulated game from the root to the terminal state in the simulation. Intuitively, in an advantageous position, the shorter the simulation length is, the better.

This technique maintained online sample mean and sample standard deviation of simulation length, denoted by μ_L^p and σ_L^p, for player p when p wins. With these statistical values, the reward R_L for simulation length was calculated as follows:

$$R_L = R + sign(R) \times a_L \times b(\lambda_L) \tag{3}$$

where R is the reward (e.g., 1 for a win and -1 for a loss), a_L the influence for this quality, $b(\lambda)$ a sigmoid function as shown in Formula (4) which scales the values to $[-1, 1]$, λ_L a normalized value in Formula (5), k a constant to be tuned and l is the length of that simulation.

$$b(\lambda) = -1 + \frac{2}{1 + e^{-k\lambda}} \tag{4}$$

$$\lambda_L = \frac{\mu_L^p - l}{\sigma_L^p} \tag{5}$$

For terminal state quality, a function describing quality of terminal states was needed. For example, for Breakthrough, the piece difference between the winning and losing players was used to measure the terminal state quality, which was scaled to $[0, 1]$. Let μ_T^p and σ_T^p respectively denote sample mean and sample standard deviation of terminal state quality for player p. Similarly, the rewards R_T for terminal state quality was calculated in a way similar to Formula (3) as follows:

$$R_T = R + sign(R) \times a_T \times b(\lambda_T) \tag{6}$$

where λ_T is a normalized value in Formula (7), and t is the terminal state quality.

$$\lambda_T = \frac{t - \mu_T^p}{\sigma_T^p} \tag{7}$$

In Formula (3) and (6), a_L and a_T are constants or is calculated according to the data accumulated online. Experiment results by Pepels *et al.* [104] showed no much difference.

2.3.4 Progressive Bias

Without any knowledge, MCTS still converges to the best action in a sufficient number of simulations eventually [73]. However, with some more knowledge, MCTS can use it to differentiate the newly generated nodes and then guide the search to converge more efficiently. *Progressive bias* [26-28, 58, 63, 96, 97] is one of such techniques, proposed to guide selections based on given knowledge. The researchers incorporated knowledge by adding additional heuristics for the games of Go, Othello, Focus, and Chinese Checkers. Chaslot *et al*. [26, 28] proposed to extend UCB function (Formula (1)) to the following:

$$\underset{a}{\operatorname{argmax}}\left\{Q(s,a)+C\cdot\sqrt{\frac{\ln(\sum_b N(s,b))}{N(s,a)}}+c_{PB}\cdot\frac{H(s,a)}{N(s,a)}\right\} \tag{8}$$

where $H(s,a)$ is the heuristic score for action a at state s and c_{PB} a coefficient representing the influence of the heuristic. Actions with higher $H(s,a)$ will be selected earlier. The heuristic score $H(s,a)$ in this subsection is action evaluation that affects action selection, while the heuristic evaluation in Subsection 2.3.2 for implicit minimax backups is position evaluation that affects the estimated values of search trees.

The extended term (the third) is the heuristics used to bias the search to favor selecting actions with higher $H(s,a)$. It gets smaller as $N(s,a)$ grows. That is, the heuristics weigh less for larger $N(s,a)$. This implies to trust more for the win rates obtained by MCTS when the actions are visited more times. The experiment results by Chaslot [26] showed increased win rates of 25.1%, 21.5%, and 4.8% respectively in 9×9, 13×13, and 19×19 Go. This technique, with different forms from the above formula, was also successfully applied to Go [27, 58, 63] and Othello [96]. Rosin [108] proposed *Predictor+UCB* (*PUCB*) which extended the UCB function to introduce biases over actions. The regret bound for PUCB was also studied. A variant of PUCB was applied to the AlphaGo program by Silver *et al*. [121-123]. Other than

the work mentioned above, Gelly and Silver [48, 49] proposed to initialize win rates and visit counts by some default values based on prior knowledge.

2.4 Incorporation into CDC

In this section, the incorporation of the four techniques (introduced in Section 2.3) into DARKKNIGHT is described in Subsections 2.4.1 to 2.4.4 respectively.

2.4.1 Early Playout Terminations

For CDC and many other games, it is important to terminate playouts earlier for two reasons, speedup and/or accuracy of simulated results. The first reason is obvious. The second is that it is more accurate to terminate playouts at the time when they are very likely to win, lose, or draw. For example, it is normally a draw in a situation, `GGk`, where Red has two guards and Black has only one king. In the case of keeping playing in the playout, Red may lose accidentally, while it is supposed to be a draw since `k` is hard to capture both `G`'s. Thus, an early termination is actually more accurate than a continuous playout. Another example is `KGmm`. No matter where the pieces are placed and whether the pieces are revealed or not, Red always wins the game, since both `K` and `G` can capture `m`, but not *vice versa*. However, it may result in draws in playouts since Red may not be able to go to capture black pieces tactically during playouts.

In this thesis, playouts are terminated earlier when detecting a *likely outcome* which is *win*, *loss*, or *draw*. Without loss of generality, likely outcomes are from Red's perspective. The detection rules are based on a material combination [32, 111], namely a set of remaining pieces on the board in CDC. For CDC, the total number of legal material combinations is 8,503,055 ($= (2^1 3^5 6^1)^2 - 1$). All the combinations are analyzed according to some heuristic rules, described in more details in Appendix A. From the analysis, 750,174 are set to wins, 750,174 losses,

108,136 draws, and the rest are unknown. For example, the likely outcomes for `KGmm`, `KGk` and `KGggm` are set to win, and those for `GGk`, `CCccp` and `GGMkp` are draw. Note that it is called a likely outcome since the outcome may not be always true in a few extreme cases as illustrated by the following example. For `KGggm`, Black can try to exchange one `g` with `G`, though it is hard to do so. If Black successfully exchanges `g` with `G` without losing `m`, then the game becomes a draw.

In playouts, if the likely outcome for a material combination is one of win, loss, and draw, the playouts end immediately and return the outcomes. The overhead for the detection is little, since this can be simply checked by a lookup table. Thus, the detection is done for every action in playouts.

A final remark is when root positions have likely wins, losses, or draws. In this situation, almost all playouts terminate immediately with the same results, and thus MCTS cannot distinguish better actions. To address this issue, a special design based on the property of CDC is as follows. For root positions with likely draws, the playouts continue until at least one piece has been captured. The results of draws may change if some pieces are captured. The design encourages MCTS to pay attention to the change of material combinations. For root positions with likely wins and losses, the playouts continue until at least one loser's piece has been captured. This encourages MCTS to favor shorter wins and longer losses.

2.4.2 Implicit Minimax Backups

For CDC, the heuristic evaluation function used in this thesis is the weighted material sum, namely the difference of the total piece weights between the player and the opponent. The piece weights are listed in Table 4. The heuristic evaluations are then scaled to $[-1, 1]$ by a sigmoid function.

One issue to discuss is the minimax scores for chance nodes, which were not mentioned

by Lanctot *et al.* [76]. An intuitive way is to use the probability distribution for unrevealed pieces when calculating the expected values for chance nodes. For example, suppose to have four unrevealed pieces, two `P`'s, one `k`, and one `m` (as in Figure 1 (b)). The probability for `P` is 1/2, while those for the other two are 1/4. However, a problem occurs when unrevealed pieces are not in the UCT yet. For example, if flipping `k` has not been expanded yet, the corresponding heuristic evaluation is unknown, making it hard to get the minimax scores for chance nodes.

In order to solve this problem, the ratios of visit counts are used as the probabilities for all children of chance nodes. For the above example, assume to flip `P` three times and `m` once, but none for `k`. The weights of the minimax scores for flipping `P` and `m` are 3/4 and 1/4 respectively.

Another issue is the quality of the heuristic evaluations including the resulting minimax scores. Although simple evaluation functions were shown leading to better performance, the scores may not reflect the values of positions well, compared to simulated win rates especially when the number of simulations is sufficiently large. To deal with this problem, a variant is proposed in this thesis to dynamically decrease w_{IMB} in Formula (2) as numbers of visits increase:

$$w_{IMB}(s,a) = (1 - c_c) \cdot \sqrt{\frac{c_s}{c_s + c_N \cdot \sum_b N(s,b)}} + c_c \qquad (9)$$

where c_c, c_s, and c_N are all coefficients which decide where and how fast w_{IMB} converges, making the adjustment of w_{IMB} more flexible.

2.4.3 Quality-Based Rewards

The two quality assessments proposed by Pepels *et al.* [104], simulation length and terminal state quality, are incorporated into DARKKNIGHT to adjust the rewards from wins and losses. For draws, the simulations are not sampled. The two quality assessments are measured when simulations end. Without early playout terminations, MCTS simulations end when one

player wins. With early playout terminations, MCTS simulations also end when one obtains a likely outcome with win or loss.

At terminal states, simulation length is simply the same as the one described by Pepels *et al.* [104], and terminal state quality is obtained in the following way. First, simply count the remaining pieces of the winner. Then, incorporate domain knowledge like piece weights. Namely, the following formula is used to calculated the terminal state quality in CDC:

$$\sum_{i \in S} (1 + c_w \times \omega_i) = |S| + c_w \sum_{i \in S} \omega_i \tag{10}$$

where S is the set of remaining pieces of the winner, $|S|$ the size of S (i.e., the number of remaining pieces of the winner), c_w a coefficient, and ω_i the weight of piece i as in Table 4. The larger the coefficient c_w is, the higher the influence of the piece weights. All values of terminal state quality are scaled to $[-1, 1]$ according to a sigmoid function. Note that the remaining pieces of the loser, if any, is not considered.

2.4.4 Progressive Bias

This thesis proposes some heuristics for obtaining $H(s, a)$ for CDC in DARKKNIGHT and then incorporates the heuristics into progressive bias by modifying Formula (8) as follows:

$$\underset{a}{\operatorname{argmax}} \left\{ Q(s, a) + C \cdot \sqrt{\frac{\ln(\sum_b N(s, b))}{N(s, a)}} + c_{PB} \cdot \frac{H(s, a)}{Count(s, a)} \right\} \tag{11}$$

More specifically, $N(s, a)$ in the third term of Formula (8) is changed to a general $Count(s, a)$. In practice, $Count(s, a)$ can be set to many functions, such as $N(s, a)$, $\sqrt{N(s, a)}$, $\ln(N(s, a) + 1)$, $L(s, a)$, $\sqrt{L(s, a)}$, and $\ln(L(s, a) + 1)$, where $L(s, a)$ is the loss count of action a for state s, which is the summation of the number of losses and half the number of draws for CDC in this thesis. The formula by Ikeda and Viennot [63] and Nguyen *et al.* [96] included the heuristics similar to the one with square root. The formula by Chaslot *et*

al. [27] contained the form of logarithm. Most of the work [26-28, 58, 63, 96] considered $N(s,a)$, while Nijssen and Winands [97] suggested to use $L(s,a)$ instead.

The rest of this subsection describes three evaluation methods for heuristic scores $H(s,a)$ based on CDC domain-specific knowledge. The first evaluation based on basic heuristics, collectively referred to $\mathbf{H^B}$ in the rest of this thesis, calculates $H(s,a)$ as follows. For a capture move that captures a piece p, add ω_p to $H(s,a)$, where ω_p is the weight of p as listed in Table 4. For an escape move that avoid a piece p from being captured, add ω_p to $H(s,a)$. Finally, for a suicide move that has a safe piece p be moved to an unsafe square (to be captured by the opponent), subtract ω_p from $H(s,a)$.

The second evaluation calculates $H(s,a)$ as above, but adds one more heuristic for curb moves, referred to as $\mathbf{H^C}$. A *curb move* is to move a piece p to a diagonal square of an opponent piece q which is capturable by p, such that the opponent piece has less mobility. As shown in Figure 3, moving `G` left curbs `r` such that `r` cannot move right and down (otherwise, `r` will be captured). This feature often occurs in CDC games and is important especially in endgames where the winning players usually use continuous curb moves to force a win. For a curb move, add $c_{curb} \times \omega_q$ to $H(s,a)$, where c_{curb} is a coefficient normally less than one since q has not been really captured yet. In the case that a move curbs two (or more) pieces at the same time, just consider the one with the highest weight. Note that if a curb move is also a suicide move, it is viewed as a suicide move only without extra bonus from curb.

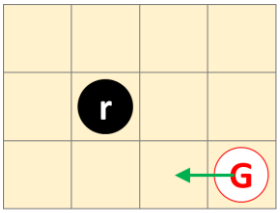


Figure 3: An example of a curb move.

The third evaluation calculates $H(s,a)$ as the second, but add one more heuristic for

flipping, referred to as $\mathbf{H^F}$. For flipping actions, the following four cases are discussed. In the first two cases, the flipped piece belongs to the player to act. In the first case, the flipped piece p can be captured immediately by some opponent piece. In this case, simply subtract $P(p) \times \omega_p$ from $H(s,a)$, where $P(p)$ is the probability of flipping p. For example, for a flipped piece, say M, if one opponent k or g is neighboring to it, then M will be captured immediately. Hence, $P(M) \times \omega_M$ is subtracted from $H(s,a)$. Note that the actual safety of p (i.e., whether p is protected by some other pieces) is disregarded since it is hard to have an efficient and accurate method to identify. In the second case, the flipped piece p can capture some opponent piece q in the next move. For example, M for p and r for q. Similarly, the actual safety of q (i.e., whether q can escape) is disregarded, since it is also hard to have an efficient and accurate method to calculate the escape. In this case, add $c_{flip} \times P(p) \times \omega_q$ to $H(s,a)$, where c_{flip} is a coefficient normally much less than one due to the uncertainty of the escape of q.

In the next two cases, the flipped piece belongs to the opponent. In the third case, the flipped piece p can capture some piece q of the player immediately. In this case, simply subtract $P(p) \times \omega_q$ from $H(s,a)$, similar to the first case. For example, for a flipped piece, say m, if R is neighboring to it, then m can capture R immediately. In the fourth case, the flipped piece p can be captured by some piece q of the player. For example, m for p and G for q. Similarly, the actual safety of p (i.e., whether p can escape) is disregarded. In this case, add $c_{flip} \times P(p) \times \omega_p$ to $H(s,a)$, where c_{flip} is used for the same reason as that in the second case. For the above, if there are more than one captured pieces, simply choose the one with the highest weight with a reason similar to that for curb moves.

2.5 Experiments

As mentioned in Subsection 1.1.1, the number of chance nodes decreases as flipping

actions performed. Figure 4 shows some statistical data for CDC games in average, including the numbers of legal actions, chance nodes, and revealed pieces on a board. The statistical data were collected from one million self-play games by the original DARKKNIGHT with 10,000 simulations per action. From Figure 4, all pieces are revealed around the 100th plies, and the most number of revealed pieces on a board is around 14 at the 35th ply.

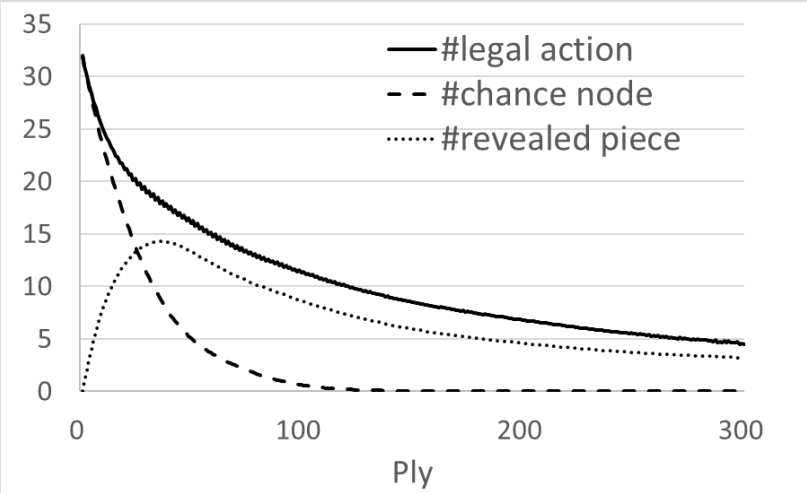


Figure 4: Statistical data for CDC.

In the experiments, each modified version played 1,000 games against the baseline, the original DARKKNIGHT, unless explicitly specified. Among the 1,000 games, one played 500 games as the first and the other 500 as the second. For each game, one scored 1 point for a win, 0 for a loss and 0.5 for a draw. For a given version, the average score of 1,000 games was the win rate against the baseline. The 95% confidence intervals are also shown for all win rates.

Initially, a strength test was performed for the baseline with different numbers of simulations per action against the one with 10,000, as shown in Figure 5. In the rest of experiments, unless explicitly specified, the one with 30,000 was chosen, which was reasonable in the sense of both strength and computation time. Namely, it had a win rate of 75.75% (±2.39%), while it took about 3.2 minutes to run a game with one thread on machines equipped with Intel (R) Core(TM) i5-3570K CPU, 3.40GHz.

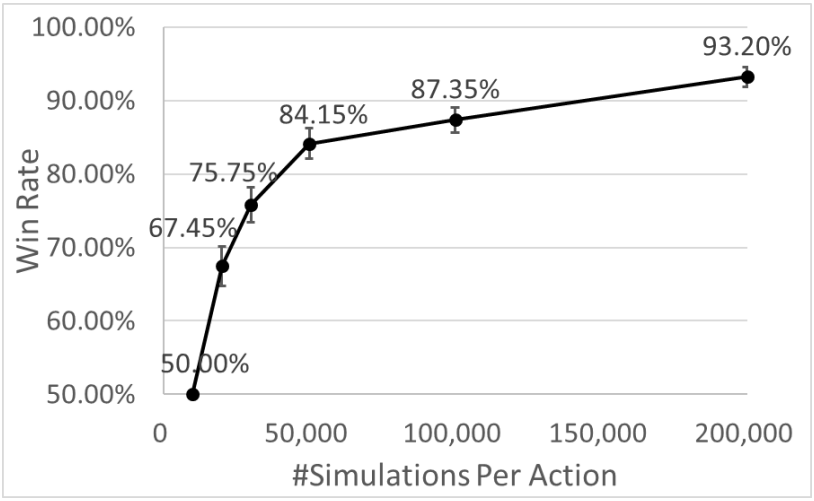


Figure 5: Baseline with different numbers of simulations per action against that with 10,000.

In the rest of this section, Subsection 2.5.1 incorporates the four techniques individually into DARKKNIGHT, and analyzes the strength improvements. Subsection 2.5.2 combines techniques to further improve the playing strength.

2.5.1 Individual Techniques

In this subsection, the experiment results for the four techniques, including early playout terminations, implicit minimax backups, quality-based rewards, and progressive bias, are shown in Subsections 2.5.1.1 to 2.5.1.4 respectively.

2.5.1.1 Early Playout Terminations

As mentioned for early playout termination (EPT) in Subsection 2.4.1, material combinations were used to detect whether a playout reaches a terminal state earlier with a likely outcome, one of win, loss, or draw. The experiment result showed a significant improvement with a win rate of 60.75% (±2.76%) against the baseline program. The result indicated that the accuracy of the simulated results was indeed increased with the help of the likely outcomes returned from EPT. In addition, the program with EPT also ran faster than the one without. The

number of simulations per second from the initial position was increased from 27,356 to 34,730. The speedup came from the reduction of nearly one third of the playout length. The program with EPT averagely played 101.7 actions in a playout (from the initial position) while the one without EPT played 148.4 actions. It is expected that the improvement is even greater during middle and end games.

An experiment was performed to show how the simulation accuracy influences playing strength. In the experiment, instead of returning likely outcomes, wrong outcomes were returned with a probability of ε_{EPT}. More specifically, in the experiment, likely wins return losses with a probability of ε_{EPT}, and *vice versa*. The results in Figure 6 shows that the playing strength got worse as ε_{EPT} increased and went down nearly to nearly 0% when ε_{EPT} was 40%.

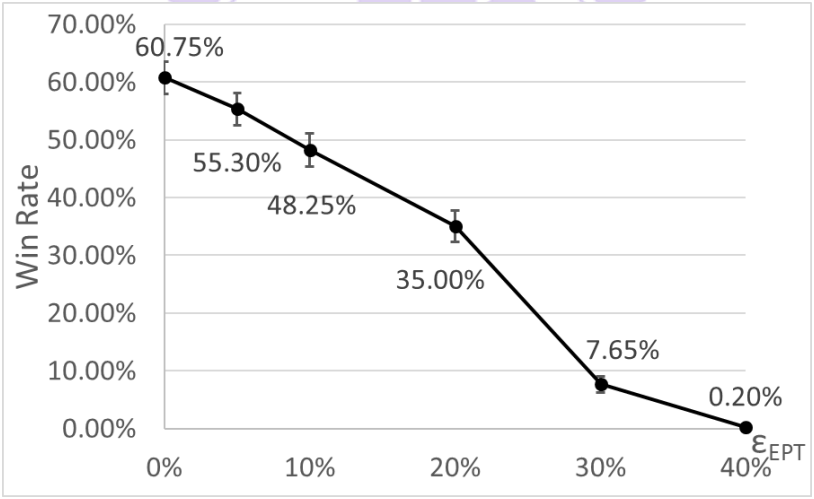


Figure 6: Win rates for EPT with different ε_{EPT}.

2.5.1.2 Implicit Minimax Backups

As mentioned for implicit minimax backups (IMB) in Subsection 2.4.2, minimax scores from heuristic evaluations were used to help guide the selections of MCTS more accurately. In the experiments, different weights of minimax score, w_{IMB} (fixed as constant), and different numbers of simulations per action were tested first. The experiment results are shown in Figure

7 which includes three lines, respectively for 10,000, 30,000, and 100,000 simulations per action. For fairness, the corresponding baselines also ran the same numbers of simulations.

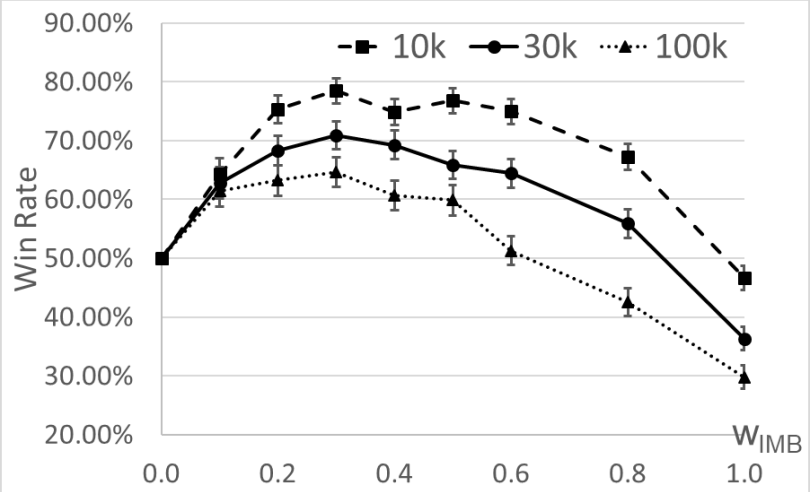


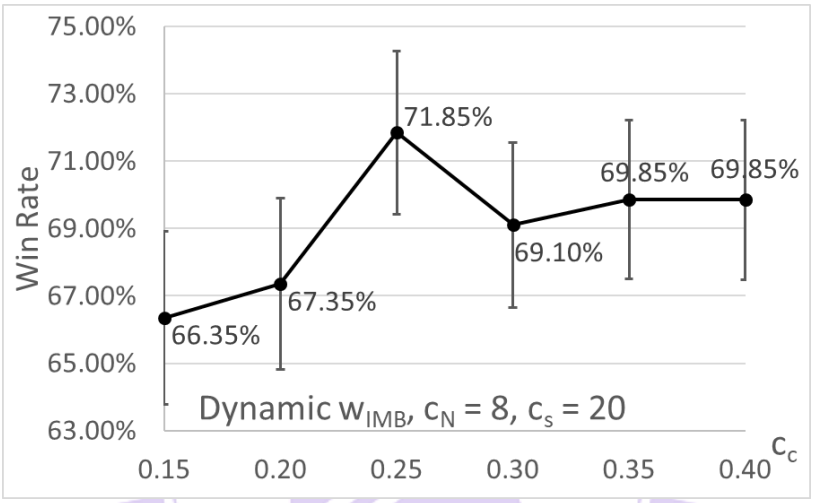
Figure 7: Win rates for IMB with different w_{IMB} and different numbers of simulations per action.

Figure 7 shows that the win rates are the highest when $w_{IMB} = 0.3$ for the three lines, and 78.45% (±2.18%) for the one with 10,000 simulations per action, 70.90% (±2.42%) for 30,000, and 64.60% (±2.51%) for 100,000. The figure shows that IMB did significantly improve the playing strength. On one hand, in the case that w_{IMB} was too high, the win rates went down for the following reason. The heuristic evaluations weighted too much higher than online estimation which is usually more accurate than heuristic evaluations for a sufficiently large number of simulations. On the other hand, in the case that w_{IMB} was too low, the win rates also went down for the following reason. Less heuristic information was used to help guide the selections of MCTS accurately, since short-term tactical information provided by the minimax scores was missing, as explained by Lanctot *et al.* [76].

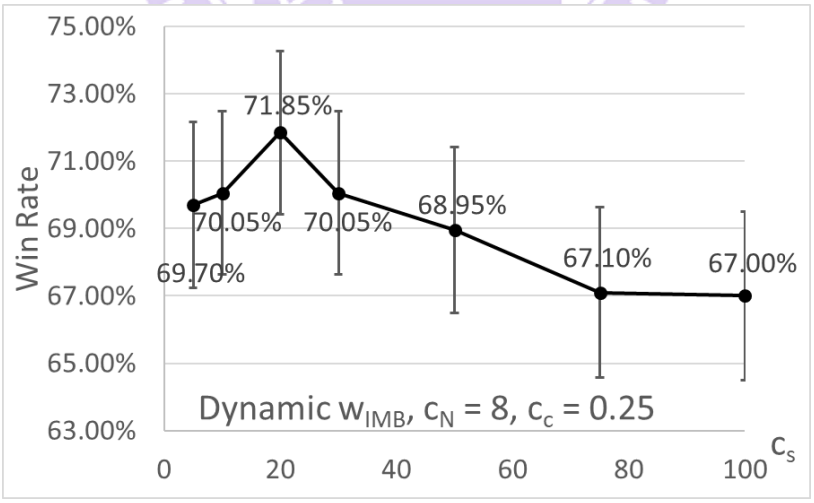
Figure 7 also has the following implication. For a higher number of simulations per action, the improvement was relatively smaller. This hints that IMB has less improvement or becomes worse for a sufficiently large number of simulations per action. The reason is: the help of

minimax scores decreases, since simulated results become more accurate with more simulations.

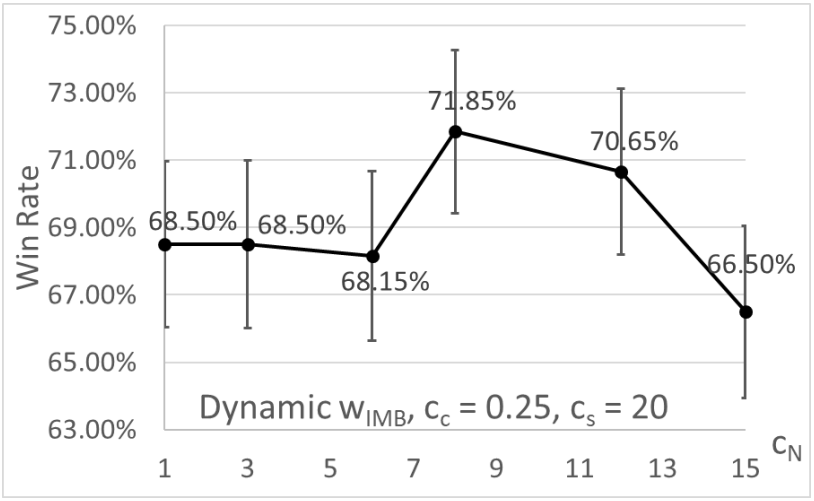
Based on the results in Figure 7, some values for c_c, c_s, and c_N were selected to test IMB with dynamic w_{IMB} in Formula (9). In the experiments, one coefficient was varied at a time, as shown in Figure 8. When $c_c = 0.25$, $c_s = 20$, and $c_N = 8$, the highest win rate of 71.85% (±2.42%) was obtained, slightly better than fixing $w_{IMB} = 0.3$ though it was not significant.



(a)



(b)



(c)

Figure 8: Win rates for IMB with dynamic w_{IMB} for different (a) c_c, (b) c_s, and (c) c_N.

Lanctot *et al.* [76] mentioned that IMB had no significant improvement for Heart, a stochastic game with imperfect information. Interestingly, the results in this thesis showed that IMB did significantly improve the playing strength for CDC.

2.5.1.3 Quality-Based Rewards

As mentioned in Subsection 2.4.3, simulation length (SL) and terminal state quality (TSQ) were used to adjust the rewards from simulations to favor those with shorter length and higher terminal state quality.

For SL, two coefficients were tested, which were the influence a_L (Formula (3)) and the sigmoid function constant k_L (Formula (4)). The results are shown in Figure 9, where the data with the same k_L are staggered to show the confidence intervals. The highest win rate was 57.50% (±2.77%) when $a_L = 0.25$ and $k_L = 7$.

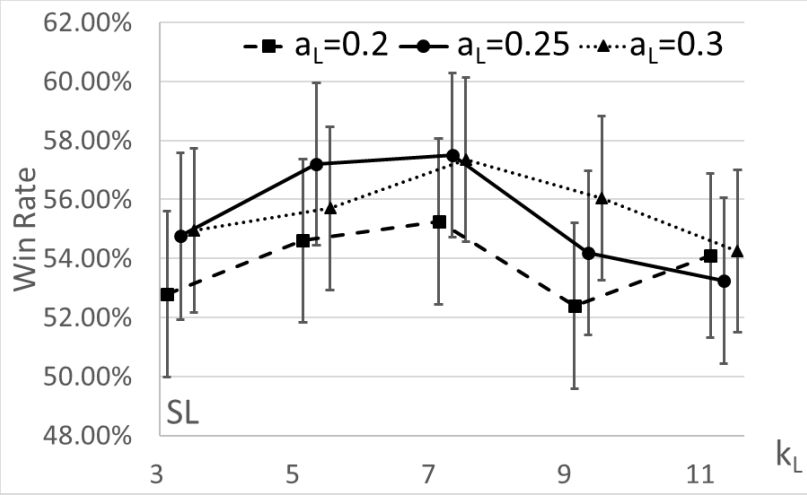
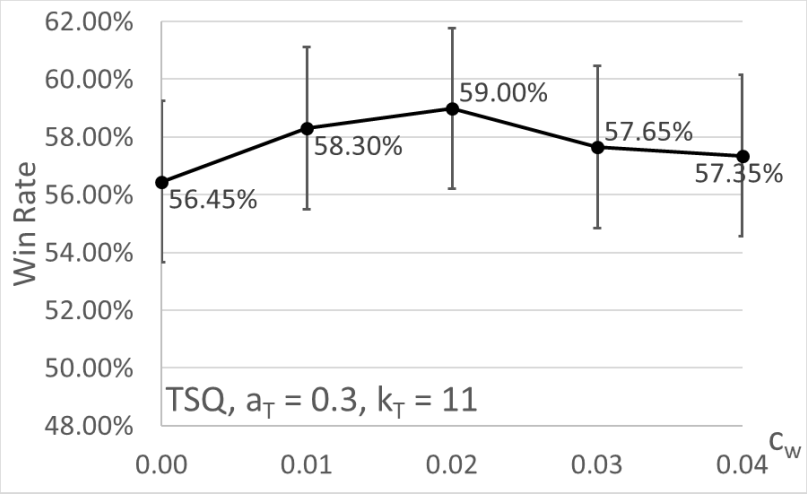
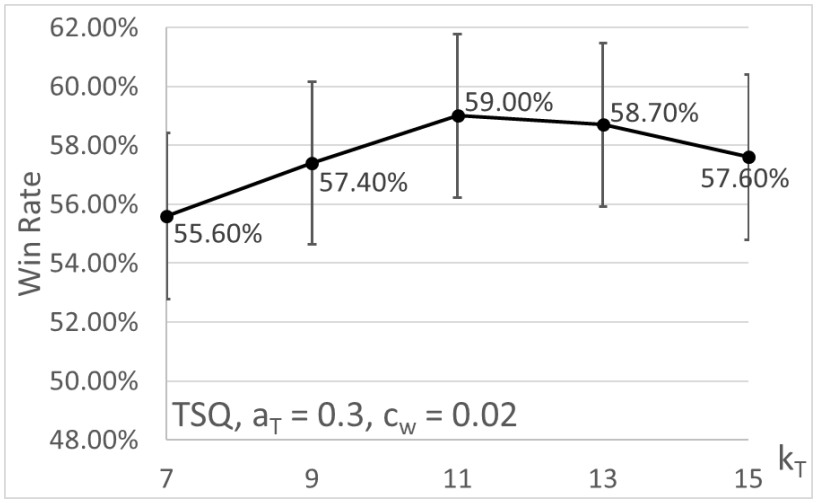


Figure 9: Win rates for SL with different a_L and k_L.

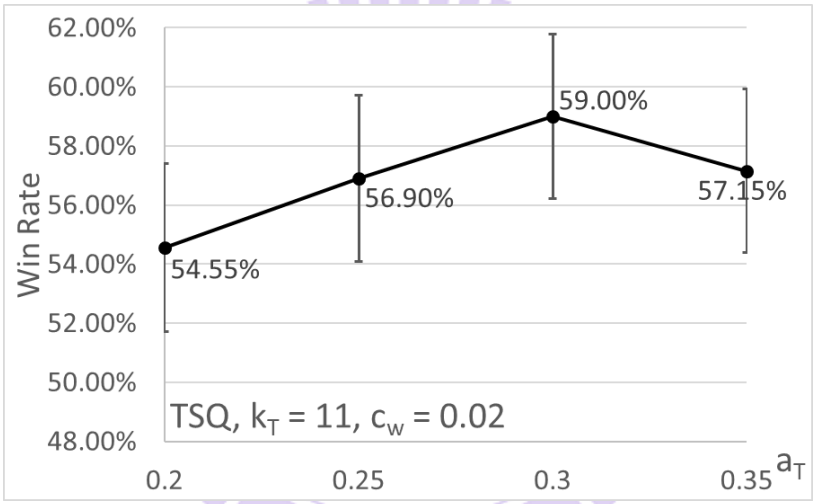
For TSQ, one more coefficient was needed, which was c_w (Formula (10)) for piece weights when measuring terminal state quality. In the experiments, the highest win rate was 59.00% (±2.78%), when $a_T = 0.3$, $k_T = 11$ and $c_w = 0.02$. By fixing $a_T = 0.3$ and $k_T = 11$, the highest win rate was obtained at $c_w = 0.02$, slightly better than the one at $c_w = 0.01$, as in Figure 10 (a). Similarly, by fixing a_T and c_w, the highest win rate was obtained at $k_T = 11$ as in Figure 10 (b). By fixing k_T and c_w, the highest win rate was obtained at $a_T = 0.3$ as shown in Figure 10 (c).



(a)



(b)



(c)

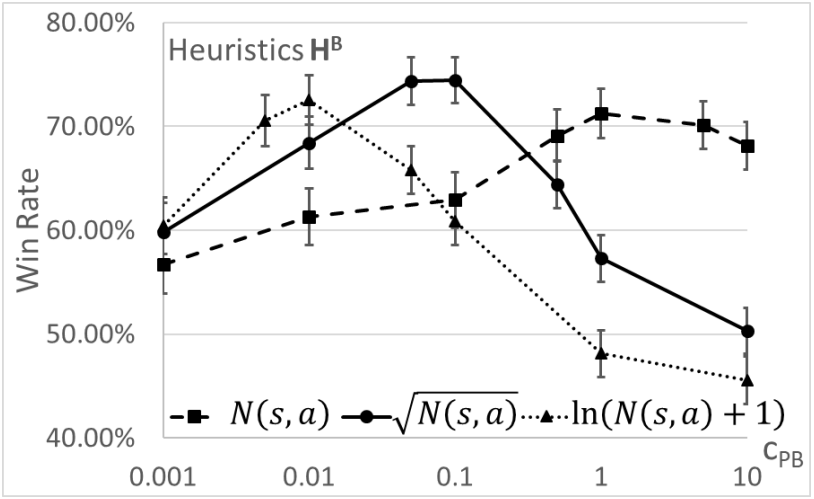
Figure 10: Win rates for TSQ with different (a) c_w, (b) k_T, and (c) a_T.

From above, quality-based rewards (SL and TSQ) improved the playing strength generally. However, the performances did not differ too much for different settings, as relatively high confidence intervals were observed.

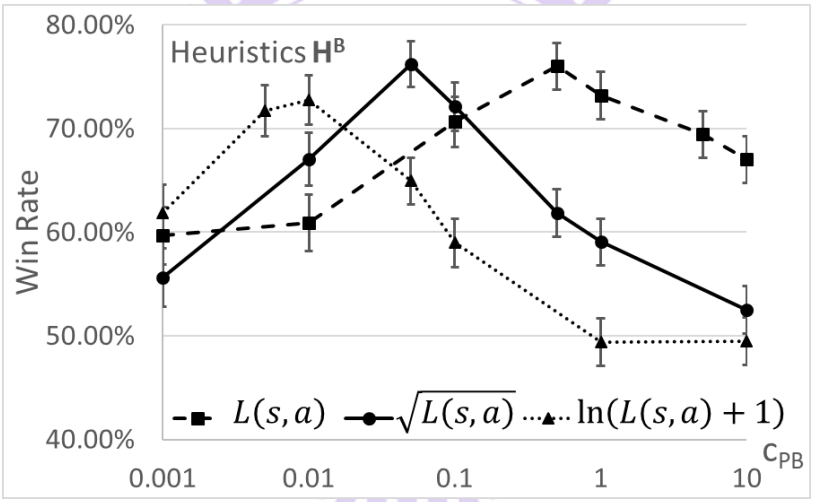
2.5.1.4 Progressive Bias

As mentioned in Subsection 2.4.4 for progressive bias (PB), heuristics on actions were used to help guide selections for newly generated nodes. In the experiments, the basic heuristics

$\mathbf{H^B}$ was used first for $\mathrm{H}(s,a)$, which considers capture, escape, and suicide moves.



(a)



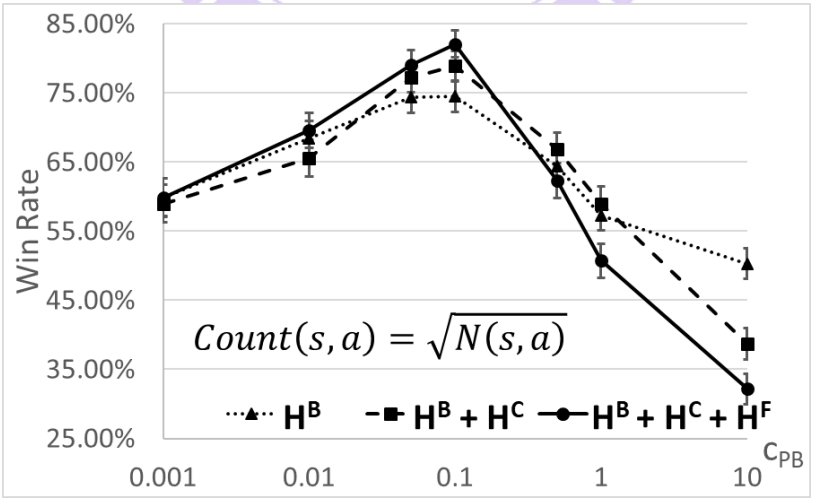
(b)

Figure 11: Win rates for PB using heuristics $\mathbf{H^B}$ with different forms of $Count(s, a)$ based on (a) $N(s, a)$ and (b) $L(s, a)$.

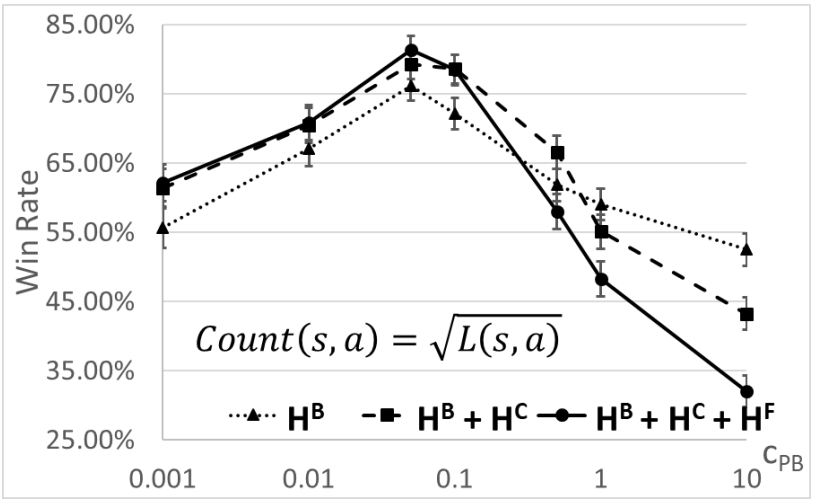
The experiments tested different $Count(s,a)$ and c_{PB} (Formula (11)). In Figure 11 (a), $N(s,a)$, $\sqrt{N(s,a)}$, and $\ln(N(s,a)+1)$ were used for $Count(s,a)$, while in Figure 11 (b), $L(s,a)$, $\sqrt{L(s,a)}$, and $\ln(L(s,a)+1)$ were. In both figures, c_{PB} varied from 0.001 to 10. From the two figures, all kinds of $Count(s,a)$ showed win rates higher than 71% with proper settings of c_{PB}. In Figure 11 (a), the highest win rate was 74.45% (±2.18%) with

$Count(s,a) = \sqrt{N(s,a)}$ and $c_{PB} = 0.1$. In Figure 11 (b), the highest win rate was 76.20% (±2.22%) with $Count(s,a) = \sqrt{L(s,a)}$ and $c_{PB} = 0.05$. The above figures show that PB did significantly improve the playing strength, and that $Count(s,a)$ with square root, $\sqrt{N(s,a)}$ and $\sqrt{L(s,a)}$, performed better than the other two. Thus, the rest of the experiments considered the two cases only, though the one with $L(s,a)$ was very close to $\sqrt{L(s,a)}$.

The experiments then tested more heuristics, namely the second evaluation with additionally considering curb moves, denoted by $\mathbf{H^B} + \mathbf{H^C}$, and the third evaluation with curb moves and flipping, denoted by $\mathbf{H^B} + \mathbf{H^C} + \mathbf{H^F}$. These evaluations included three coefficients c_{PB}, c_{curb} and c_{flip}. First, both coefficients $c_{curb} = 1/4$ and $c_{flip} = 1/12$ were fixed. In the case of $\sqrt{N(s,a)}$ (for $Count(s,a)$), Figure 12 (a) shows that $\mathbf{H^B} + \mathbf{H^C} + \mathbf{H^F}$ performed best with the highest win rate of 82.10% (±1.98%), when $c_{PB} = 0.1$. In the case of $\sqrt{L(s,a)}$, Figure 12 (b) shows that $\mathbf{H^B} + \mathbf{H^C} + \mathbf{H^F}$ performed best with the highest win rate of 81.30% (±2.04%), when $c_{PB} = 0.05$. Thus, the above shows that the playing strength can be improved with more heuristics with proper setting for these coefficients. To sum up, among all cases, $\mathbf{H^B} + \mathbf{H^C} + \mathbf{H^F}$ performed best in the case that $Count(s,a) = \sqrt{N(s,a)}$ and $c_{PB} = 0.1$.



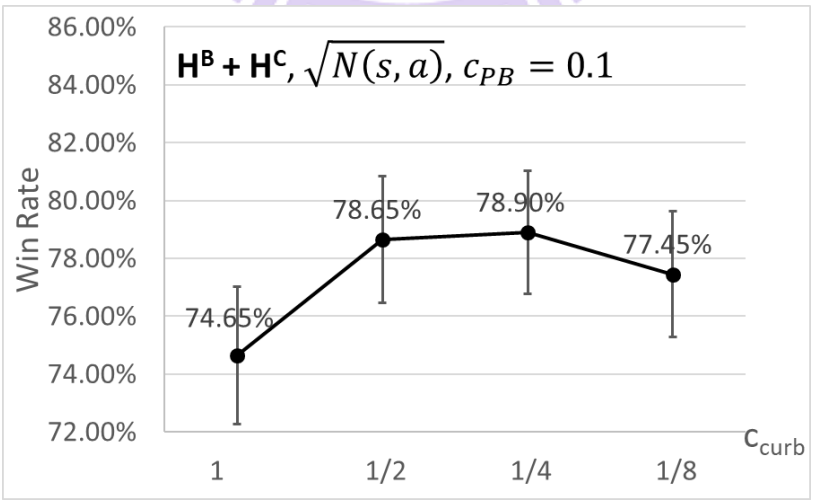
(a)



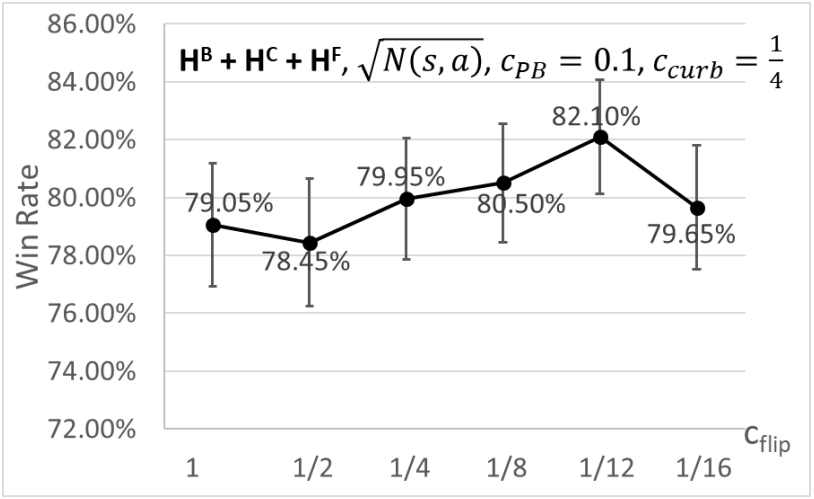
(b)

Figure 12: Win rates for PB using different heuristics with $Count(s, a)$ being (a) $\sqrt{N(s, a)}$ and (b) $\sqrt{L(s, a)}$.

In order to confirm the best values for coefficients c_{curb} and c_{flip}, the next experiments chose $\sqrt{N(s,a)}$ for $Count(s,a)$ and set $c_{PB} = 0.1$ for $\mathbf{H^B + H^C}$ and $\mathbf{H^B + H^C + H^F}$. Figure 13 (a) shows that $\mathbf{H^B + H^C}$ performed the best when $c_{curb} = 1/4$, though the one with $c_{curb} = 1/2$ had a very close win rate. Figure 13 (b) shows the experiment results for $\mathbf{H^B + H^C + H^F}$ with $c_{curb} = 1/4$. The results indicated that $\mathbf{H^B + H^C + H^F}$ performed best when $c_{flip} = 1/12$.



(a)



(b)

Figure 13: Win rates for PB (a) using $\mathbf{H^B + H^C}$ with different c_{curb} and (b) using $\mathbf{H^B + H^C + H^F}$ with different c_{flip}.

Figure 14 shows the cumulative probability of finding the target action within a given rank for the three heuristics $\mathbf{H^B}$, $\mathbf{H^B + H^C}$, and $\mathbf{H^B + H^C + H^F}$ respectively. A *target action* was the action selected by the best version of DARKKNIGHT (see Subsection 2.5.2 below) with 30,000 simulations per action. The results were collected from 1,000 games with 156,333 actions. From Figure 14, the effectiveness of using more heuristics can also be obtained from the ability to find the target action. The prediction rate for $\mathbf{H^B + H^C + H^F}$ was higher than that for $\mathbf{H^B + H^C}$, which in turn was higher than that for $\mathbf{H^B}$. From this, it is expected in CDC that the heuristics with higher cumulative probabilities of finding target actions perform better in progressive bias.

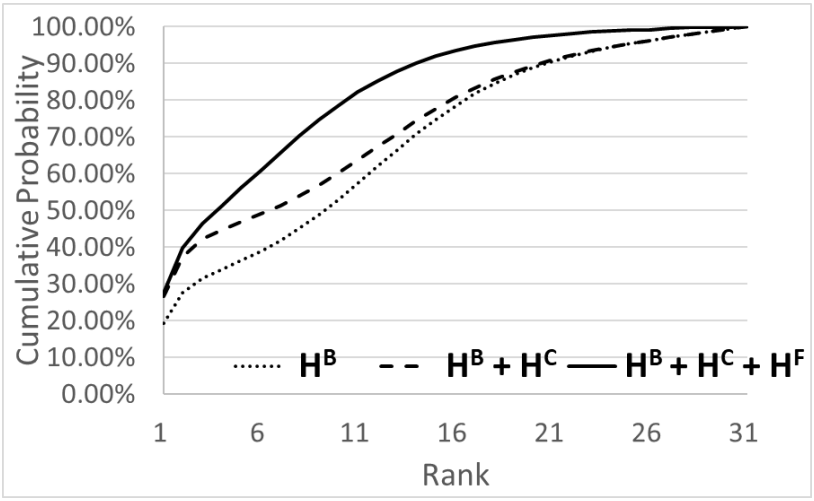


Figure 14: Cumulative probability of finding the target action within a given rank.

2.5.2 Combinations of Techniques

In this subsection, the results in the previous subsection are first summarized. The playing strength are then further improved by combining the above techniques.

	Baseline	EPT	IMB	SL	TSQ	PB
Win Rate	--	60.75% ±2.76%	70.90% ±2.42%	57.50% ±2.77%	59.00% ±2.78%	82.10% ±1.98%
Speed (#sim/sec)	27,356 ±38	34,730 ±57	27,093 ±41	27,284 ±37	27,182 ±39	25,127 ±39

Table 5: The best win rates and speeds of EPT, IMB, SL, TSQ, and PB.

Table 5 summarizes the best results for incorporating each individual techniques, EPT, IMB, SL, TSQ, and PB, where each technique used the best settings as described above, except that $w_{IMB} = 0.3$ was fixed for IMB for simplicity of analysis. These settings will be used in the rest of experiments. Table 5 also lists the speeds (from the initial position). Among all the techniques, EPT ran the fastest since playouts were terminated earlier with less overhead. IMB, SL, and TSQ updated data incrementally, thus they only ran slightly slower than the baseline.

Finally, for PB, since it calculated the expected evaluations for flipping actions, it ran the slowest, but improved the strength most.

The first experiment combined every two techniques, and the results for all the combinations are listed in Table 6. Generally, combining two techniques improved the playing strength, except for a few cases. Besides, the results also showed a tendency as follows. If a technique, say PB, had a better win rate (in the second column) than another, say IMB, all other techniques adding PB had better win rates than those adding IMB. However, it is interesting to see that the win rate for PB decreased with combining IMB. The conjecture for this phenomenon is that the heuristics for both are correlated. For example, for a capture move, both weighted material sum in IMB and heuristic $H(s,a)$ in PB increase at the same time. Therefore, the heuristic tends to weigh higher. The effect is just like to have a high w_{IMB} in IMB or a high c_{PB} in PB. This implies that the best individual settings of these coefficients, w_{IMB} and c_{PB}, became less accurate when combined together.

	Individual	**+SL**	**+TSQ**	**+EPT**	**+IMB**
SL	57.50% ±2.77%	--	--	--	--
TSQ	59.00% ±2.78%	61.70% ±2.68%	--	--	--
EPT	60.75% ±2.76%	62.75% ±2.73%	67.30% ±2.66%	--	--
IMB	70.90% ±2.42%	72.70% ±2.39%	73.10% ±2.39%	74.75% ±2.38%	--
PB	82.10% ±1.98%	81.70% ±2.00%	82.10% ±2.01%	83.65% ±2.00%	78.30% ±2.07%

Table 6: Win rates for the combinations of EPT, IMB, SL, TSQ, and PB.

Furthermore, more techniques were combined as the three combinations listed in Table 7. For the third combination, combining all the techniques together (in the rightmost column), the

win rate reached up to 84.75% (±1.90%). The first combination was to remove PB from the third, while the second was to remove IMB. For the first combination, the win rate dropped to 76.70% (±2.31%). For the second combination, the win rate slightly dropped to 83.85% (±1.91%). This showed again that PB improved the playing strength more significantly than IMB. An additional experiment was to use the first combination as baseline, the third combination had a win rate of 64.90% (±2.55%) which showed significant improvement.

	+EPT +IMB +SL+TSQ	**+EPT +SL+TSQ +PB**	**+EPT +IMB +SL+TSQ +PB**
Win Rate	76.70% ±2.31%	83.85% ±1.91%	84.75% ±1.90%

Table 7: Win rates for combining more techniques.

2.6 Fine Tunings for Tournaments and Results

In tournaments, DARKKNIGHT used lock-free multithreaded MCTS [41] to perform more simulations. For opening positions, the speed was about 140,000 simulations per second on a machine equipped with Intel (R) Core(TM) i7-CPU, 3.20GHz, 6 threads. The thinking time per action was determined by a semi-dynamic strategy named EXP [9]. More specifically, the thinking time was the remaining time divided by the expected number of remaining actions to play (fixed at 90 in DARKKNIGHT). In addition, a dynamic strategy named STOP [9] was applied to stop the searches as soon as the best actions do not change. In more detail, this strategy measured the speeds of simulations and estimated the expected remaining numbers of simulations. As long as the sum of the expected remaining number and the second-most visit count was less than the most visit count, the search could be stopped safely.

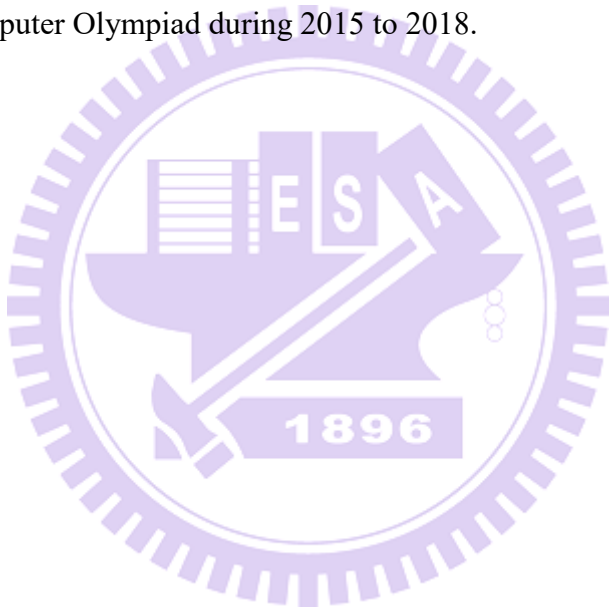
A final remark is an implementation detail on dealing with the condition of draw games. In tournaments, an endgame without capturing and flipping is judged as draws after 180 plies instead of 40 plies. The reason is that J.-C. Chen *et al.* [33] found a case where a player needs 177 plies to win in a five-man endgame. Without specialization to the 180-ply rule, when DARKKNIGHT was almost winning, it just played unimportant actions, did not capture opponent pieces, and finally ran out of time to lose. To deal with this problem, DARKKNIGHT used *internal draw-ply* to judge draws in simulations. The internal draw-ply was calculated by a step function $\min(\lfloor n_{ply}/20 \rfloor * 20 + 40,180)$, where n_{ply} is the number of plies without capturing and flipping at the root position. The design urged the program to win within at most 40 plies, but also had risks to miss the chances to win if the wins need more plies than 40.

Some versions of DARKKNIGHT based on the techniques introduced in Section 2.4 and the fine tunings in this section have attended the CDC tournaments during 2014 to 2018. These versions won the champions in the Computer Olympiad during 2015 to 2018 [53, 61, 94, 141], the TAAI Cup in 2016 and 2017 [92], and the TCGA Cup in 2015. In addition, DARKKNIGHT also won silver medal in TCGA Cup in 2014 and 2017 [80, 93], and bronze medal in TAAI Cup in 2018 [95]. The techniques, except EPT, and the fine tunings were also directly applied to 8×8 Chinese dark chess [79], and won the champion in TAAI Cup 2015 [91].

2.7 Chapter Conclusions

This chapter incorporates the four techniques, early playout terminations (EPT), implicit minimax backups (IMB), quality-based rewards (SL and TSQ), and progressive bias (PB) together and obtained significant improvement. The improvements were analyzed through an MCTS-based CDC program, DARKKNIGHT. The experiment results showed that all of these techniques did significantly improve the playing strength with win rates of 60.75% (±2.76%),

71.85% (±2.42%), 57.50% (±2.77%), 59.00% (±2.78%), and 82.10% (±1.98%) against the original DARKKNIGHT when incorporating EPT, IMB, SL, TSQ, and PB, respectively. The results also indicated that the improvement by progressive bias was the most significant. By incorporating all together, the win rate reached up to 84.75% (±1.90%). The results demonstrated the effectiveness of the above techniques for an MCTS-based program playing a stochastic game CDC. The analysis provides some hints to games like CDC or even other stochastic problems, though the amount of improvement from each technique may be different from CDC's. An enhanced version of DARKKNIGHT with some fine tunings won the CDC tournament in the Computer Olympiad during 2015 to 2018.



Chapter 3 Strength Analysis Metrics for CDC Programs

When developing game-playing programs, one important thing is to measure the strengths of the programs. Assuming that an *optimal player* exists, a program's *absolute strength* can be measured by the win rate playing against the optimal player. However, it is impractical to expect access to such optimal players in many real-world games since these games are too complex to be solved entirely [72, 145]. Thus, for the strength analysis of game-playing programs of these games in practice, many researchers used win rates as well as some other metrics such as prediction rates and mean squared errors to measure their strengths.

The first goal of this chapter is to analyze the correlation between these practically measured strengths and absolute strengths for the stochastic game of Chinese dark chess (CDC) as a case study. A strong correlation implies that the practically measured strengths can be used to represent the absolute strengths. The analysis was done on the solved 2×4 CDC [22, 24]. The second goal is to investigate whether the win rates obtained in reduced and small-sized games can be used to predict those in the full-sized games, since games with smaller sizes usually share similar properties to the original ones. A strong correlation in this case implies that new algorithms can be tested in small-sized games instead of full-sized games to save time.

The rest of this chapter is organized as follows. Section 3.1 introduces three investigated metrics: win rates against a designated player, prediction rates to expert moves, and mean squared errors to values of positions. Section 3.2 describes simple linear regression, a mathematical tool that is used for analyses. Section 3.3 then contains the experiment results and discussions on the three metrics in 2×4 CDC. Section 3.4 analyzes projecting the win rates obtained in 2×4 CDC to those in 4×8 CDC. Finally, Section 3.5 makes concluding remarks of this chapter. Detailed experiment results are included in Appendix B.

3.1 Investigated Strength Analysis Metrics

In this section, three strength analysis metrics, win rates against a designated player, prediction rates to expert moves, and mean squared errors to values of positions, are introduced in Subsections 3.1.1 to 3.1.3 respectively.

3.1.1 Win Rates Playing against a Designated Player

For game-playing programs, win rates against designated players are commonly used as a metric to measure the strengths of the programs. It is usually claimed that the higher the win rate obtained, the stronger the program is [134]. Playing against human players is also a way [19-21, 35, 42, 121, 123, 134]; however, since it is too slow to obtain statistically significant results, usually, the designated players are computer programs, often called *baseline programs*.

Baseline programs in practice can be a random player (randomly make an action) [70, 71], an early version of the tested program, or other available programs. Usually, random players are weak and are easily defeated. In the past, when demonstrating the strengths of new algorithms or heuristics, usually they were made to play against early versions of the tested programs [8, 9, 17, 26-28, 30, 42, 45, 46, 48, 56, 57, 74, 76, 96, 97, 100, 104, 109, 121-123, 128, 138, 142, 147, 149, 151-153, 156, 162]. Sometimes, they were made to play against other (openly) available programs [6, 7, 9, 12, 28, 35, 37, 38, 42, 46, 48-50, 63, 67, 74, 76, 84, 87, 96, 121-123, 126, 135, 136, 138, 148, 149, 151, 152, 162].

However, it is still a question whether and how much the win rates can reflect the absolute strengths of the tested programs. In the solved 2×4 CDC, since theoretical values of all positions are known, there exists an optimal player which always selects *best actions*, defined as the actions leading to the highest theoretical value. This optimal player can serve as the baseline program. More specifically, theoretical values of actions are determined as follows.

For a moving action, the theoretical value directly comes from the position after moving. For a flipping action, the theoretical value is the weighted average for unrevealed pieces based on the probability distribution of these unrevealed pieces. For example, the move b1(?) in Figure 15 has a theoretical value of 0.000 which is the weighted average of the theoretical values of the three possibilities G $(0.167 \times 2/4)$, k $(0.333 \times 1/4)$, and p $(-0.667 \times 1/4)$.

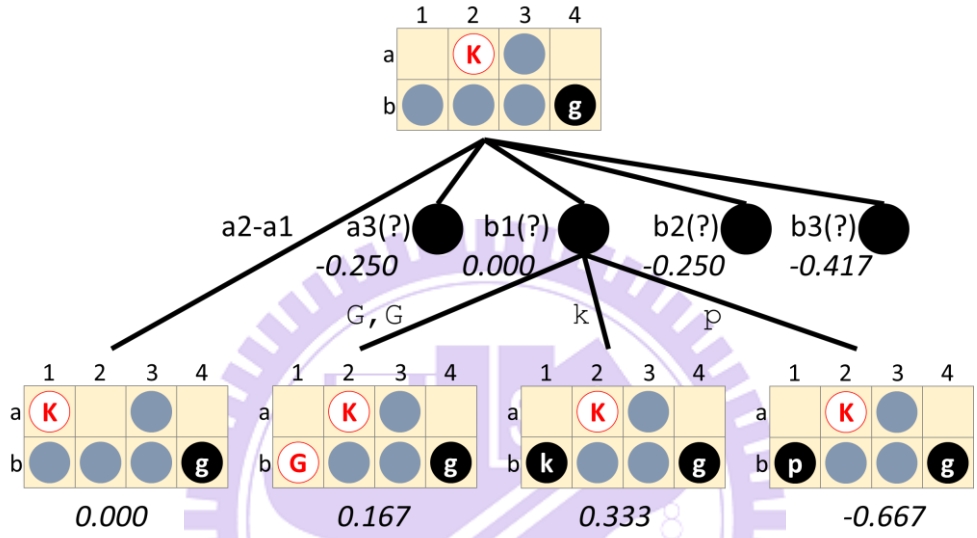


Figure 15: A partial tree with theoretical values of positions in the red player's view from a game of "KGGP vs. kggp" where one of the two g's as well as the P are captured and it is the red player's turn at the root position.

Note that, for deterministic positions with wins or losses, the theoretical values did not provide information of distances to win or loss [22, 24]. Thus, the optimal player does not take into consideration shortest wins and longest losses. In addition, the optimal player does not take into consideration whether a move is risky or not. For the example in Figure 15, the move b1(?) has a theoretical value of 0.000 from the red player's view, which is as good as the move a2-a1. Thus, the optimal player just randomly chooses from the two moves. However, the move b1(?) is riskier since the red player has a probability of 1/4 to flip the p at b1 which will lead to a highly disadvantageous situation.

3.1.2 Prediction Rates to Expert Actions

Assuming that a set of positions with actions played by experts (optimal players at best) are available, *prediction rates* to those expert actions are commonly used to analyze and measure strengths of programs [25, 37, 46, 75, 82, 109, 121, 123, 138, 146, 156]. The prediction rate of a program is the probability that the actions selected by the program match expert actions. For example, actions from game records played by human players were used as expert actions in the game of Go [37, 121, 123, 138, 156], Othello [82, 109], and backgammon [146]. In some cases, the actions generated by stronger programs also served as expert actions. For example, Lai [75] and Gao *et al.* [46] used the actions from programs with time-limited searches as expert actions in chess and Hex respectively.

In 2×4 CDC, since theoretical values of all positions are known, best actions can be used to derive the prediction rate. It is possible that a position has more than one best action. An example is shown in Figure 15 where the actions a2-a1 and b1(?) are all best actions. In this case, selecting any of these actions is said to match the best actions. As explained in Subsection 3.1.1, the theoretical values cannot distinguish shorter wins or longer losses from wins or losses. Thus, if more than one action leads to a win, all are considered as best actions. In the past, prediction rates of best actions were measured by Chang *et al.* [25] for a basic MCTS program in 2×4 CDC. However, only positions within the first two plies of a game were tested.

If a program has a high prediction rate to the expert actions, it can be said that the strength of the program is close to the experts' and thus can be inferred as a strong program. However, any actions not matching expert actions do not necessarily indicate sub-optimal plays for the following two reasons. First, the expert actions (from human players or programs) may not be the theoretically best actions. Second, it is possible that there exist several equally good actions given a position, but only the one selected by the expert is counted as an expert action. An

example that stronger programs had lower prediction rates was shown by Silver *et al.* [123] for the game of Go, where AlphaGo Zero had much higher win rates but lower prediction rates to actions by professional players (cf. Figures 3 and 4 by Silver *et al.* [123]).

3.1.3 Mean Squared Errors to Values of Positions

Assuming that a set of positions with *ground-truth values* (theoretical values under best circumstances) are available, *mean squared errors* between the given ground-truth values and the values estimated by game-playing programs can be used to analyze the strengths of the programs [48, 121, 123, 156]. The mean squared error of a program on a given set of positions P and the corresponding ground-truth values $\{v_p\}_{p \in P}$ is calculated as

$$MSE = \frac{1}{|P|} \sum_{p \in P} \left(\hat{v}_p - v_p\right)^2 \tag{12}$$

where $|P|$ is the number of positions in P and $\hat{v}_p$ the value of position p estimated by the program.

In 2×4 CDC, theoretical values of all positions are known and can be used to serve as the ground-truth values. However, since it is hard for many games to obtain their theoretical values, usually the outcomes of the games played by experts or reasonably strong programs were used instead [121, 123, 132, 156]. Using the outcomes of the games meant that v_p for all positions in a game were set to the outcome of the game. In addition, Takeuchi *et al.* [132] labeled the values of positions by the theoretical values if exhaustive search can be finished in a reasonable time.

If a program has a low mean squared error of values of positions, it means that the program understands the positions better and thus is usually stronger. Mean squared errors were shown to have a strong correlation to the strengths of programs [48, 123].

3.2 Simple Linear Regression

Simple linear regression, described in more detail by Lane *et al.* [77], is a model to derive how two random variables are linearly correlated. This section illustrates it by the example of two random variables X and Y with sample data $(x, y) \in \{(1,4), (3,6), (5,10), (5,12), (6,13)\}$, as shown in Figure 16. In this thesis, the Pearson's correlation coefficient r is used, which is calculated by

$$r = \frac{\sum_{i=1}^{N}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \overline{x})^2 \cdot \sum_{i=1}^{N}(y_i - \overline{y})^2}} \tag{13}$$

where N is the total number of the sample data, x_i and y_i the i-th data of x and y, and $\overline{x}$ and $\overline{y}$ the sample means of X and Y which are $\frac{1}{N}\sum_{i=1}^{N} x_i$ and $\frac{1}{N}\sum_{i=1}^{N} y_i$ respectively. The higher the absolute values of correlation coefficients are, the stronger the correlations between the two random variables. Correlation coefficients range from -1 to 1, and the correlation coefficient of the given example is $r = \frac{30}{\sqrt{16 \times 60}} \approx 0.9682$. A rule of thumb interprets the correlation coefficient as follows, though it depends on the purpose of the analysis and on the number of sample data. For the absolute value of r, $[0.9, 1]$ shows a very high correlation, $[0.7, 0.9)$ a high correlation, $[0.5, 0.7)$ a moderate correlation, $[0.3, 0.5)$ a low correlation, and $[0, 0.3)$ little if any correlation [3]. Thus, in the above example, X and Y are said to have a very highly positive correlation.

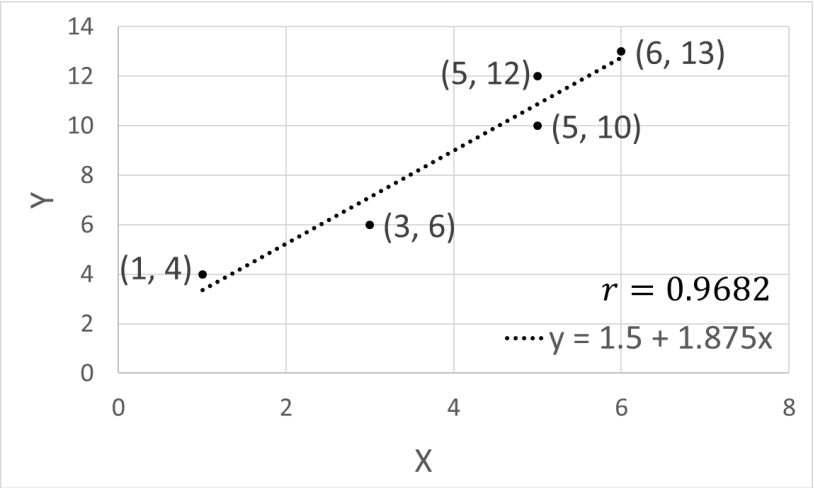


Figure 16: An example of simple linear regression.

The regression line, the best-fitting straight line, is

$$y = \overline{y} + \frac{\sum_{i=1}^{N}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{N}(x_i - \overline{x})^2}(x - \overline{x}) \tag{14}$$

which predicts the values of y given x. Thus, the regression line of the given example is $y = 1.5 + 1.875 \cdot x$, the dotted line shown in Figure 16. The sample standard error of the prediction on Y is then calculated by

$$s = \sqrt{\frac{\sum_{i=1}^{N}(y_i - y_i')^2}{N - 2}} \tag{15}$$

where y_i' is the prediction on the regression line given x_i. In the given example, the sample standard error is $s = \sqrt{\frac{3.75}{3}} \approx 1.1180$. A small value of sample standard error shows that the prediction can be trusted.

3.3 Experiments on 2×4 CDC

In the experiments, all analyzed CDC programs were based on MCTS with different techniques or different parameters introduced in Chapter 2. Although the programs were originally written to play 4×8 CDC, it was generalized to different board sizes. All techniques

incorporated for 4×8 CDC were directly applied to 2×4 CDC. In total, the strengths of 129 program settings, as listed in Table 8, were analyzed. The program settings generally followed the experiments in Chapter 2. All of the experiments were run on machines equipped with Intel (R) Core(TM) i5-3570K CPU 3.40GHz with 8GB memory.

Incorporated Technique	**Number of Simulations Per Action**	**Settings of Parameters**
None	1, 10, 100, 500, 1,000, 2,000, 5,000, 10,000, 20,000, 30,000, 50,000, 100,000, 200,000	None
EPT	1,000 30,000	$\varepsilon_{EPT} \in \{0, 0.05, 0.1, 0.2, 0.3, 0.4\}$
IMB		$w_{IMB} \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$
SL		$(a_L, k_L) \in \{0.2, 0.25, 0.3\} \times \{3, 5, 7, 9, 11\}$
TSQ		$a_T = 0.3,\ k_T = 11,\ c_w \in \{0, 0.01, 0.02, 0.03, 0.04\}$ $a_T = 0.3,\ c_w = 0.02,\ k_T \in \{7, 9, (11,)\ 13, 15\}$ $k_T = 11,\ c_w = 0.02,\ a_T \in \{0.2, 0.25, (0.3,)\ 0.35\}$
PB		$\mathbf{H^B},\ \sqrt{N(s,a)},\ c_{PB} \in \{0.001, 0.01, 0.1, 1, 10\}$ $\mathbf{H^B} + \mathbf{H^C},\ \sqrt{N(s,a)},\ c_{curb} = 1/4,$ $c_{PB} \in \{0.001, 0.01, 0.1, 1, 10\}$ $\mathbf{H^B} + \mathbf{H^C} + \mathbf{H^F},\ \sqrt{N(s,a)},\ c_{curb} = 1/4,$ $c_{flip} = 1/12,\ c_{PB} \in \{0.001, 0.01, 0.1, 1, 10\}$

Table 8: Details of the 129 analyzed program settings for 2×4 CDC.

In the following subsections, for each of the settings of MCTS above, win rates are collected given five different baseline programs in Subsection 3.3.1, prediction rates are collected given three different sets of expert actions in Subsection 3.3.2, and mean squared errors are collected given three different sources of ground-truth values in Subsection 3.3.3. All the detailed values of win rates, prediction rates, and mean squared errors are presented in Appendix B.

3.3.1 Win Rates

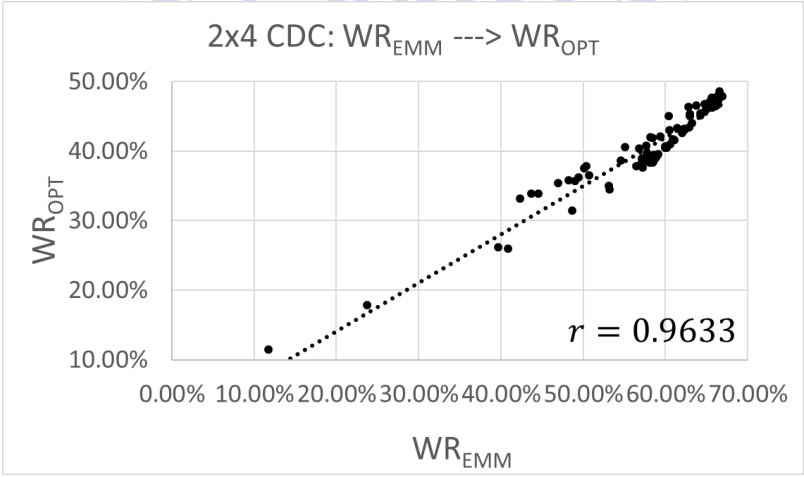
This subsection investigates whether the win rates (WRs) against practical and available baseline programs are correlated to the win rates against the optimal player (absolute strengths). In the experiments, the win rate of a program against a designated baseline program was obtained in the following way. The program played against the designated baseline program 1,000 games, with altering turns playing first, in each of the 24 combinations as listed in Table 3. Its win rate was the average score of the 24,000 games where a win was counted as 1, a draw as 0.5, and a loss as 0.

In 2×4 CDC, five baseline programs were considered. The first was the optimal player (OPT) as described in Subsection 3.1.1. The second was a four-ply[1] expectiminimax player (EMM) [110] using weighted material sum (as described in Subsection 2.4.2) as heuristic evaluation. EMM had a win rate of 35.22% (±0.57%) against OPT. The remaining three baseline programs were the original MCTS with 1,000, 5,000, and 30,000 simulations per action (MCTS-1k, MCTS-5k, and MCTS-30k), which had win rates of 37.67% (±0.57%), 43.41% (±0.57%), and 46.78% (±0.58%) against OPT respectively. The five baseline programs from the weakest to the strongest were EMM, MCTS-1k, MCTS-5k, MCTS-30k, and OPT. In the experiments, EMM was intentionally included to have a baseline program not based on MCTS but also practically available.

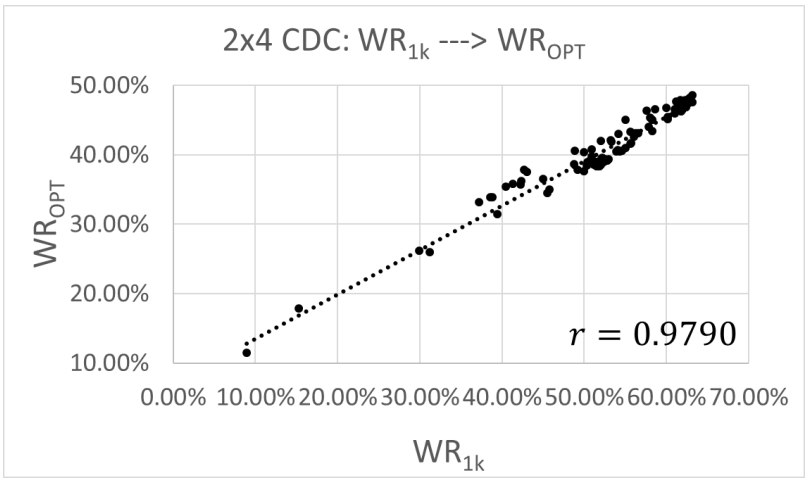
In order to analyze the correlation between win rates against practical and available baseline programs and those against OPT, each of the 129 program settings (listed in Table 8) played against the five baseline programs respectively. For these program settings, let WR_{OPT}, WR_{EMM}, WR_{1k}, WR_{5k} and WR_{30k} be the win rates respectively against OPT, EMM, MCTS-1k,

[1] Six-ply EMM ran about 150 times slower than that of four-ply. In average, six-ply EMM cost 1.78 seconds per action, while four-ply EMM only cost 0.01. Due to time constraints, experiments on EMM with more plies were not conducted.

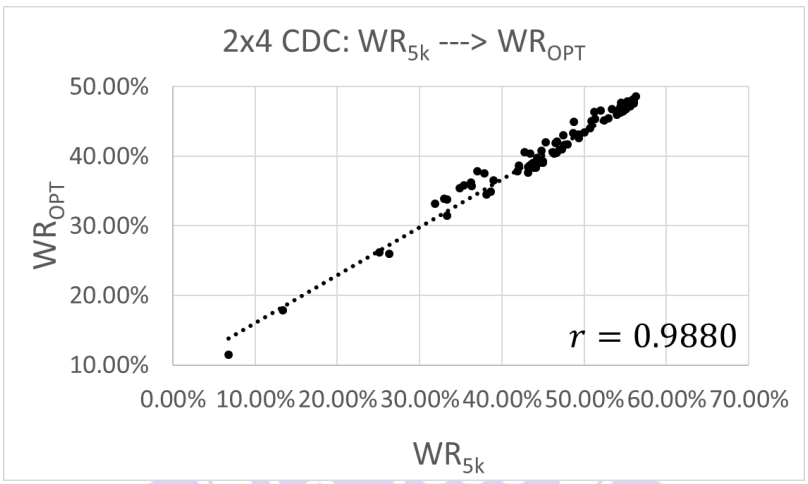
MCTS-5k, and MCTS-30k. Figure 17 (a)-(d) show respectively WR_{EMM}, WR_{1k}, WR_{5k}, and WR_{30k} with respect to the absolute strength WR_{OPT} for all program settings. These plots show very highly positive correlations whose correlation coefficients r (Formula (13)) were 0.9633, 0.9790, 0.9880, and 0.9922 respectively. The stronger the baseline program was, the higher the correlation coefficient obtained. The sample standard errors (Formula (15)) with respect to the predicted lines (Formula (14), the dotted lines in Figure 17) were 1.57%, 1.19%, 0.90%, and 0.73% respectively. The above observations suggested that the baseline program should be as strong as possible in order to obtain accurate strengths of game-playing programs when the optimal player was not available. However, this would be a tradeoff between the accuracy of the metric and the time needed to perform experiments, since a stronger baseline program usually implies more computation time.



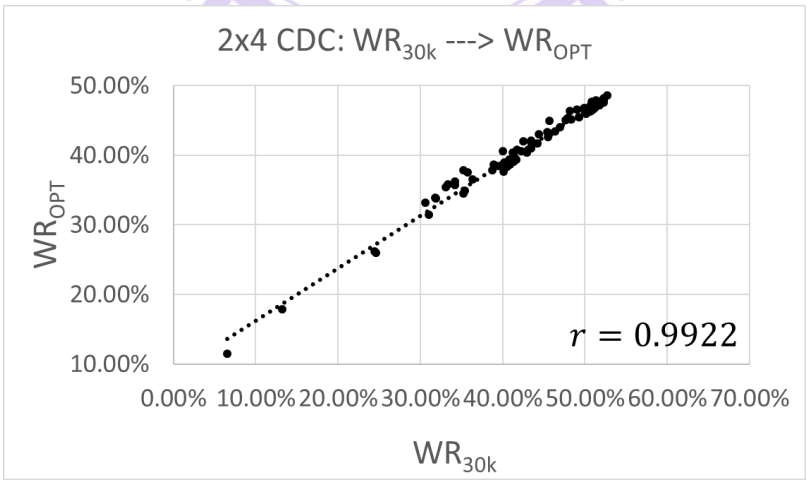
(a)



(b)



(c)



(d)

Figure 17: Scatter plots for (a) WR_{EMM}, (b) WR_{1k}, (c) WR_{5k}, and (d) WR_{30k} to WR_{OPT}.

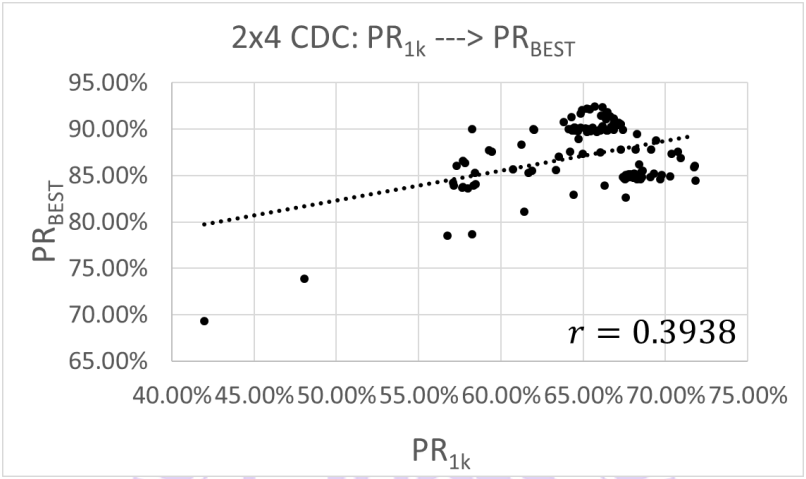
3.3.2 Prediction Rates

This subsection investigates whether the prediction rates (PRs) to practical and available expert actions are correlated to those to the best actions (actions by OPT), and whether all of these prediction rates are correlated to the absolute strengths (win rates against OPT). A set of 2,400 games, 100 in each of the 24 material combinations listed in Table 3, were collected and about 47,000 positions in the game set were used for experiments on prediction rates.

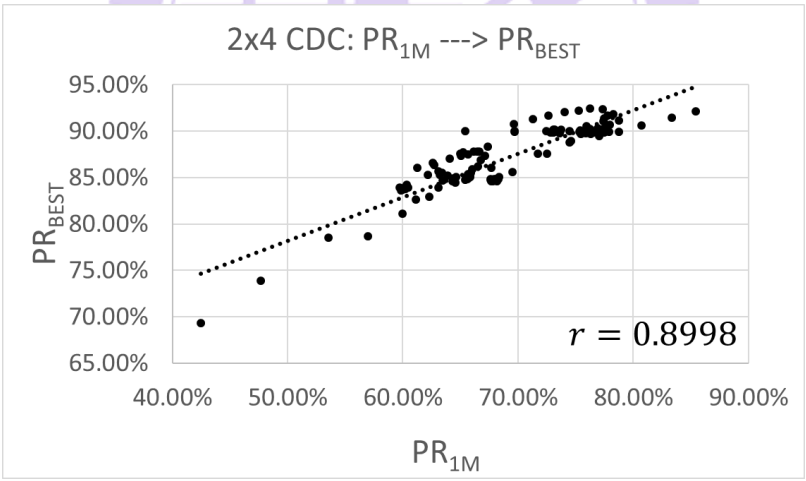
Three sets of expert actions were considered, the best actions as described in Subsection 3.1.2, the actions by MCTS-1k, and the actions by MCTS-1M (the original MCTS with 1,000,000 simulations per action). MCTS-1k and MCTS-1M had win rates of 37.67% (±0.57%) and 49.21% (±0.58%) against OPT respectively. Note that MCTS-1k was intentionally included to investigate the cases where the quality of the collected expert actions was low. To obtain the prediction rate of a program, the program played actions for the positions in the selected game set. In the experiment, both MCTS-1k and MCTS-1M obtained prediction rates respectively to the best actions with 84.46% (±0.33%) and 93.31% (±0.23%). The results clearly showed that MCTS-1M played the best actions more often than MCTS-1k.

In order to analyze the correlation of prediction rates, each of the 129 program settings (listed in Table 8) played actions for the positions in the selected game set. For these program settings, let PR_{BEST}, PR_{1k} and PR_{1M} be the prediction rates respectively to the best actions, the actions by MCTS-1k, and the actions by MCTS-1M. Figure 18 (a) and (b) show respectively PR_{1k} and PR_{1M} with respect to PR_{BEST} for all program settings. Note that the values in PR_{BEST} tended to be higher than those in PR_{1k} and PR_{1M}, for the reason that it was easier to match best actions since a position may have multiple best actions as described in Subsection 3.1.2. From the experiment results, as expected, PR_{1k} could not reflect PR_{BEST} well, which only had a correlation coefficient of 0.3938. On the contrary, PR_{1M} had a highly positive correlation to

PR_{BEST}, which had a correlation coefficient of 0.8998. The sample standard errors with respect to the predicted lines were 3.29% and 1.56%. The results showed that the quality of expert actions influenced the results of prediction rates greatly, and that prediction rates to actions by a stronger program can be used to approximate those to the best actions.



(a)

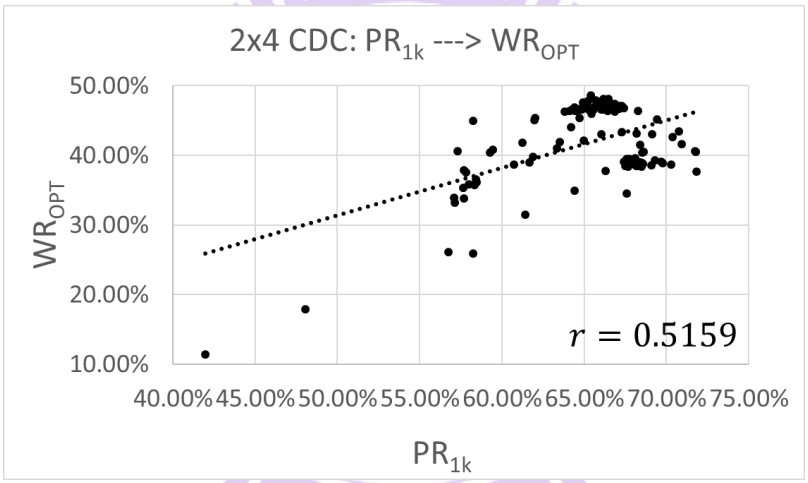


(b)

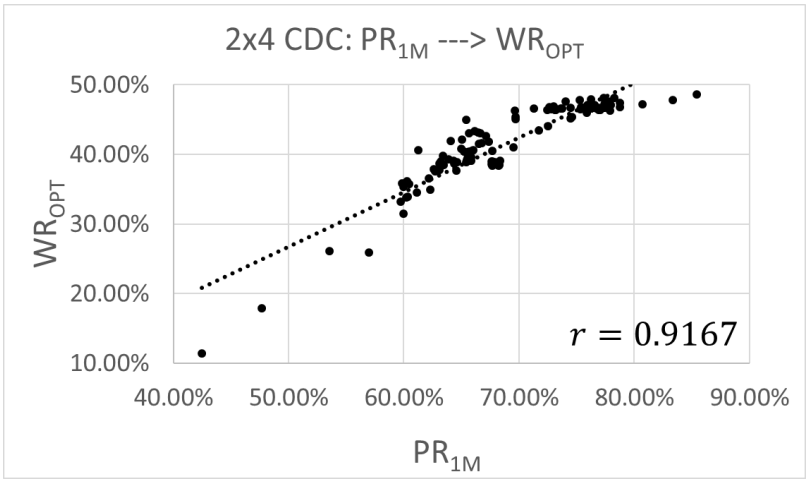
Figure 18: Scatter plots for (a) PR_{1k} and (b) PR_{1M} to PR_{BEST}.

Next, the correlation between the prediction rates and the absolute strengths (win rates against OPT) was investigated. Figure 19 (a)-(c) show the prediction rates, PR_{1k}, PR_{1M} and PR_{BEST}, with respect to WR_{OPT}. Similar to those shown in Figure 18, PR_{1k} was not a good metric to the absolute strengths either, which only had a correlation coefficient of 0.5159. PR_{1M} had a

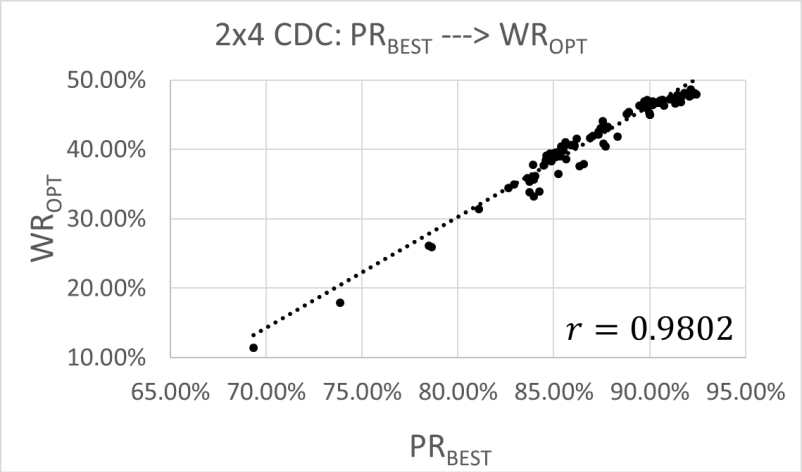
very highly positive correlation coefficient of 0.9167 to the absolute strengths. The sample standard errors with respect to the predicted lines were 5.00% and 2.33% respectively. These suggested that when expert actions with high quality were available, prediction rates to those expert actions were good metrics to the strengths of game-playing programs, though not as good as win rates playing against a baseline program as shown in Subsection 3.3.1. In addition, PR_{BEST} had a higher correlation to absolute strengths than both PR_{1k} and PR_{1M}. Namely, the correlation coefficient of PR_{BEST} to WR_{OPT} was 0.9802, and the sample standard error with respect to the predicted line was 1.16%. The analyses showed that the strengths of game-playing programs had strong correlations to the ability to select the best actions.



(a)



(b)



(c)

Figure 19: Scatter plots for (a) PR_{1k}, (b) PR_{1M}, and (c) PR_{BEST} to WR_{OPT}.

3.3.3 Mean Squared Errors

For 2×4 CDC, it is straightforward to use the theoretical values (as described in Subsection 3.1.3) as the ground-truth values to analyze mean squared errors (MSEs) of given programs. In addition to this set, let the moves played by MCTS-1M be regarded as expert actions, and thus the outcomes of these games be another source of the ground-truth values. The correlation of mean squared errors with different sources of ground-truth values is then analyzed. In addition, this subsection also studies whether all of these mean squared errors are correlated to the absolute strengths (win rates against OPT).

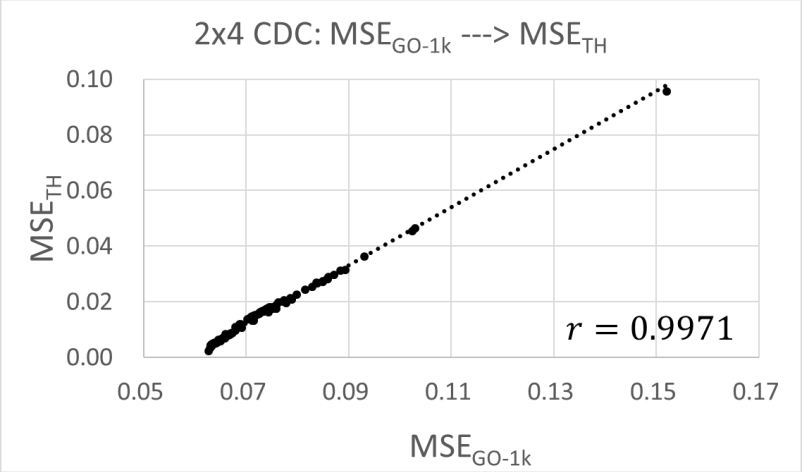
In the experiments, the same set of about 47,000 positions as in Subsection 3.3.2 was used. Two sources of ground-truth values for these positions were considered, the theoretical values and the outcomes of games from self-plays by MCTS-1M. Namely, a self-play game was performed by MCTS-1M from a given position to the end of the game. The game outcomes, consisting of wins, draws, and losses, were converted to 1's, 0's and -1's respectively, and subsequently used as a source of ground-truth values of the positions. Without loss of generality, all of the values were from the red player's point of view. To obtain a mean squared error of a

program, the program ran MCTS on each of the positions in the game set. The value of a position p estimated by the program, $\hat{v}_p$, was the Q value of the root s_0 of the search tree for the position p. More specifically, the Q value of the root s_0 was calculated by

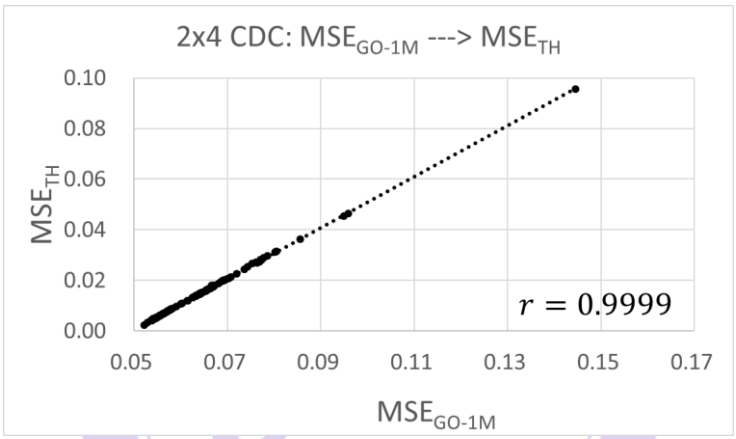
$$\frac{\sum_a \big(Q(s_0,a)\cdot N(s_0,a)\big)}{\sum_a N(s_0,a)} \tag{16}$$

where $Q(s_0,a)$ and $N(s_0,a)$ are the value and the visit count of action a for state s_0 respectively. Mean squared errors were then calculated based on the given ground-truth values and the collected $\{\hat{v}_p\}$. Note that mean squared errors were scaled by a factor of 1/4 to the range of [0, 1] in this thesis. To investigate whether game outcomes with a lower quality influenced the results of mean squared errors, those from self-plays by MCTS-1k were included as the third source of ground-truth values.

In order to analyze the correlation of mean squared errors, each of the 129 program settings (listed in Table 8) ran MCTS on all positions in the game set. For these programs, let MSE_{TH}, $MSE_{GO\text{-}1k}$ and $MSE_{GO\text{-}1M}$ be the mean squared errors respectively with theoretical values, game outcomes from self-plays by MCTS-1k, and game outcomes from self-plays by MCTS-1M. Figure 20 (a) and (b) show respectively $MSE_{GO\text{-}1k}$ and $MSE_{GO\text{-}1M}$ with respective to MSE_{TH} for all program settings. Note that the values in MSE_{TH} tended to be lower than those in $MSE_{GO\text{-}1k}$ and $MSE_{GO\text{-}1M}$, for the reason that the granularity of theoretical values was lower than that of game outcomes. The experiment results showed very highly positive correlation coefficients, 0.9971 and 0.9999, and the sample standard errors with respect to the predicted lines were 0.0008 and 0.0002. These showed that mean squared errors with different sources of game outcomes could predict those with respect to theoretical values well even when the *expert* had a lower quality such as MCTS-1k.



(a)

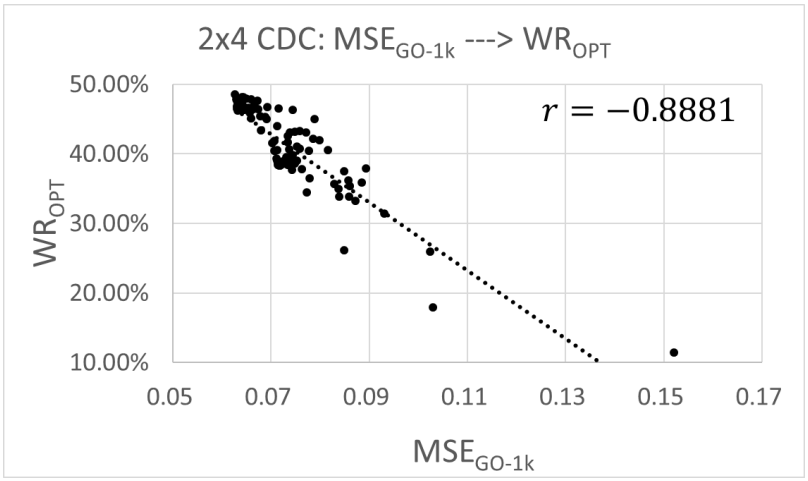


(b)

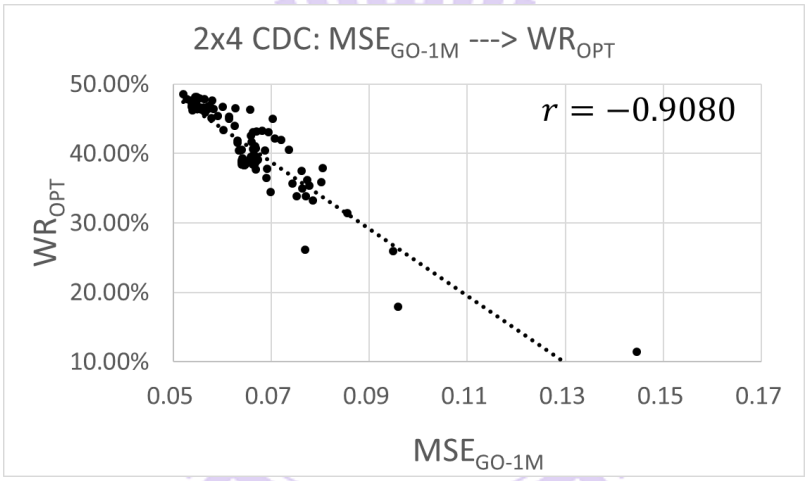
Figure 20: Scatter plots for (a) MSE_{GO-1k} and (b) MSE_{GO-1M} to MSE_{TH}.

Next, the correlation between the mean squared errors and the absolute strengths (win rates against OPT) was investigated. Figure 21 (a)-(c) show the mean squared errors MSE_{GO-1k}, MSE_{GO-1M}, and MSE_{TH} with respect to WR_{OPT}. All showed highly negative correlations, which had correlation coefficients of -0.8881, -0.9080, and -0.9078 respectively, and the sample standard errors with respect to the predicted lines were 2.68%, 2.45%, and 2.45%. Note that the results here showed negative correlations since stronger game-playing programs tended to have lower mean squared errors. Generally, mean squared errors were good metrics to the strengths of game-playing programs, though not as good as win rates playing against a baseline program as shown in Subsection 3.3.1. However, it was a relatively stable metric compared to prediction

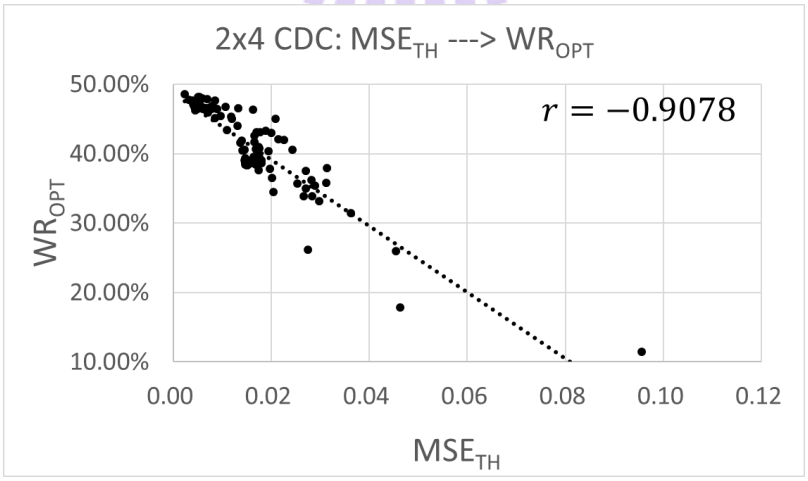
rates, since game outcomes with a lower quality did not influence this metric too much.



(a)



(b)



(c)

Figure 21: Scatter plots for (a) MSE$_{GO-1k}$, (b) MSE$_{GO-1M}$, and (c) MSE$_{TH}$ to WR$_{OPT}$.

3.4 Experiments on Predicting Win Rates from Smaller Variants

In this section, the absolute strengths of game-playing programs in 2×4 CDC were used to help analyze the strengths of game-playing programs in 4×8 CDC. Since games with different board sizes may have similar properties, it is worth investigating whether the experiment results obtained in smaller sizes can be used to predict those in the original size or a bigger size.

Incorporated Technique	**Number of Simulations Per Action**	**Settings of Parameters**
None	2,000, 5,000, 10,000, 20,000, 30,000, 50,000, 100,000, 200,000	None
EPT	30,000	$\varepsilon_{EPT} \in \{0, 0.05, 0.1, 0.2, 0.3, 0.4\}$
IMB		$w_{IMB} \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$
SL		$(a_L, k_L) \in \{0.2, 0.25, 0.3\} \times \{3, 5, 7, 9, 11\}$
TSQ		$a_T = 0.3,\ k_T = 11,\ c_w \in \{0, 0.01, 0.02, 0.03, 0.04\}$ $a_T = 0.3,\ c_w = 0.02,\ k_T \in \{7, 9, (11,)\ 13, 15\}$ $k_T = 11,\ c_w = 0.02,\ a_T \in \{0.2, 0.25, (0.3,)\ 0.35\}$
PB		$\mathbf{H^B},\ \sqrt{N(s,a)},\ c_{PB} \in \{0.001, 0.01, 0.1, 1, 10\}$ $\mathbf{H^B} + \mathbf{H^C},\ \sqrt{N(s,a)},\ c_{curb} = 1/4,$ $c_{PB} \in \{0.001, 0.01, 0.1, 1, 10\}$ $\mathbf{H^B} + \mathbf{H^C} + \mathbf{H^F},\ \sqrt{N(s,a)},\ c_{curb} = 1/4,$ $c_{flip} = 1/12,\ c_{PB} \in \{0.001, 0.01, 0.1, 1, 10\}$

Table 9: Details of the 66 analyzed program settings for both 2×4 and 4×8 CDC.

The details of the 66 program settings for both 2×4 and 4×8 CDC are listed in Table 9. For each program setting, let the two win rates WR_{2x4} and WR_{4x8} be obtained as follows. WR_{2x4} was the win rate against OPT (the optimal player) in 2×4 CDC which was the average score of 24,000 games as described in Subsection 3.3.1. WR_{4x8} was the win rate against MCTS-30k (the original DARKKNIGHT with 30,000 simulations per action), the baseline program used in

Chapter 2. Most of the win rates in WR_{4x8} were presented in Chapter 2, where each win rate in 4×8 CDC was the average score of 1,000 games.

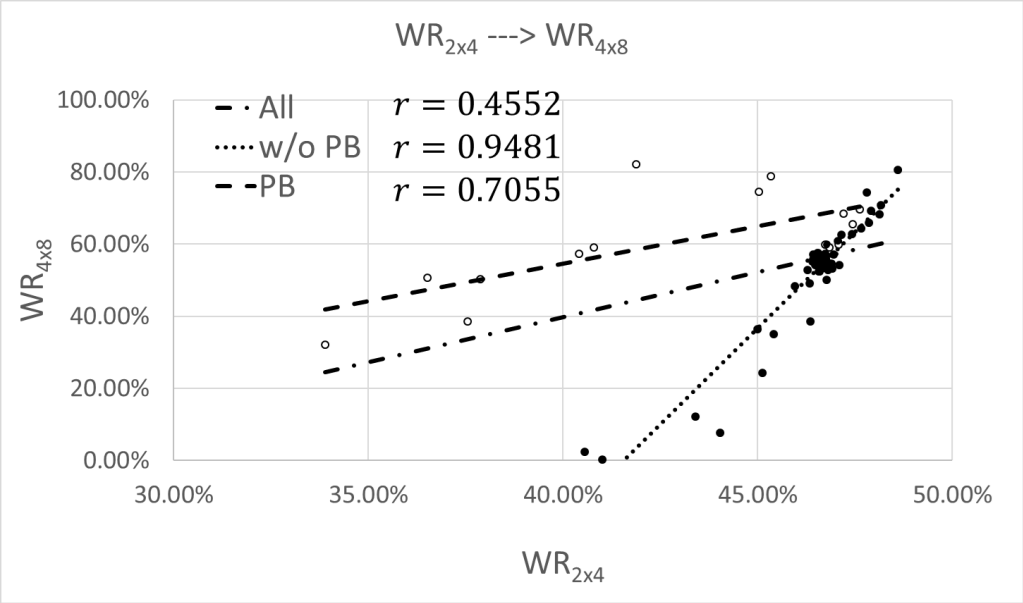


Figure 22: Scatter plot for WR_{2x4} to WR_{4x8} where the hollow points are related to program settings with PB.

Figure 22 shows that WR_{2x4} were not highly correlated to WR_{4x8} for all program settings listed in Table 9. The correlation coefficient of 0.4552 showed a low positive correlation only, and the sample standard error with respect to the predicted line (the dot-dashed line in Figure 22) was 14.51% which was high. In addition to a lower correlation coefficient, another reason that the sample standard error here was relatively higher than those in Section 3.3 was that the number of sample data were lower here (66 vs. 129). After checking the distribution of points in Figure 22, the points at the top-left part (hollow points) were all from the program settings with PB. By removing those related to PB, a much higher correlation was obtained with a correlation coefficient of 0.9481. The sample standard error with respect to the new predicted line (the dotted line in Figure 22) was 5.27%. In addition, the correlation coefficient for the program settings with PB was 0.7055. Although a high positive correlation was obtained, only 15 sample data were included. The sample standard error with respect to the predicted line (the

dashed line in Figure 22) was 10.15%, which was still high.

In brief, WR_{2x4} can be roughly used to predict WR_{4x8} except for PB. In order to investigate the inconsistency of PB, the win rates from different program settings with PB, shown in Appendix B, were further analyzed. It was observed that PB in 2×4 CDC obtained higher win rates with $c_{PB} = 0.01$ for all of the three experimented heuristics, $\mathbf{H^B}$, $\mathbf{H^B + H^C}$ and $\mathbf{H^B + H^C + H^F}$. However, it was $c_{PB} = 0.1$ for 4×8 CDC. Judging by this, it was speculated that heuristic scores of actions might not generalize well to different board sizes. Thus, the heuristics used in PB were further experimented. More specifically, in both 2×4 and 4×8 CDC, the strengths of the strongest heuristics $\mathbf{H^B + H^C + H^F}$ were measured by the original MCTS with different numbers of simulations per action. In 2×4 CDC, the program using the heuristics without search had a playing strength approximate to the original MCTS with only 30 simulations per action, where the win rate was 50.41% (±0.52%). In contrast, the program using the same heuristics without search had a playing strength approximate to the original MCTS with 1,800 simulations per action in 4×8 CDC, where the win rate was 50.30% (±2.08%). The results showed that the advantages brought by heuristic scores of actions were much higher in 4×8 CDC than in 2×4 CDC. From the above, the inconsistency of PB was conjectured to be a result of the used heuristic scores of actions. However, this should be justified with further research, and therefore is left as an open problem in the future.

3.5 Chapter Conclusions

In this chapter, three metrics measuring strengths of game-playing programs are analyzed in a solved game, 2×4 CDC. The commonly used metric, win rates playing against a baseline program, has been shown to have the highest correlation to the absolute strengths of programs (win rates against the optimal player). Moreover, using stronger baseline programs reflects the

absolute strengths better. The analysis in this chapter offers evidence that win rates against a baseline program – the stronger the better – can be used to approximate the absolute strengths of programs.

In addition, the second metric, prediction rates to expert actions, also has a high correlation to the absolute strengths, as long as expert actions with high qualities were collected. The third metric, mean squared errors to values of positions, also has a high correlation to the absolute strengths, which is influenced less by the quality of the collected data. The results show that the strengths of game-playing programs can be roughly measured by these two metrics.

The win rates for game-playing programs with the same settings in both 2×4 and 4×8 CDC are also analyzed. Except for those related to PB, the win rates obtained in different board sizes also have a high correlation. A conjecture for the inconsistency of PB is related to the used heuristic scores of actions. Further research is needed to study the condition that the win rates obtained in smaller sizes can or cannot be used to predict those in the full or larger sizes. In the future, it is also worth applying the analysis presented in this chapter to other solved games, even deterministic games such as Hex [47, 52, 102].

Chapter 4 Applying AlphaZero to 2×4 CDC

The AlphaZero algorithm [122] achieved superhuman levels of play in three classical board games chess, shogi, and Go without domain-specific knowledge except game rules. It is worthy studying whether the AlphaZero algorithm also learns theoretical values and optimal plays even in stochastic games. The algorithm has been demonstrated to be successfully applied to deterministic games including chess, shogi, and Go. The theoretical values of the positions in such games are expected to be a win, loss, or draw. On the contrary, in stochastic games such as Chinese dark chess (CDC), the theoretical values are expected win rates. It is interesting to investigate the ability of the AlphaZero algorithm to approximate expected win rates. However, CDC is still too large to be solved. The reduced 2×4 CDC is used in this thesis to verify whether the AlphaZero algorithm approximates to the expected win rates.

In addition, the AlphaZero algorithm have some hyper-parameters. Silver *et al.* [123] tuned hyper-parameters in AlphaGo Zero by Bayesian optimization [117]. In the AlphaZero algorithm, most of the hyper-parameters followed those tuned in AlphaGo Zero [122]. However, they specifically tuned one of the hyper-parameters (the α in Dirichlet noise) for different games according to the approximate number of legal actions. It is also interesting to investigate how the AlphaZero algorithm is influenced by hyper-parameters.

The rest of this chapter is organized as follows. Section 4.1 reviews the AlphaZero algorithm. Section 4.2 illustrates a tabular AlphaZero algorithm applied to 2×4 CDC. Lookup tables are used instead of neural networks to keep for each position the probabilities of moves and the estimated value, since the state space is small enough. The experiment results are then included in Section 4.3. Finally, Section 4.4 makes concluding remarks of this chapter.

4.1 AlphaZero Algorithm

In 2017, Silver *et al.* [123] presented a program named AlphaGo Zero which achieved a superhuman level in the game of Go. Starting from random play, the program was trained from self-play games, and did not use domain-specific knowledge other than game rules. This is a kind of *reinforcement learning* [130] algorithm. They then generalized the approach to the AlphaZero algorithm [122], and successfully applied it to chess and shogi as well as Go. The trained programs defeated world-champion programs in each of the three games.

In AlphaGo Zero [123], deep neural networks (DNNs) were trained by reinforcement learning to predict policies (i.e., probability distribution of actions) and values given positions. The key idea was to make the AlphaGo program become its own teacher through tree search and self-play. The policies and the values generated by DNNs were incorporated into tree search to obtain actions with higher quality, and then the DNNs were trained to predict the actions selected by tree search and the final winner in the self-play games.

In more detail, the learning algorithm contained three parts including self-play, optimization, and evaluation. These three parts were executed asynchronously in parallel in a large-scale computation system.

Self-play games were generated by PUCT, a variant of MCTS, with 1,600 simulations per action. PUCT in AlphaGo Zero contained three phases in each simulation, which were selection, expansion, and backpropagation. In the selection phase, PUCB function

$$\underset{a}{\operatorname{argmax}}\left\{\frac{W(s,a)}{N(s,a)}+c_{puct}\cdot P(s,a)\cdot\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}\right\} \tag{17}$$

was applied to traverse the search tree to a leaf, where $W(s,a)$ is the total value of action a for node s, $N(s,a)$ the visit count, $P(s,a)$ the prior probability of selecting a at s, and c_{puct} a constant influencing the level of exploration in search. During the expansion phase, the

leaf node s_L was evaluated by DNN to obtain the value v and the policy $\boldsymbol{p}$, and then expanded with each of the actions initialized to $N(s_L, a) = 0$, $W(s_L, a) = 0$ and $P(s_L, a) = p_a$. Backpropagation updated the traversed nodes by $N(s, a) = N(s, a) + 1$ and $\mathrm{W}(s, a) = W(s, a) + v$. Finally, the action to be actually played at the root was chosen in proportion to the visit count. More specifically, the probability to select action a at the root s_0 was

$$\pi(a|s_0) = \frac{N(s_0, a)^{\frac{1}{\tau}}}{\sum_b N(s_0, b)^{\frac{1}{\tau}}} \tag{18}$$

where τ is a temperature parameter determining the level of exploration in self-play games. At the end of a self-play game, the policies $\boldsymbol{\pi}$'s and the final outcome z of the game were saved as training data.

To ensure all legal actions at the root may be tried, Dirichlet noises [44] were added to the prior probabilities in the root node. The prior probability $P(s_0, a)$ became

$$P(s_0, a) = (1 - \varepsilon)p_a + \varepsilon\eta_a \tag{19}$$

where η_a is sampled from Dirichlet distribution $Dir(\alpha)$ with α controlling the level of concentration, and ε is the weight of the noise. Under the same numbers of samples, lower α's generally lead to more concentrated distribution, and thus higher levels of exploration.

In optimization, AlphaGo Zero randomly sampled positions from the most recent 500,000 self-play games. The DNNs were then optimized by stochastic gradient descent using the positions and the corresponding policies and values. The optimized DNN was evaluated by playing games against the current *best* DNN. Both of the DNNs were incorporated into PUCT with 1,600 simulations per action. If the new DNN obtained a win rate greater than 55% in 400 games, it became the new best DNN. The self-play part always used the best DNNs to generate games, and the first best DNN was the one with weights randomly initialized. The evaluation part was to ensure that the quality of the collected self-play games either remained the same or became better. The resulting program achieved superhuman performance, defeating the

AlphaGo program that won against the world champion in 2016 [121].

The AlphaZero algorithm [122] was generalized from AlphaGo Zero. First, in addition to binary outcomes of wins and losses, draws were taken into consideration. Second, the evaluation part was removed, which meant that only one DNN was maintained and updated continually. The algorithm was tested on three games which were chess, shogi, and Go, and achieved superhuman levels of play.

In AlphaGo Zero, the hyper-parameters related to PUCT were selected by Bayesian optimization [117, 123]. The AlphaZero algorithm reused the hyper-parameters for all the three games, except the α parameter in the Dirichlet noise. The value was scaled proportional to the approximate numbers of legal actions in a typical position of the games. More specifically, the values were 0.03 for Go, 0.15 for shogi, and 0.3 for chess.

The codes of AlphaGo Zero and AlphaZero have not been released yet. The success of the algorithm led open-sourced projects such as Leela Zero [78], Mingo [133], and ELF Open Go [43] try to reproduce the results of the algorithm in the game of Go.

4.2 Tabular AlphaZero for 2×4 CDC

This thesis replaces the DNNs in the AlphaZero algorithm by lookup tables since the state space is small enough. In a lookup table for 2×4 CDC, each position has its own policy and value. The policy is represented by a vector of real numbers $\hat{\boldsymbol{p}}$. The size of the vector is 40, which is the total number of all possible actions in 2×4 CDC (eight for flipping a1(?), a2(?), …, b4(?) and the rest for moving a1-a2, a1-a3, a1-a4, a1-b1, a2-a1, …, b4-b3). The probability distribution of the actions is calculated by the softmax function [15],

$$p_a = \frac{e^{(\hat{p}_a)}}{\sum_b e^{(\hat{p}_b)}} \tag{20}$$

where p_a is the probability of selecting action a and $\hat{p}_a$ the policy value of action a in

lookup table. A real number $\hat{v}$ is used to represent the value of a position, and is scaled by the sigmoid function [51] to the range of $[0, 1]$,

$$v = \frac{1}{(1 + e^{-\hat{v}})} \tag{21}$$

where a v of 0 represents 100% loss and 1 represents 100% win. To compare learned values (v) with the theoretical values, the range of theoretical values is linearly scaled from $[-1, 1]$ to $[0, 1]$ in this chapter without loss of generality.

From the collected self-play games, the updates for one position, with the corresponding policy $\boldsymbol{\pi}$ and the game outcome z, are performed as follows. For policy, the goal is to have the estimated probability distribution $\boldsymbol{p}$ (Formula (20)) similar to $\boldsymbol{\pi}$. As for value, the goal is to have the estimated value v (Formula (21)) close to z. Following AlphaGo Zero, the loss functions for policy and value [123] apply cross-entropy losses

$$L_p = -\boldsymbol{\pi}^T \ln \boldsymbol{p} = -\sum_a \pi_a \cdot \ln(p_a) \tag{22}$$

and mean squared error

$$L_v = \frac{1}{2}(v - z)^2 \tag{23}$$

respectively. Thus, policy and value in lookup tables are updated according to stochastic gradient decent [89, 107] by

$$\hat{p}_a \leftarrow \hat{p}_a - lr \cdot (p_a - \pi_a) \tag{24}$$

and

$$\hat{v} \leftarrow \hat{v} - lr \cdot (v - z) \cdot v \cdot (1 - v) \tag{25}$$

respectively, where lr is the learning rate.

The gradients for policy $(p_a - \pi_a)$ and value $((v - z) \cdot v \cdot (1 - v))$ are derived as follows. For policy, the partial derivative of the cross-entropy loss L_p (Formula (22)) to the policy value $\hat{p}_a$ of a specific action a is

$$\begin{aligned}\frac{\partial L_p}{\partial \hat{p}_a} &= \frac{\partial(-\sum_k \pi_k \cdot \ln(p_k))}{\partial \hat{p}_a} \\ &= -\sum_k \pi_k \frac{\partial \ln(p_k)}{\partial \hat{p}_a} \\ &= -\sum_k \pi_k \frac{\partial \ln(p_k)}{\partial p_k}\frac{\partial p_k}{\partial \hat{p}_a} \\ &= -\sum_k \pi_k \frac{1}{p_k}\frac{\partial p_k}{\partial \hat{p}_a}\end{aligned} \tag{26}$$

where the fourth equality is according to the differentiation rule for logarithmic functions. The partial derivative of the probability p_k of action k to the policy value $\hat{p}_a$ of action a is

$$\begin{aligned}\frac{\partial p_k}{\partial \hat{p}_a} &= \frac{\partial\left(e^{\hat{p}_k}/\sum_b e^{\hat{p}_b}\right)}{\partial \hat{p}_a} \\ &= \frac{\frac{\partial e^{\hat{p}_k}}{\partial \hat{p}_a}\left(\sum_b e^{\hat{p}_b}\right) - e^{\hat{p}_k}\frac{\partial \sum_b e^{\hat{p}_b}}{\partial \hat{p}_a}}{(\sum_b e^{\hat{p}_b})^2} \\ &= \frac{\frac{\partial e^{\hat{p}_k}}{\partial \hat{p}_a}\left(\sum_b e^{\hat{p}_b}\right) - e^{\hat{p}_k}\cdot e^{\hat{p}_a}}{(\sum_b e^{\hat{p}_b})^2}\end{aligned} \tag{27}$$

where the second equality is according to the quotient rule. Formula (27) can be separated into two cases, $k = a$ and $k \neq a$, and the derivation is Formula (28) and (29) respectively (next page).

$$
\begin{aligned}
\frac{\partial p_k}{\partial \hat{p}_a} &= \frac{\partial p_a}{\partial \hat{p}_a} \\
&= \frac{\frac{\partial e^{\hat{p}_a}}{\partial \hat{p}_a}\left(\sum_b e^{\hat{p}_b}\right) - e^{\hat{p}_a} \cdot e^{\hat{p}_a}}{\left(\sum_b e^{\hat{p}_b}\right)^2} \\
&= \frac{e^{\hat{p}_a} \cdot \left(\sum_b e^{\hat{p}_b}\right) - e^{\hat{p}_a} \cdot e^{\hat{p}_a}}{\left(\sum_b e^{\hat{p}_b}\right)^2} \\
&= \frac{e^{\hat{p}_a} \cdot \left(\sum_b e^{\hat{p}_b} - e^{\hat{p}_a}\right)}{\left(\sum_b e^{\hat{p}_b}\right) \cdot \left(\sum_b e^{\hat{p}_b}\right)} \\
&= p_a \cdot (1 - p_a)
\end{aligned} \tag{28}
$$

$$
\begin{aligned}
\frac{\partial p_k}{\partial \hat{p}_a} &= \frac{\partial p_{k \neq a}}{\partial \hat{p}_a} \\
&= \frac{\frac{\partial e^{\hat{p}_{k \neq a}}}{\partial \hat{p}_a}\left(\sum_b e^{\hat{p}_b}\right) - e^{\hat{p}_{k \neq a}} \cdot e^{\hat{p}_a}}{\left(\sum_b e^{\hat{p}_b}\right)^2} \\
&= \frac{0 \cdot \left(\sum_b e^{\hat{p}_b}\right) - e^{\hat{p}_{k \neq a}} \cdot e^{\hat{p}_a}}{\left(\sum_b e^{\hat{p}_b}\right)^2} \\
&= \frac{-e^{\hat{p}_{k \neq a}} \cdot e^{\hat{p}_a}}{\left(\sum_b e^{\hat{p}_b}\right) \cdot \left(\sum_b e^{\hat{p}_b}\right)} \\
&= -p_{k \neq a} \cdot p_a
\end{aligned} \tag{29}
$$

Substituting the results in Formula (28) and (29) back into Formula (26), the gradient for policy is obtained as follows.

$$
\begin{aligned}
\frac{\partial L_p}{\partial \hat{p}_a} &= -\sum_k \pi_k \frac{1}{p_k} \frac{\partial p_k}{\partial \hat{p}_a} \\
&= -\pi_a \frac{1}{p_a}\left(p_a(1 - p_a)\right) - \sum\nolimits_{k \neq a} \pi_k \frac{1}{p_k}(-p_k \cdot p_a) \\
&= -\pi_a(1 - p_a) + \sum\nolimits_{k \neq a} \pi_k \cdot p_a \\
&= -\pi_a + p_a \cdot \sum\nolimits_k \pi_k \\
&= p_a - \pi_a
\end{aligned} \tag{30}
$$

For value, the partial derivative of mean squared error L_v (Formula (23)) to the value $\hat{v}$ is

$$\begin{aligned}
\frac{\partial L_v}{\partial \hat{v}} &= \frac{\partial L_v}{\partial v} \cdot \frac{\partial v}{\partial \hat{v}} \\
&= \frac{\partial \frac{1}{2}(v-z)^2}{\partial v} \cdot \frac{\partial \frac{1}{(1+e^{-\hat{v}})}}{\partial \hat{v}} \\
&= (v-z) \cdot \frac{0 \cdot \left(1+e^{-\hat{v}}\right) - 1 \cdot \frac{\partial \left(1+e^{-\hat{v}}\right)}{\partial \hat{v}}}{(1+e^{-\hat{v}})^2} \\
&= (v-z) \cdot \frac{-1 \cdot \left(-e^{-\hat{v}}\right)}{(1+e^{-\hat{v}})^2} \\
&= (v-z) \cdot \frac{1 \cdot \left(\left(1+e^{-\hat{v}}\right) - 1\right)}{(1+e^{-\hat{v}}) \cdot (1+e^{-\hat{v}})} \\
&= (v-z) \cdot v \cdot (1-v)
\end{aligned} \tag{31}$$

where the third equality is according to the quotient rule and the fourth is according to the differentiation rule for exponential functions and the chain rule.

Algorithm 1 – Tabular AlphaZero for 2×4 CDC

1: Initialize the lookup table
2: Repeat N iterations
3: Collect M self-play games
4: Optimize by the most recent $K \times M$ games (if any)

A synchronous version of the AlphaZero algorithm, as presented in Algorithm 1, is used to train the lookup tables for 2×4 CDC. At the beginning, the policies and the values in the lookup table are initialized to random values. The lookup table is then incorporated into PUCT to collect M self-play games. The most recent $K \times M$ games, if any, are used to optimize the lookup table. For example, with $K = 2$, the first iteration only uses M games generated in that

iteration to train. The process of self-play and optimization repeats for N times in total.

4.3 Experiments

In this section, settings of experiments are illustrated in Subsection 4.3.1. The results on investigating symmetries of positions in training data is presented in Subsection 4.3.2. The results on varying c_{puct}, Dirichlet α, Dirichlet ε, and temperature τ are then shown in Subsections 4.3.3 to 4.3.6 respectively. Finally, Subsection 4.3.7 gives overall discussions on the results of the four hyper-parameters.

4.3.1 Experiment Settings

In the preliminary study, the experiments were conducted based on the material combination "`PPPP` vs. `pppp`." The total number of possible positions in the games starting from this combination is 194,933. The theoretical value of the initial position is about 0.2286, which shows an obvious disadvantage to the first player[2]. All of the experiments were run on machines equipped with Intel (R) Core(TM) i5-3570K CPU 3.40GHz with 24GB memory.

First, symmetries of positions in training data were investigated. Positions in 2×4 CDC have a total of eight kinds of symmetries. In more detail, the symmetries contain two kinds of color symmetries (original and reversing color) and four kinds of positional symmetries (original, flipping horizontally, flipping vertically, and rotating 180 degrees). The four kinds of positional symmetries are shown in Figure 23 (a) to (d) respectively. Symmetries of positions were considered in AlphaGo Zero for the game of Go, but were excluded in the AlphaZero algorithm since the positions in chess and shogi are asymmetric [122, 123]. Experiments by Silver *et al.* [122] demonstrated that the AlphaZero algorithm without considering symmetries

[2] As mentioned in Section 4.2, the range of the theoretical values is linearly scaled from $[-1,1]$ to $[0,1]$.

had stronger playing strengths in the game of Go under the same numbers of training steps. However, when considering the same amount of wall clock time, the versions with considering symmetries were stronger.

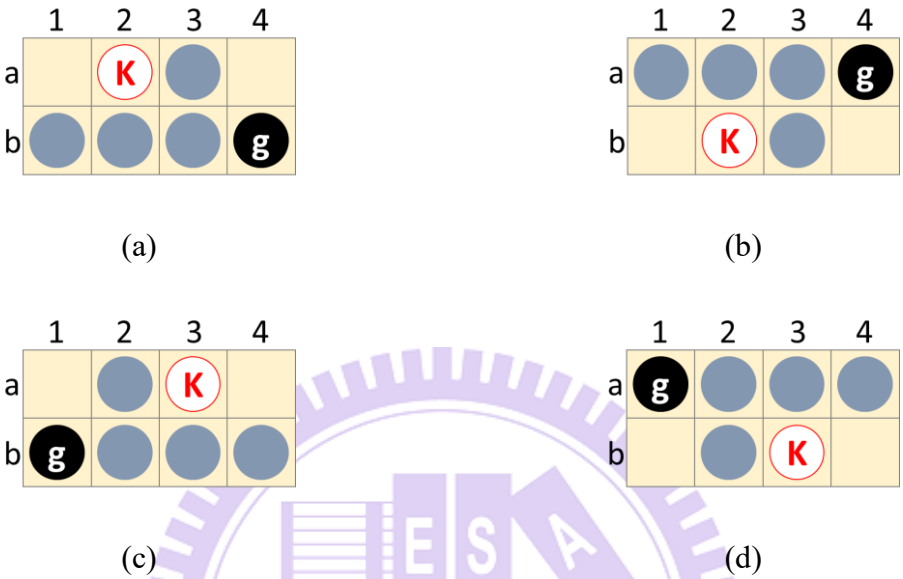


Figure 23: An example of the four kinds of positional symmetries in 2×4 CDC, (a) the origin, (b) flipping horizontally, (c) flipping vertically, and (d) rotating 180 degrees.

Moreover, four hyper-parameters, all related to exploration, c_{puct}, Dirichlet α, Dirichlet ε, and temperature τ were investigated. The default values were 1, 1.5, 0.25, and 1 respectively. In the experiments, the policies and values in the lookup table were initialized by a normal distribution with the mean of 0 and the variance of 0.01. The values of M, N, and K in Algorithm 1 were set to 1,000, 50, and 2 respectively. PUCT in self-play used 800 simulations per action, as Silver *et al.* [122] did for the AlphaZero algorithm. The learning rate lr was set to 1 at the beginning, and then decreased to 0.1 at the 25th iteration to stabilize the learning.

At the end of each iteration, the lookup table was evaluated by the following three metrics. The first was to play 10,000 games against the optimal play (Subsection 3.1.1), with 5,000 games playing as the first player and the other 5,000 as the second. The lookup table agent played the action with the highest probability among the 40 actions. In cases that illegal actions

were selected, the agent lost immediately. This metric aimed to evaluate the learned policy where win rates closing to 50% indicated that the learned policy were close to the optimal play. The second metric was the absolute error (AE) between the estimated value and the theoretical value of the initial position, which was used to judge whether the algorithm was able to learn the theoretical value of the game. The absolute error was used instead of mean squared error (MSE) presented in Subsection 3.1.3 for following reason. AE could present the data better than MSE since many of the errors in the experiments were low. The final metric was the cumulative number of distinct seen positions, which was used to measure the level of exploration. All the detailed values (win rates, AEs, cumulative numbers of distinct positions) collected at the end of each iteration for the experiments on the AlphaZero algorithm are presented in Appendix C.

4.3.2 Symmetries of Positions

Four settings of symmetries for 2×4 CDC were studied, which were *None*, *Color*, *Position*, and *Full*. The four settings considered none, color, positional, and both color and positional symmetries respectively. The training data for the settings of *Color*, *Position*, and *Full* were augmented twice, four times, and eight times respectively according to the corresponding symmetries.

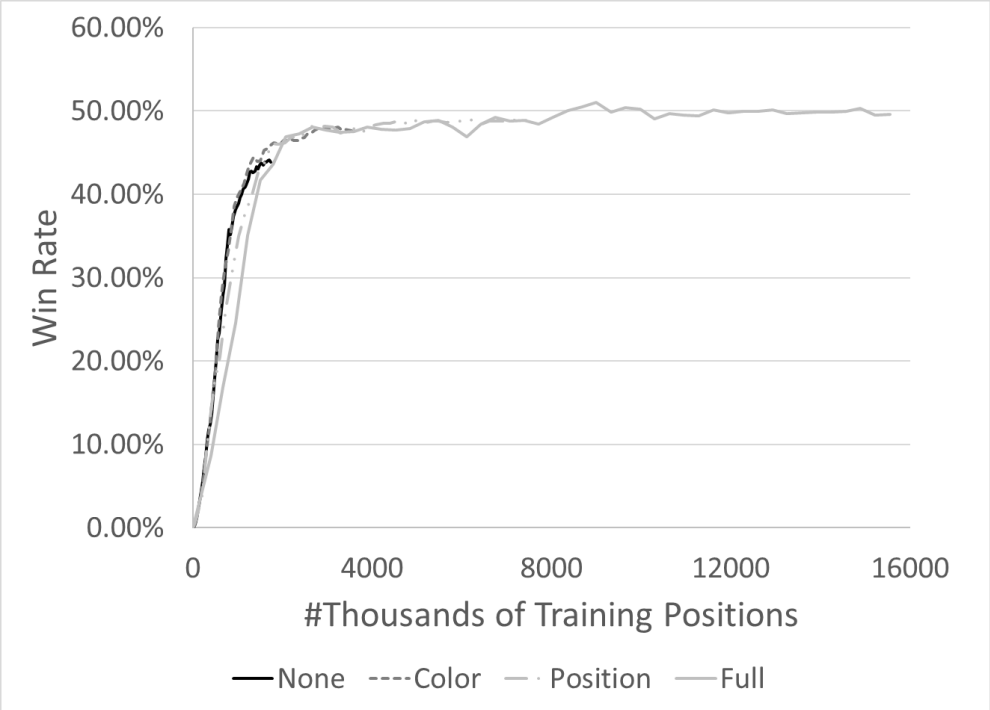
The training curves, composed of the win rates obtained at the end of each iteration, under the scales of iterations and the scales of training positions[3] are shown in Figure 24 (a) and (b) respectively. Since *None* had the least amount of training data, the setting learnt the slowest during the same numbers of self-play games, which is demonstrated in Figure 24 (a). Figure 24 (a) also shows that the training could be speeded up by introducing symmetries of positions, the more the better. In addition, the AEs averaged from the last five iterations for *None*, *Color*,

[3] A training step updates a model (e.g., DNN and lookup table) once. Silver *et al.* [122] uses a mini-batch of 4,096 positions as a training step. In this thesis, a training step only contains one training position. To avoid confusion, the results in this thesis are presented under "training positions".

Position, and *Full* were 0.0754 (±0.0110), 0.0447 (±0.0249), 0.0248 (±0.0155), and 0.0186 (±0.0119) respectively. The total numbers of distinct seen positions during training were 105,748, 121,875, 136,836, and 148,369 respectively. With more symmetries, the AlphaZero algorithm reached lower AEs and saw more distinct positions under the same number of self-play games.

From Figure 24 (b), a consistent tendency with Silver *et al.* [122] can be found. The win rate of *None* was a little higher than that of *Position* and *Full*. In other words, under the same numbers of training positions, the AlphaZero algorithm can generate better training data than just augmenting by symmetries of positions. However, since symmetries indeed speeded up the required time, the rest of the experiments applied *Full* for symmetries.

(a)



(b)

Figure 24: Training curves for different settings of symmetries under the scales of (a) iterations and (b) training positions.

4.3.3 c_{puct}

The tested values for c_{puct} included 0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, and 16. The training curves are depicted in Figure 25. With an extremely low level of exploration ($c_{puct} = 0.0625$), the AlphaZero algorithm failed to learn to play the game. For $c_{puct} \geq 1$, the speed of learning decreased as c_{puct} increased (higher levels of exploration). In addition, the progress of learning became unstable for larger values of c_{puct}.

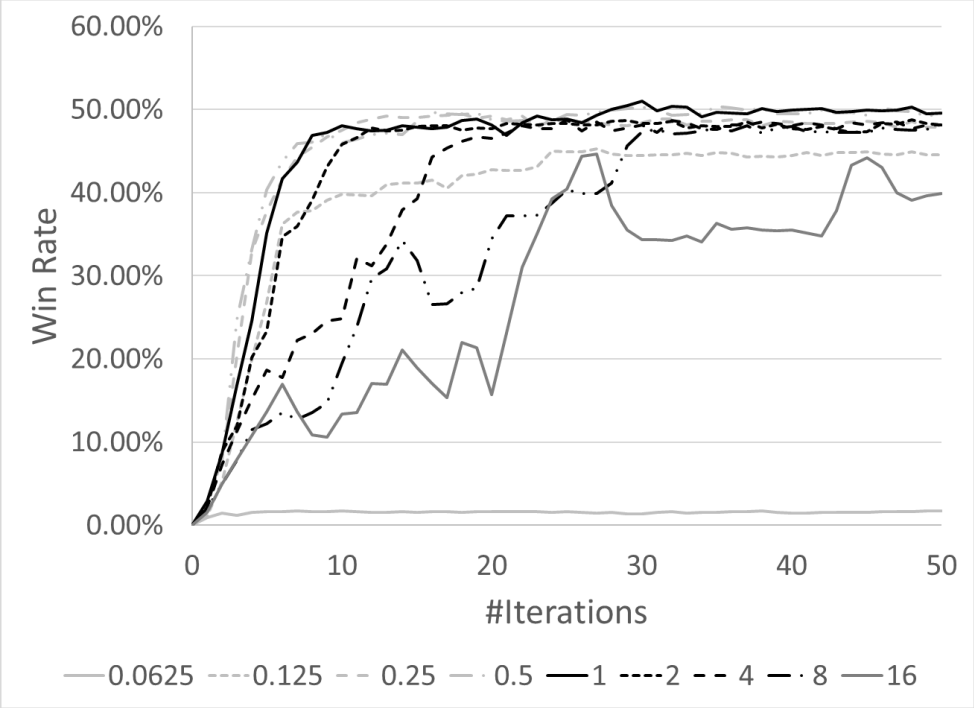


Figure 25: Training curves for different c_{puct}.

The win rates achieved by different c_{puct}'s at the 50th iteration are shown in Figure 26. The 95% confidence intervals are also shown, though they are not clearly visible (about 0.98%). Except for the two extreme ends $c_{puct} = 0.0625$ and $c_{puct} = 16$, the win rates were all close to 50%. Among these settings, $c_{puct} = 0.5$ (49.44±0.98%) and $c_{puct} = 1$ (49.60 ±0.98%) were considered to reach the optimal play.

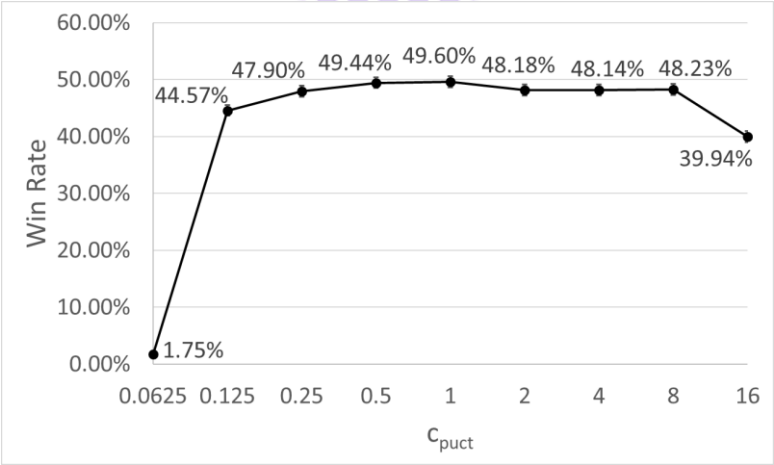


Figure 26: The win rates at the 50th iteration for different c_{puct}.

Figure 27 shows the AEs for different c_{puct}'s, averaged from the last five iterations. The 95% confidence intervals are also depicted in Figure 27. The lowest AE was reached at $c_{puct} = 0.5$ (0.0131±0.0076). The settings of $c_{puct} = 0.125$, $c_{puct} = 0.25$, and $c_{puct} = 1$ also had low AEs. This demonstrated that the algorithm could approximate the theoretical value of the game well. For $c_{puct} \geq 1$, the AE increased significantly as c_{puct} increased.

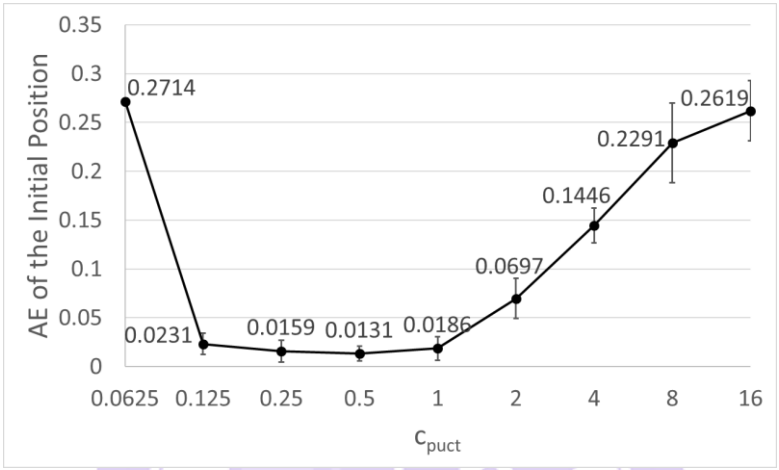


Figure 27: The averaged AEs of the initial position over the last five iterations for different c_{puct}.

The total numbers of distinct seen positions during training for different c_{puct}'s are depicted in Figure 28. As expected, a higher c_{puct} (a higher level of exploration) led to a higher number of seen positions, with only one exception $c_{puct} = 16$.

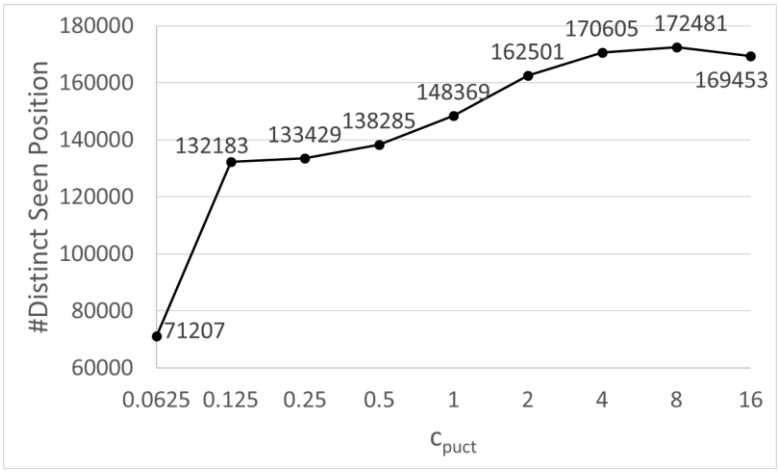


Figure 28: The total numbers of distinct seen positions during training for different c_{puct}.

4.3.4 Dirichlet α

The tested values of Dirichlet α included 0.03, 0.15, 0.3, 0.75, 1.5, 3, 6, 9, and 12. The training curves are not presented since they did not have much difference. Figure 29 shows the win rates achieved by different Dirichlet α's at the 50th iteration. Except for Dirichlet $\alpha = 0.03$, all were considered to reach the optimal play. However, the win rate of Dirichlet $\alpha = 0.03$ (48.44±0.98%) was still near 50%. This showed that Dirichlet α did not influence too much on learning the optimal play.

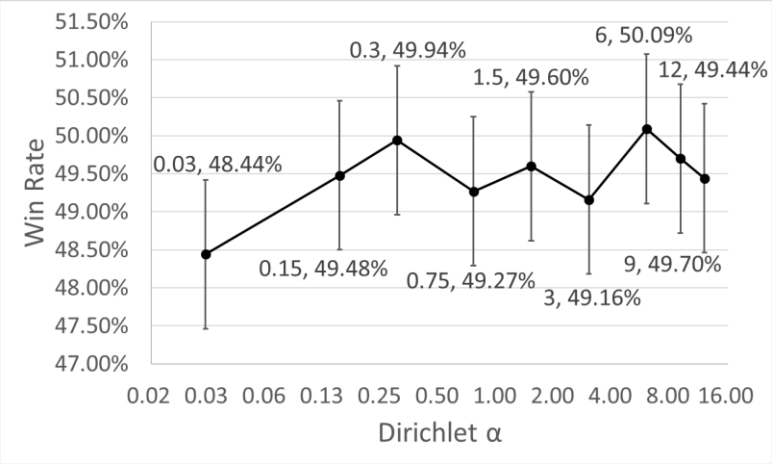


Figure 29: The win rates at the 50th iteration for different Dirichlet α.

The AEs for different Dirichlet α's are depicted in Figure 30. The lowest AE (0.0100±0.0074) was reached at Dirichlet $\alpha = 12$. The settings of 1.5, 3, and 6 also obtained relatively low AEs. Similar to the observations from c_{puct}, higher levels of exploration (lower Dirichlet α) led to higher AEs.

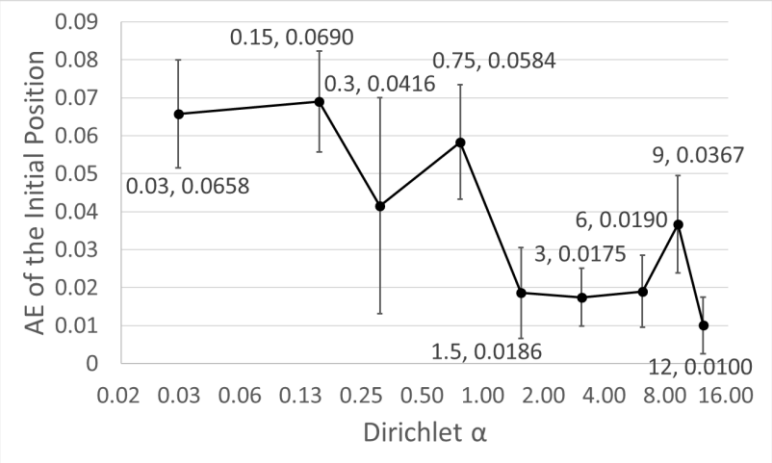


Figure 30: The averaged AEs of the initial position over the last five iterations for different Dirichlet α.

Figure 31 shows the total numbers of distinct seen positions during training for different Dirichlet α's. As expected, the number of seen positions decreased as Dirichlet α increased, though the influence was not as obvious as c_{puct}.

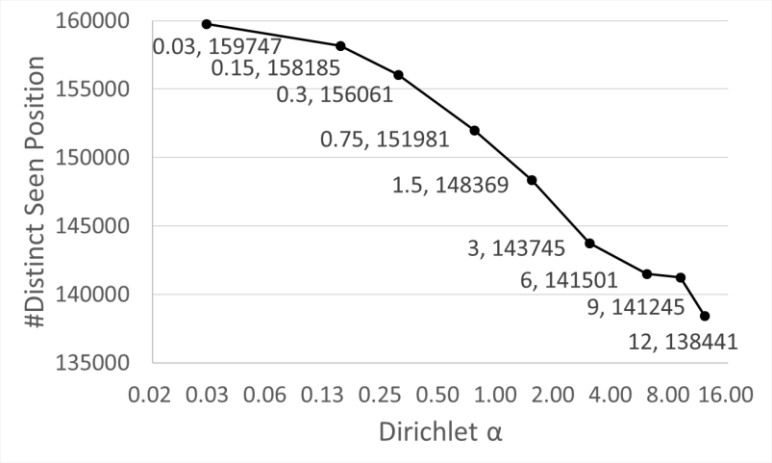


Figure 31: The total numbers of distinct seen positions during training for different Dirichlet α.

4.3.5 Dirichlet ε

The tested values of Dirichlet ε included 0, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, and 0.5, where 0.25 was used in AlphaGo Zero and the AlphaZero algorithm [122, 123]. The training

curves are also omitted since they did not show much difference. Figure 32 depicts the win rates achieved by different Dirichlet ε's at the 50th iteration. The range between 0.2 to 0.4 all reached the optimal play. The win rate was the lowest (46.93±0.98%) when the Dirichlet noises were ignored (Dirichlet $\varepsilon = 0$), which showed that the Dirichlet noises did help the learning.

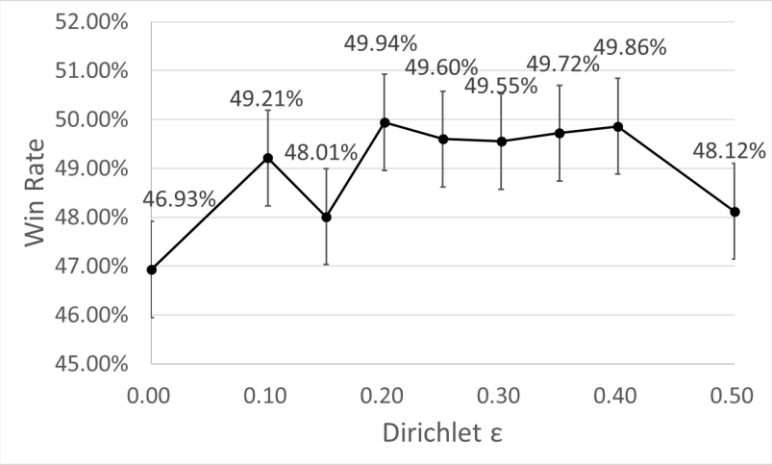


Figure 32: The win rates at the 50th iteration for different Dirichlet ε.

The AEs for different Dirichlet ε's are shown in Figure 33. Similar to previous observations, higher levels of exploration (higher Dirichlet ε) led to higher AEs. As for the total numbers of distinct seen positions during training, depicted in Figure 34, the results were as expected. Namely, the number of seen positions increased as Dirichlet ε increased.

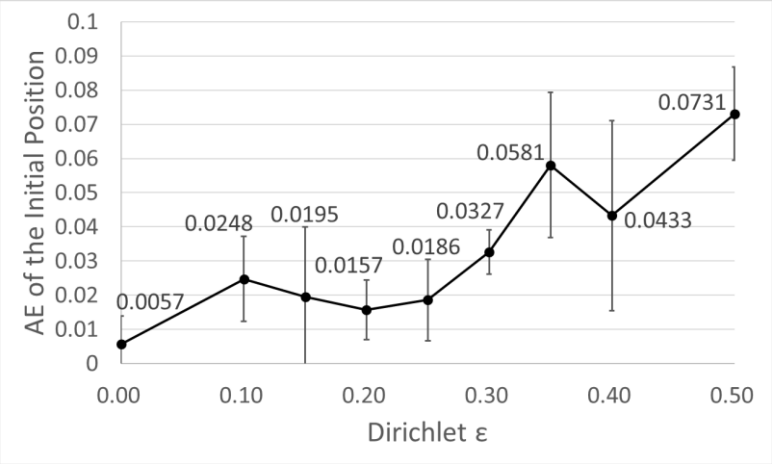


Figure 33: The averaged AEs of the initial position over the last five iterations for different Dirichlet ε.

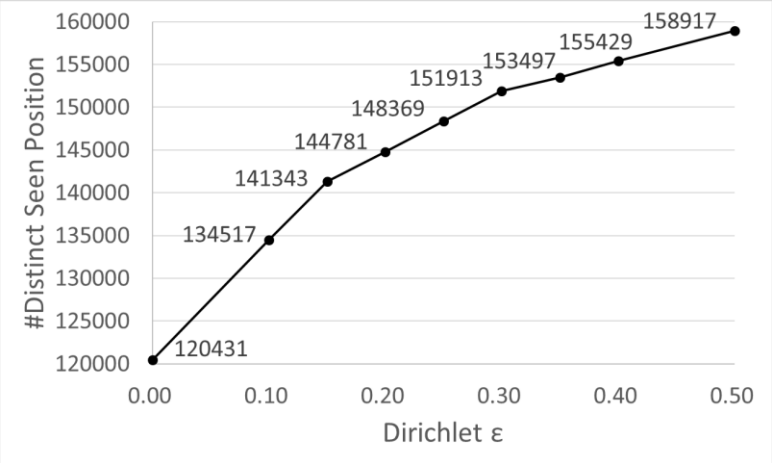


Figure 34: The total numbers of distinct seen positions during training for different Dirichlet ε.

4.3.6 Temperature τ

The tested values for temperature τ included 0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, and 16. The training curves in Figure 35 showed that for temperature $\tau \geq 1$, the speed of learning decreased as temperature τ increased (higher levels of exploration). The win rates achieved by different temperature τ's at the 50th iteration are shown in Figure 36. All of the win rates were close to 50%, while only temperature $\tau = 1$ (49.60±0.98%) was considered to reach the optimal play.

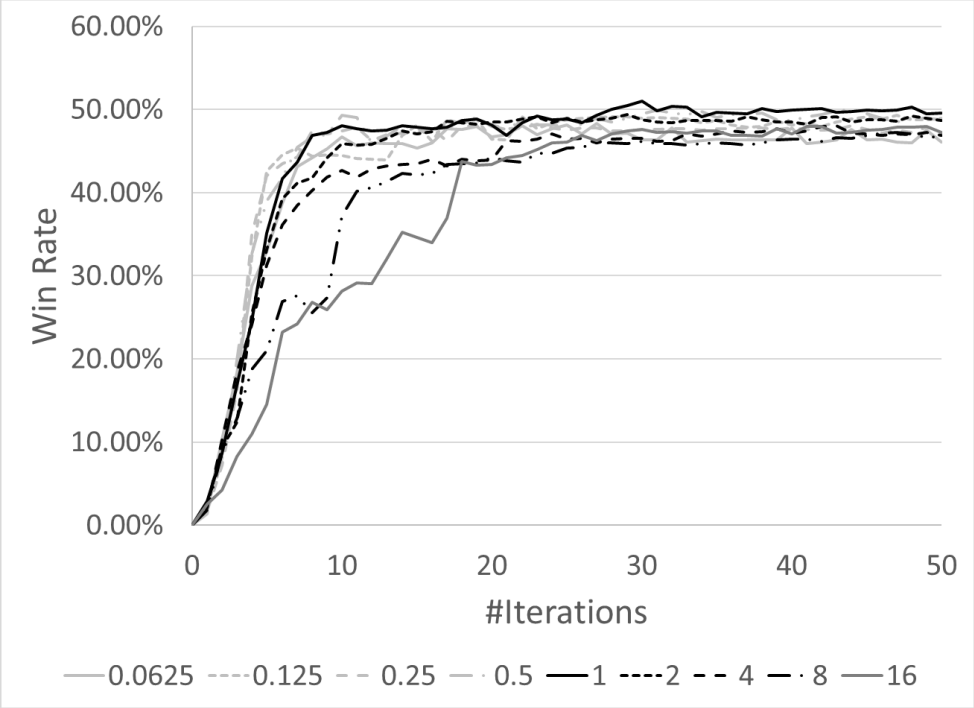


Figure 35: Training curves for different temperature τ.

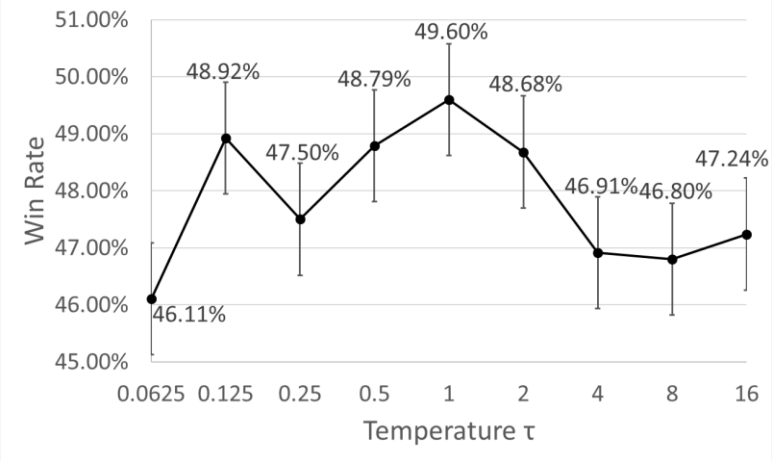


Figure 36: The win rates at the 50th iteration for different temperature τ.

Figure 37 depicts the AEs for different temperature τ's. Similarly, with higher levels of exploration (higher temperature τ), the AEs became higher. The results on the total numbers of distinct seen positions during training, shown in Figure 38, had similar tendency as c_{puct}. Namely, when the level of exploration was too high, the number of seen positions decreased within the same amount of training games.

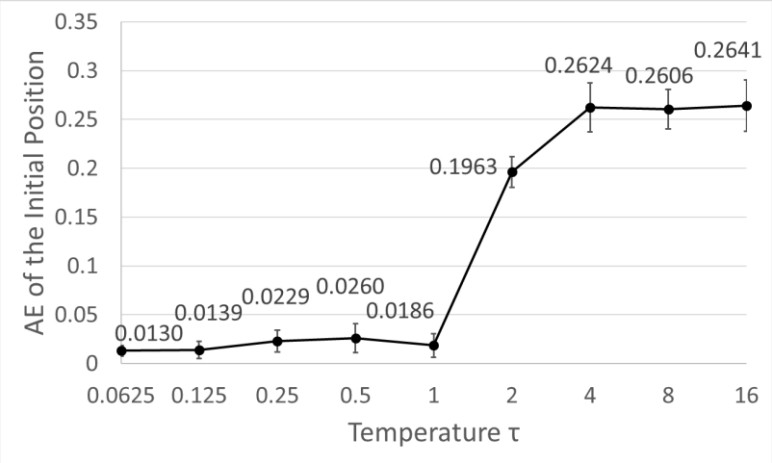


Figure 37: The averaged AEs of the initial position over the last five iterations for different temperature τ.

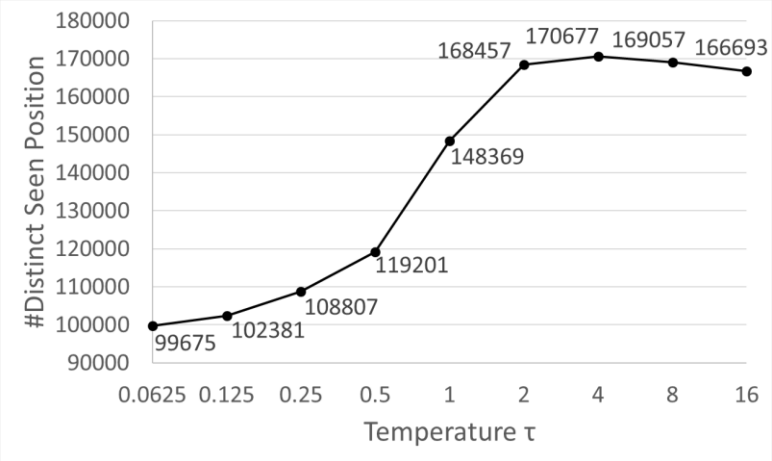


Figure 38: The total numbers of distinct seen positions during training for different temperature τ.

4.3.7 Discussions on Four Tested Hyper-parameters

In general, the AlphaZero algorithm learned the theoretical value and the optimal play in the material combination "PPPP vs. pppp," unless extreme values of hyper-parameters were used. From Figure 27, Figure 30, Figure 33, and Figure 37, many of the low AEs indicated that the values derived from the algorithm approximated to the theoretical value. From Figure 26, Figure 29, Figure 32, and Figure 36, many of these win rates were near 50% against the optimal

play. The results demonstrated the robustness of the algorithm.

The four tested hyper-parameters were all related to exploration. Higher c_{puct}'s, Dirichlet ε's, and temperature τ's, or lower Dirichlet α's led to higher levels of exploration. Comparing the total numbers of distinct seen positions during training (Figure 28, Figure 31, Figure 34, and Figure 38), c_{puct} and temperature τ had more influence on the algorithm than Dirichlet α and Dirichlet ε. However, a consistent tendency was observed in these four hyper-parameters. The AEs tended to increase when the levels of exploration increased beyond some threshold (Figure 27, Figure 30, Figure 33, and Figure 37).

In addition, for c_{puct} and temperature τ, when the levels of exploration were too high, the number of seen positions decreased (Figure 28 and Figure 38). In these two hyper-parameters, it was also observed in Figure 25 and Figure 35 that the learning speeds decreased as the levels of exploration increased beyond some threshold (i.e., $c_{puct} \geq 1$ and temperature $\tau \geq 1$).

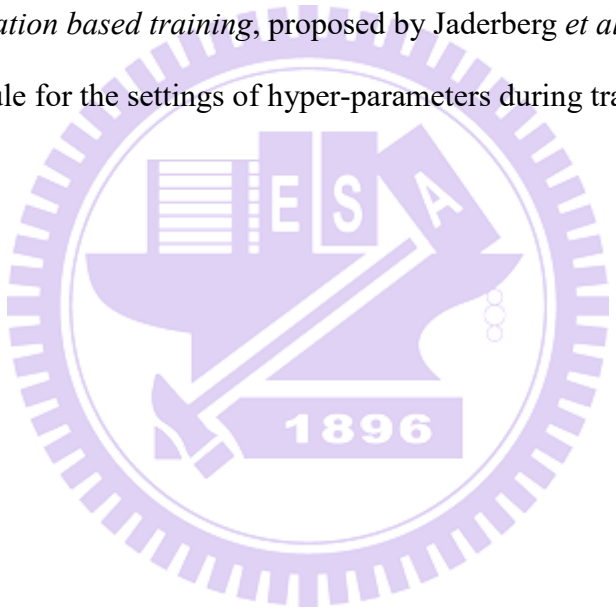
The result in Figure 32 (Dirichlet $\varepsilon = 0$) showed that the AlphaZero algorithm was benefited from Dirichlet noises, though the effect was small. Possible explanations were that the influence of the Dirichlet noises was bounded by c_{puct}, and that the average number of legal moves in 2×4 CDC was too small (~4).

4.4 Chapter Conclusions

This chapter applies the AlphaZero algorithm to a solved stochastic game 2×4 CDC. Symmetries of positions and four hyper-parameters related to exploration in the algorithm are experimented. The results in the material combination "`PPPP` vs. `pppp`" demonstrate that the AlphaZero algorithm generally learns the theoretical value and the optimal play.

For future work, promising directions include the followings. The first is to conduct

experiments on more complicated material combinations in 2×4 CDC to verify whether the AlphaZero algorithm is also able to learn theoretical values and optimal plays well. In addition, experiments can be done to analyze whether the good settings of the hyper-parameters in this thesis is applicable to other material combinations. The second is to replace lookup tables by feature extractors such as neural networks and study whether the algorithm produces consistent results. This may provide insights on whether the algorithm can be incorporated to solve games. The third is to investigate the influence of the hyper-parameters to the algorithm more thoroughly, and then compare the values selected by Bayesian optimization. The fourth is to incorporated the *population based training*, proposed by Jaderberg *et al.* [66], to automatically find out a better schedule for the settings of hyper-parameters during training.



Chapter 5 More Stochastic Games

This chapter investigates two more stochastic games with perfect information, which are EinStein Würfelt Nicht! (EWN) [1, 34, 84, 100, 128] and 2048-like games [36, 68, 88, 98, 125, 131, 155, 159]. Temporal difference (TD) learning [129, 130], a classical reinforcement learning algorithm, is incorporated into the two games to learn feature weights represented by n-tuple networks [6, 67, 68, 74, 87, 88, 96, 98, 109, 131, 135, 136, 142, 155]. The learned feature weights are further combined to tree search algorithms to improve playing strength. More specifically to n-tuple networks, it is a kind of feature extraction method that has been successfully applied to Othello [67, 74, 87, 96, 109, 142], Connect-4 [6, 135, 136], and 2048 [68, 88, 98, 131, 155]. An n-tuple network consists of several n-tuples, where an n-tuple usually covers a part of a board of size n. For each n-tuple, a lookup table keeps the feature weights for all possible configuration of the covered board. Given an n-tuple network, the state-value of a position is calculated by linearly summing up the feature weights from the n-tuples.

The rest of this chapter is organized as follows. Section 5.1 introduces the temporal difference learning algorithm. The game of EWN and 2048-like games are introduced and studied in Sections 5.2 and 5.3 respectively. Finally, Section 5.4 makes concluding remarks of this chapter.

5.1 Temporal Difference Learning

Temporal difference (TD) learning [129, 130] is a kind of reinforcement learning, which learns state values from the subsequent evaluations. It has been successfully applied to many games, such as backgammon [12, 134], checkers [113], chess [12, 147], shogi [13, 14], Go [120], Connect6 [153], Othello [67, 74, 87, 142], Connect-4 [6, 135, 136], Chinese chess [139,

164], and 2048 [68, 88, 98, 131, 155].

In TD learning, the state-value function $V(s)$ is used to approximate the *expected return* of state s (i.e., how many scores the player is expected to further obtain from s to the termination). The simplest form of TD learning, TD(0), updates $V(s_t)$ for state s_t at time step t according to the immediately observed reward r_t and $V(s_{t+1})$ for the next state s_{t+1} by

$$V(s_t) \leftarrow V(s_t) + lr \cdot [r_t + \gamma \cdot V(s_{t+1}) - V(s_t)] \tag{32}$$

where lr is the learning rate and γ the discount rate in the range of $[0,1]$ determining the present value of future rewards.

In the general form, TD(λ), instead of the immediate reward r_t, the *n-step return*

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k \cdot r_{t+k} + \gamma^n \cdot V(s_{t+n}) \tag{33}$$

is used. The first term is the true discounted return from s_t for n steps (or actions). The second term is used to approximate the expected return for the rest of the steps from s_{t+n} to termination. TD(λ) updates $V(s_t)$ by

$$V(s_t) \leftarrow V(s_t) + lr \cdot \left[R_t^{\lambda} - V(s_t)\right] \tag{34}$$

where R_t^{λ} is the *λ-return* defined by

$$R_t^{\lambda} = (1-\lambda) \cdot \sum_{n=1}^{T-t-1} \lambda^{n-1} \cdot R_t^{(n)} + \lambda^{T-t-1} \cdot R_t \tag{35}$$

assuming T is the terminal time step. The R_t in Formula (35) calculated by

$$R_t = \sum_{k=t}^{T} \gamma^{k-t} \cdot r_k \tag{36}$$

which is the true discounted return from s_t to termination (i.e., time step T).

In TD(λ), the range of λ is $[0,1]$. The two extreme ends, $\lambda = 0$ and $\lambda = 1$, are discussed as follows. For $\lambda = 0$, as shown in Formula (32), the update for the state-value only considers one-step return. For $\lambda = 1$, only the true discounted return R_t from s_t to

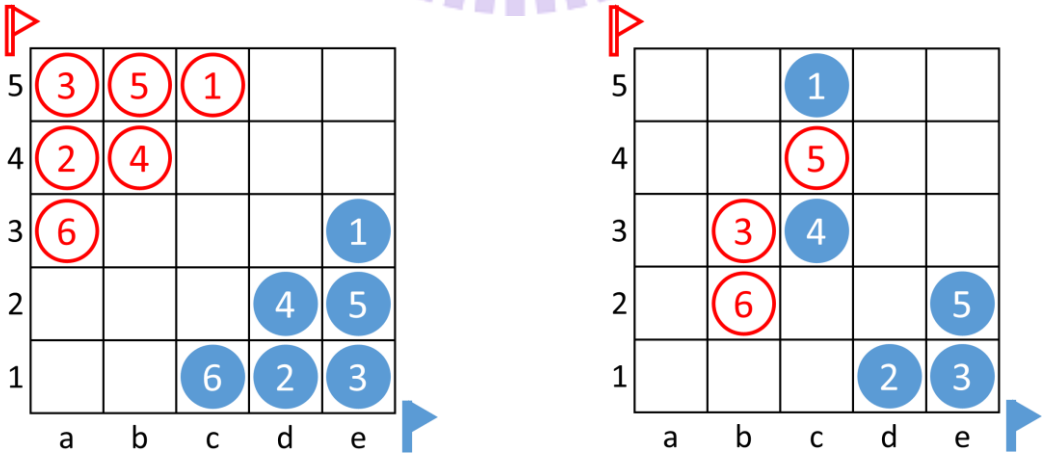
termination is considered. In other words, it only learns based on rewards actually observed. TD(1) is also known as Monte-Carlo algorithm [130].

5.2 Investigation on EinStein Würfelt Nicht!

In this section, the game of EinStein Würfelt Nicht! is introduced in Subsection 5.2.1. Subsections 5.2.2 and 5.2.3 present three techniques incorporating n-tuple networks into MCTS and the results respectively.

5.2.1 EinStein Würfelt Nicht!

EinStein Würfelt Nicht! (EWN) [1, 34, 84, 100, 128] is a two-player zero-sum stochastic board game with perfect information designed by Althöfer in 2004. The game is played with a six-sided dice on a 5×5 square board, as shown in Figure 39 (a). Two players, Red and Blue, own six pieces numbered `1` to `6` respectively. At the beginning of a game, red and blue pieces are placed in the upper-left and lower-right corners respectively. The order of the pieces is determined by rolling dice, and is diagonally symmetric for the two colors for fairness, as shown in Figure 39 (a).



(a) (b)

Figure 39: (a) An initial position and (b) an example of position for EWN.

The two players perform actions in turn, which consisting of rolling the dice and then moving the piece of the rolled number. A piece can be moved one square vertically, horizontally, or diagonally toward the opponent corner. More specifically, red pieces, starting in the upper-left corner, can be moved one square to the right, down, or down-right, and *vice versa* for blue pieces. If the destination of a move contains a piece, the piece is captured and removed from the board regardless of the color. For example, if Red rolls 3 in the position in Figure 39 (b), moving red 3 to c3 captures blue 4 and to b2 captures red 6. When a player does not have the piece with the rolled number, it can move a remaining piece with a number either the next-highest or the next-lowest to the rolled number. For example, in Figure 39 (b), Red only has pieces numbered 3, 5, and 6. If it rolls 4, red 3 or red 5 can be moved instead. A player wins by moving one piece to the farthest opponent corner (Red to e1 and Blue to a5) or capturing all opponent pieces.

5.2.2 Incorporating N-tuple Networks into MCTS

In this subsection, the n-tuple network for EWN contains 24 six-tuples whose shapes are either 2×3 or 3×2 rectangles [34]. The n-tuple network is trained by TD(1) (Monte-Carlo algorithm) with $\gamma = 1$. The learned feature weights are further incorporated into MCTS by three techniques including progressive bias [26, 28, 57], prior knowledge [48, 49], and epsilon-greedy playouts [126].

For progressive bias (Formula (11)), the heuristic score $H(s,a)$ for action a at state s is the state-value after performing a at s. The second technique is prior knowledge [48, 49]. Instead of initializing the visit $N(s,a)$ and the value $Q(s,a)$ of leaves to some default values such as 0, heuristic scores were used to initialize $Q(s,a)$. In addition, the initial visit was set to N_0, a parameter selected by experiments. With the n-tuple network for EWN, the state-value after performing a at s is used to initialize $Q(s,a)$ for prior knowledge.

The third technique is epsilon-greedy playouts [126]. The n-tuple network was used to determine the *best* actions, i.e., the actions leading to states with the highest state-value, during playouts. For each of the playout actions, the best actions were selected with a probability of $(1 - \varepsilon_{EG})$; otherwise, random actions were played. With the n-tuple network for EWN, the best action of a state s is determined by comparing the state-values after performing different actions.

5.2.3 Results

In the experiments, each setting played 2,000 games, with altering the first player to play, against a plain MCTS program with 3,000 simulations per action. The trained n-tuple network (selecting the actions leading to states with the highest state-values) already had a win rate of 42.80% (±2.17%) against the plain MCTS. The win rates were further improved by combining the n-tuple network with MCTS. For progressive bias, the win rate was the highest (53.95±2.18%) with $Count(s,a) = L(s,a)$ and $c_{PB} = 10$. For prior knowledge, the win rate was the highest (54.05±2.18%) with $N_0 = 175$. For epsilon-greedy playouts, the win rate was the highest (61.05±2.14%) with $\varepsilon_{EG} = 0.2$. By combining all the three techniques, the win rate slightly increased to 62.25% (±2.12%). A program named EWIN based on the three techniques won the EWN tournament champion in the Computer Olympiad in 2018 [94].

5.3 Investigation on 2048-Like Games

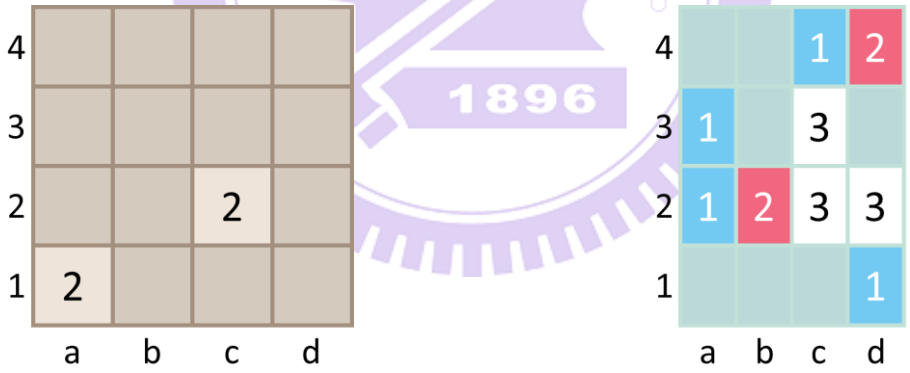
This section introduces 2048-like games in Subsection 5.3.1. Subsections 5.3.2 and 5.3.3 present a variant of TD learning, multistage TD (MS-TD) learning, and the results on 2048-like games respectively.

5.3.1 2048-Like Games

This subsection introduces two 2048-like games, 2048 [36, 68, 88, 98, 131, 155, 159] and Threes [125, 159]. These games are single-player stochastic games with perfect information played on a 4×4 square board. Each square on the board is either empty or placed with a v-tile, which is labeled by a specific number v. 2048 was considered to originate from Threes [124]. However, since the rules of 2048 are easier to explain, this subsection introduces 2048 first in Subsection 5.3.1.1 and then Threes in Subsection 5.3.1.2.

5.3.1.1 2048

The labeling numbers in 2048 are powers of two starting from 2 (i.e., 2, 4, 8, 16, 32, …). In the initial positions in 2048, two tiles, either 2-tiles or 4-tiles, are placed in two squares at random. An example of initial position is shown in Figure 40 (a).



(a) (b)

Figure 40: Examples of initial positions for (a) 2048 and (b) Threes.

In each turn, the player performs an action, which is to choose one of the four directions, up, down, left, and right. For a chosen direction, all tiles on the board move in that direction as far as they can until they reach the border or there is already a tile labeled by different number next to it. When sliding a tile, say v-tile, if the tile next to it is also a v-tile, then the two v-

tiles are merged into a larger tile, $2v$-tile. At the same time, the player gains $2v$ more points in the score. For example, after performing the "left" action in the position in Figure 41 (a), the two 4-tiles at c4 and d4 are merged into an 8-tile at c4, the two 4-tiles at a2 and b2 into an 8-tile at a2, the 2-tile at b1 is moved to a1, and the 2-tile at d3 is moved to c3, as shown in Figure 41 (b). Since two 8-tiles are created, the player gains a total of 16 points for this action. A direction is legal if at least one tile can be moved in that direction. For the position in Figure 41 (a), the "up" action is illegal. After an action is done, the game generates a new 2-tile with a probability of 9/10 or a 4-tile with a probability of 1/10 in an empty square selected randomly. For the same example in Figure 41, a new 2-tile is generated at b2 as shown in Figure 41 (c).

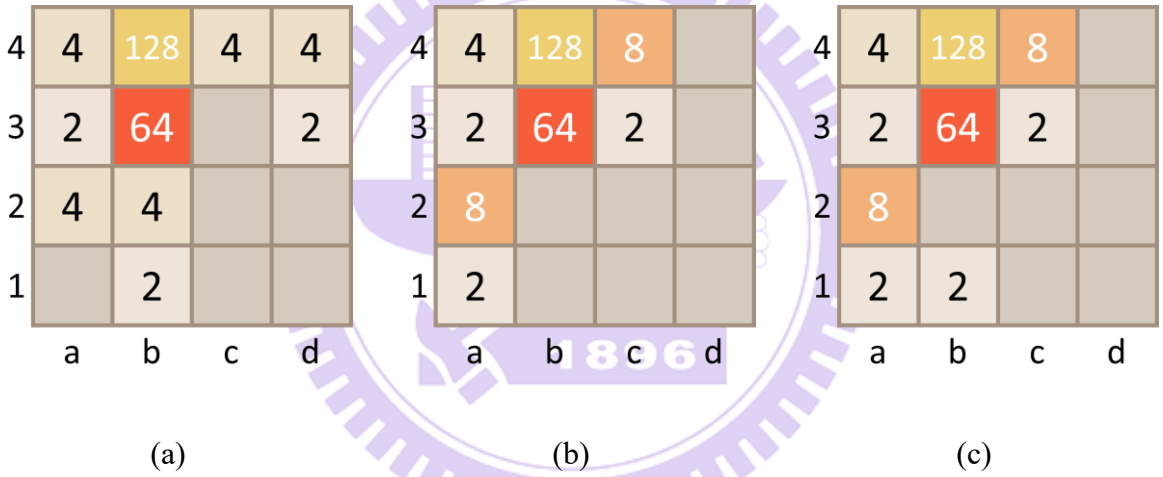


Figure 41: Position transition of an action in 2048, examples of positions (a) to act, (b) after performing the "left" action, and (c) after randomly generating a tile.

A game ends when the player cannot make any legal actions. The final score is the points accumulated during the game. The objective of the game is to accumulate as many points as possible. The game claims that the player wins when a 2048-tile is created, but still allows players to continue playing optionally.

5.3.1.2 Threes

The labeling numbers in Threes are 1, 2, 3, and powers of two times by three up to 6144 (i.e., 1, 2, 3, 6, 12, 24, 48, …, 6144). In the initial positions in Threes, each square is either empty or placed one of the 1-, 2-, and 3-tiles, as shown in Figure 40 (b). The action in each turn also chooses one of the four directions, up, down, left, and right. However, the sliding distance is at most one. For example, the 1-tile at d1 in Figure 40 (b) moves to c1 when performing the "left" action. With regard to the merging rules, a 1-tile and a 2-tile are merged into a 3-tile, while two v-tiles are merged into a $2v$-tile for $3 \leq v < 6144$. Note that tiles are merged when they reach the border or there is already a tile labeled by different number next to it. For example, performing the "right" action in the position in Figure 40 (b) merges the 1-tile at c4 and the 2-tile at d4 into a 3-tile at d4 and the two 3-tiles at c2 and d2 into a 6-tile at d2. The 1-tile at a2 and the 2-tile at b2 are just moved to b2 and c2 respectively without merging.

The rules of generating new tiles are much more complex. Initially, there is a bag of twelve tiles composed of equal amounts of 1-, 2-, and 3-tiles. A *normal* random tile is to sample without replacement from the bag until the bag is empty and then refilled. Let v_{max}-tile denote the tile labeled with the largest number v_{max} on the current board. If $v_{max} \geq 48$, a new tile could be a *bonus* random tile, namely a v-tile with $v > 3$. The probability to generate a bonus random tile is 1/21; otherwise, a normal random tile is generated. A bonus random tile ranges from a 6-tile to a v_{bonus}-tile, where $v_{bonus} = v_{max} \times (1/8)$, with equal probability. One more special rule in Threes is that the game provides hints (1-tile, 2-tile, 3-tile, or a bonus tile) to the next generated tile.

Similarly, a game ends when the player is no longer to perform actions. The final score is calculated based on all the v-tiles on the board for $v \geq 3$. More specifically, the score is $3^{(\log_2(v/3)+1)}$. The objective of the game is to obtain scores as high as possible.

5.3.2 Multistage Temporal Difference Learning

From the games played, it was observed that the plain TD learning hardly reached large tiles such as 32768-tiles even with expectimax search [155, 159]. However, to obtain higher scores, reaching larger tiles is necessary.

Multistage TD (MS-TD) learning, a kind of hierarchical reinforcement learning [10], addresses this issue for 2048-like games. The learning process is divided into multiple stages, each of which has its own learning agent and sub-goal, e.g., reaching 8192-tiles or 16384-tiles for 2048. An example of splitting into three stages for 2048 at T_{16k} and T_{16k+8k} is used in this subsection to explain the method, where T_{16k} denotes the first time in a game when a 16384-tile is created and T_{16k+8k} the first time when both 16384-tile and 8192-tile are created.

In the first stage (before T_{16k}), TD learning is used to train the feature weights for millions of games starting from initial positions. The trained feature weights are used to keep playing games to collect a sufficient number (e.g., 100,000) of positions at T_{16k}. In the second stage (T_{16k} to T_{16k+8k}), the collected positions in the first stage become the initial positions, which are repeatedly used in a round-robin manner. TD learning is then used to train a new set of feature weights for millions of games. Similarly, the second stage collects positions at T_{16k+8k} for the third stage (after T_{16k+8k}), which is then trained in a way similar to the second stage.

When playing games, the process is also split accordingly. For the same example with three stages, feature weights trained in the first stage are used to play before T_{16k}. Similarly, the feature weights trained in the second and the third stages are used to play during T_{16k} to T_{16k+8k} and after T_{16k+8k} respectively.

5.3.3 Results

The n-tuple network for 2048 contains four 6-tuples, two rectangular and two knife-shaped[4], as well as large-tile features [159]. For comparison, the n-tuple network is trained by the original TD learning and MS-TD learning with the three-stage split in Subsection 5.3.2 respectively. TD learning, for both MS-TD and the original TD, in this subsection refers to TD(0) with $\gamma = 1$. MS-TD learning trained five million games in each stage, while TD learning trained a total of fifteen million games for fairness. All the feature weights were initialized to zero. The feature weights were incorporated into expectimax search [155, 159] to further improve the playing strengths. Both with 3-ply expectimax search, MS-TD learning significantly improved the reaching rates of 32768-tiles from 0% to 13.78% (±0.68%) and the average scores from 321,496 (±2,859) to 350,394 (±8,579) compared to TD learning. The reaching rates were obtained from 10,000 games.

The settings of experiments for Threes were similar to those for 2048. For the n-tuple network, some Threes-specific features (e.g., the hint tiles) were included. With regard to the three-stage split, T_{16k} and T_{16k+8k} were replaced by T_{1536} and T_{3072} respectively. With 3-ply expectimax search, MS-TD learning significantly improved the reaching rates of 6144-tiles from 0.45% (±0.13%) to 7.83% (±0.53%) and the average scores from 139,351 (±4,839) to 220,393 (±12,393) compared to TD learning for 10,000 testing games. These results demonstrated the effectiveness of MS-TD learning for 2048-like games.

[4] Oka and Matsuzaki [88, 98] had a thorough investigation on the sizes and the shapes of the tuples. Their results showed that these two kinds of shapes were good designs.

5.4 Chapter Conclusions

Two different kinds of stochastic games with perfect information, EWN and 2048-like games, are studied in this chapter. Among the two games, EWN is a two-player game while 2048-like games are single-player games. TD learning is used to learn weights of features extracted from positions by n-tuple networks.

For EWN, feature weights learned by TD(1) are incorporated into MCTS. By combining three techniques, progressive bias, prior knowledge, and epsilon-greedy playouts, a win rate of 62.25% (±2.12%) is obtained against the plain MCTS.

For 2048-like games, feature weights learned by TD(0) and MS-TD(0) are incorporated into 3-ply expectimax search. In 2048, the reaching rate of 32768-tiles and average score increase from 0% to 13.78% (±0.68%) and 321,496 (±2,859) to 350,394 (±8,579) respectively by applying MS-TD learning. In addition, the reaching rate of 6144-tiles and the average score in Threes increase from 0.45% (±0.13%) to 7.83% (±0.53%) and 139,351 (±4,839) to 220,393 (±12,393) respectively.

Chapter 6 Conclusions and Future Research Directions

In this chapter, the conclusions to the thesis are given in Section 6.1. Several promising future research directions are discussed in Section 6.2.

6.1 Conclusions

This thesis thoroughly investigated Chinese dark chess (CDC) and 2×4 CDC, two kinds of stochastic games with perfect information. To create strong game-playing programs, four existing techniques combing additional knowledge are incorporated into Monte-Carlo tree search (MCTS) to further enhance the strengths. More specifically, the four techniques are early playout terminations (EPT), implicit minimax backups (IMB), quality-based rewards with simulation length (SL) or terminal state quality (TSQ), and progressive bias (PB). The techniques combine manually designed heuristics for the game of CDC to enhance an MCTS-based award winning program named DARKKNIGHT. When playing against the original DARKKNIGHT, the techniques of EPT, IMB, SL, TSQ, and PB, with the best settings of parameters, obtain win rates of 60.75% (±2.76%), 71.85% (±2.42%), 57.50% (±2.77%), 59.00% (±2.78%), and 82.10% (±1.98%). By combining all, the win rate slightly improves to 84.75% (±1.90%). A version of enhanced DARKKNIGHT based on these techniques has won the CDC tournaments in Computer Olympiad during 2015 to 2018.

A question then comes to the definition of strengths. For solved games such as 2×4 CDC, the strengths of developed programs can be measured by directly playing against the optimal player, where the win rate is defined as absolute strengths in this thesis. However, since many real-world games are impractical to solve, other metrics such as win rates playing against another program are considered instead. This thesis studies three commonly used strength

analysis metrics, which are win rates against a baseline program, prediction rates to expert actions, and mean squared errors to values of positions. Experiments show that win rates against baseline programs other than the optimal player have very high correlations to absolute strengths. The correlation grows higher as the baseline program is stronger. Prediction rates and mean squared errors are also shown to have high correlations to absolute strengths. For prediction rates, a lower quality of the collected expert actions harms the correlation a lot. An additional interesting analysis is that the absolute strengths obtained in 2×4 CDC have a high correlation to the win rates in 4×8 CDC except the programs related to PB.

In addition to strength analysis metrics, 2×4 CDC is also used to analyze the AlphaZero algorithm, a kind of reinforcement learning algorithm achieved superhuman levels of plays in chess, shogi, and Go. This thesis applies the algorithm to 2×4 CDC, more specifically, the "`PPPP` vs. `pppp`" combination, and shows that theoretical values and optimal plays can be learned well, even though 2×4 CDC is a stochastic game. This thesis also investigates how the algorithm is influenced by symmetries of positions in training data and four hyper-parameters related to exploration, which are c_{puct}, Dirichlet α, Dirichlet ε, and temperature τ. With regard to symmetries of positions, four settings of symmetries in 2×4 CDC are investigated, which are none, color, positional, and both color and positional symmetries. Experiments show that the version without considering symmetries has a higher win rate under the same numbers of training positions. This demonstrates that the AlphaZero algorithm can generate better training data than augmenting by symmetries. However, the training time (wall clock time) can be significantly reduced by introducing both color and positional symmetries. With regard to the four tested hyper-parameters, generally, the algorithm converged to nearly the theoretical values and the optimal plays of the game in many of the tested settings. This demonstrates the robustness of the AlphaZero algorithm.

Finally, another two kinds of stochastic games with perfect information are studied, which

are EinStein Würfelt Nicht! (EWN) and 2048-like games. In these two games, another kind of reinforcement learning algorithm, TD(λ), is applied to learn feature weights based on n-tuple networks. For EWN, the feature weights are trained by TD(1), or called Monte-Carlo algorithm, and are incorporated into MCTS by three techniques including PB, prior knowledge, and epsilon-greedy playouts. The three techniques with the best settings of parameters obtain win rates of 53.95% (±2.18%), 54.05% (±2.18%), and 61.05% (±2.14%) playing against the original plain MCTS. By combining all, a slightly higher win rate of 62.25% (±2.12%) is obtained. A program named EWIN based on the n-tuple network and the three techniques won the EWN tournament in the 21st Computer Olympiad in 2018.

For 2048-like games, TD(0), the simplest form of TD(λ), is used to learn feature weights. The reaching rates of large tiles and the average scores are significantly enhanced by multistage temporal difference (MS-TD) learning compared to the original TD learning. The experiments show that the 32768-tile reaching rate and the average score in 2048 are improved from 0% to 13.78% (±0.68%) and 321,496 (±2,859) to 350,394 (±8,579) respectively. The 6144-tile reaching rates and the average score in Threes are improved from 0.45% (±0.13%) to 7.83% (±0.53%) and 139,351 (±4,839) to 220,393 (±12,393) respectively.

6.2 Future Research Directions

This section first discusses five promising research directions specific to the AlphaZero algorithm, and then two for overall computer game researches.

Although the AlphaZero algorithm did achieve superhuman levels of plays in chess, shogi, and Go, the results are hard to reproduce mainly due to an extremely high consumption of computing resources. The first research direction is then to develop more efficient ways to learn. The numbers of games required for humans becoming experts from beginners are greatly less

than those for the AlphaZero algorithm. Thus, it is expected to have rooms to improve. The second and the third directions, as mentioned in Chapter 4, are to try other complicated combinations to confirm whether the algorithm still learns theoretical values and optimal plays, and to investigate more thoroughly on the hyper-parameters respectively. In this thesis, the algorithm is shown to work on a toy stochastic game, "`PPPP` vs. `pppp`" in 2×4 CDC, with lookup tables. For further investigation, hyper-parameters can be selected by grid search [103] as this thesis do, Bayesian optimization [117] as Silver *et al*. [122, 123] did, or population based training proposed by Jaderberg *et al.* [66]. The fourth research direction is to apply the algorithm to real-world stochastic games with perfect information such as CDC, EWN, backgammon, and dice-shogi [5]. It is interesting to investigate whether the algorithm also works on larger games with (deep) neural networks. The fifth direction is to apply the algorithm to imperfect information games such as phantom Go and Mahjong, which is expected to be harder than the fourth. It is interesting to investigate how the algorithm deals with opponent modeling.

The first promising research direction for overall computer game researches is to develop *explainable artificial intelligence* (*XAI*) [40]. For games where programs are already much stronger than human players (e.g., Othello [19], chess, and Go), it is even more important to enable the programs to teach human players. The goal of XAI is to make machine learning systems transparent, explainable, interpretable, and trustable by humans, especially for critical real-world applications such as healthcare, finance, and self-driving cars. Even if machine learning systems can make good decisions, they are hardly deployed to real-world unless the rationale behind reaching these decisions are understandable by humans. Applying to computer games, XAI can be developed to explain actions or positions (e.g., why an action is better than others or why a position is advantageous/disadvantageous to some player). This can be further used to teach human players. For board games where an action does not change the position too much (e.g., CDC, EWN, chess, and shogi), n-tuple networks are a possible way. Insights may

be discovered by comparing feature weights of a position and another after one or two actions. For game-playing programs based on deep neural networks, visualization techniques developed for image recognition problems such as Grad-CAM [115] may be tried.

The second promising research direction for computer games is to develop *artificial general intelligence* (AGI) [11], which may require greater efforts than XAI. AGI aims to create a single AI for general purpose. Currently, most of the AIs are for particular purposes (i.e., an AI can do one task well but not the others). For example, AlphaZero trained for chess can only play chess to superhuman level, not shogi or Go, but say nothing of recognizing images or navigating self-driving cars. Developing AGI for computer games may provide insights, which can help the development of real AGI.

References

1. Althöfer, I., *On the Origins of "EinStein würfelt nicht!"*, retrieved Mar. 5, 2019, from: https://althofer.de/origins-of-ewn.html.
2. Althöfer, I., "Hanfried wins EinStein Würfelt Nicht! tournament", ICGA Journal, 38(4), pp. 246, 2015, doi: 10.3233/ICG-2015-38408.
3. Asuero, A. G., Sayago, A., and González, A. G., "The correlation coefficient: An overview", Critical Reviews in Analytical Chemistry, 36(1), pp. 41-59, 2006, doi: 10.1080/10408340500526766.
4. Auer, P., Cesa-Bianchi, N., and Fischer, P., "Finite-time analysis of the multiarmed bandit problem", Machine Learning, 47(2-3), pp. 235-256, 2002, doi: 10.1023/A:1013689704352.
5. Baba, T. and Ito, T. "Improvement and evaluation of search algorithm in computer dice-shogi", in Proceedings of 2017 Conference on Technologies and Applications of Artificial Intelligence (TAAI 2017), Taipei, Taiwan, Dec. 1-3, 2017, IEEE, pp. 84-87, doi: 10.1109/TAAI.2017.49.
6. Bagheri, S., Thill, M., Koch, P., and Konen, W., "Online adaptable learning rates for the game Connect-4", IEEE Transactions on Computational Intelligence and AI in Games, 8(1), pp. 33-42, 2016, doi: 10.1109/TCIAIG.2014.2367105.
7. Baier, H., Sattaur, A., Powley, E. J., Devlin, S., Rollason, J., and Cowling, P. I., "Emulating human play in a leading mobile card game", IEEE Transactions on Games, in press, doi: 10.1109/TG.2018.2835764.
8. Baier, H. and Winands, M. H. M. "Monte-Carlo tree search and minimax hybrids with heuristic evaluation functions", in Proceedings of The 8th International Conference on Computers and Games (CG 2013), Yokohama, Japan, Aug. 13-15, 2013, Cazenave, T., Winands, M. H. M., and Björnsson, Y. (Eds.), Springer International Publishing, pp. 45-63, doi: 10.1007/978-3-319-14923-3_4.
9. Baier, H. and Winands, M. H. M., "Time management for Monte Carlo tree search", IEEE Transactions on Computational Intelligence and AI in Games, 8(3), pp. 301-314, 2016, doi: 10.1109/TCIAIG.2015.2443123.
10. Barto, A. G. and Mahadevan, S., "Recent advances in hierarchical reinforcement learning", Discrete Event Dynamic Systems, 13(1-2), pp. 41-77, 2003, doi: 10.1023/A:1022140919877.
11. Baum, S. D., "A survey of artificial general intelligence projects for ethics, risk, and policy", Global Catastrophic Risk Institute Working Paper 17-1, pp. 1-99, 2017, doi:

10.2139/ssrn.3070741.

12. Baxter, J., Tridgell, A., and Weaver, L., "Learning to play chess using temporal differences", Machine Learning, 40(3), pp. 243-263, 2000, doi: 10.1023/A:1007634325138.
13. Beal, D. F. and Smith, M. C. "First results from using temporal difference learning in shogi", in Proceedings of The 1st Conference on Computers and Games (CG'98), Tsukuba, Japan, Nov. 11-12, 1998, van den Herik, H. J. and Iida, H. (Eds.), Springer Berlin Heidelberg, pp. 113-125, doi: 10.1007/3-540-48957-6_7.
14. Beal, D. F. and Smith, M. C., "Temporal difference learning applied to game playing and the results of application to shogi", Theoretical Computer Science, 252(1), pp. 105-119, 2001, doi: 10.1016/S0304-3975(00)00078-5.
15. Bishop, C. M., Pattern Recognition and Machine Learning, 1st edition, Springer-Verlag New York, 738 pages, 2006.
16. Bjornsson, Y. and Finnsson, H., "CadiaPlayer: A simulation-based general game player", IEEE Transactions on Computational Intelligence and AI in Games, 1(1), pp. 4-15, 2009, doi: 10.1109/TCIAIG.2009.2018702.
17. Borsboom, J., Saito, J.-T., Chaslot, G. M. J. B., and Uiterwijk, J. W. H. M. "A comparison of Monte-Carlo methods for phantom Go", in The 19th Belgian-Dutch Conference on Artificial Intelligence (BNAIC 2007), Utrecht, the Netherlands, Nov. 5-6, 2008 of Conference.
18. Browne, C. B., Powley, E. J., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., . . . Colton, S., "A survey of Monte Carlo tree search methods", IEEE Transactions on Computational Intelligence and AI in Games, 4(1), pp. 1-43, 2012, doi: 10.1109/TCIAIG.2012.2186810.
19. Buro, M., "The Othello match of the year: Takeshi Murakami vs. Logistello", ICCA Journal, 20(3), pp. 189-193, 1997, doi: ICG-1997-20311.
20. Campbell, M., Hoane, A. J., and Hsu, F.-h., "Deep Blue", Artificial Intelligence, 134(1), pp. 57-83, 2002, doi: 10.1016/S0004-3702(01)00129-1.
21. Cazenave, T. "A phantom-Go program", in Proceedings of The 11th International Conference on Advances in Computer Games (ACG 2005), Taipei, Taiwan, Sep. 6-9, 2005, van den Herik, H. J., Hsu, S.-C., Hsu, T.-s., and Donkers, H. H. L. M. (Eds.), Springer Berlin Heidelberg, pp. 120-125, doi: 10.1007/11922155_9.
22. Chang, H.-J., Chen, J.-C., Hsueh, C.-W., and Hsu, T.-s., "Analysis and efficient solutions for 2×4 Chinese dark chess", ICGA Journal, 40(2), pp. 61-76, 2018, doi: 10.3233/ICG-180049.
23. Chang, H.-J., Fan, G.-Y., Chen, J.-C., Hsueh, C.-W., and Hsu, T.-s. "Validating and fine-tuning of game evaluating functions using endgame databases", in Proceedings of The

6th Workshop on Computer Games (CGW 2017), Held in Conjunction with the 26th International Conference on Artificial Intelligence (IJCAI-17), Melbourne, Australia, Aug. 20, 2017, Cazenave, T., Winands, M. H. M., and Saffidine, A. (Eds.), Springer International Publishing, pp. 137-150, doi: 10.1007/978-3-319-75931-9_10.

24. Chang, H.-J. and Hsu, T.-s. "A quantitative study of 2×4 Chinese dark chess", in Proceedings of The 8th International Conference on Computers and Games (CG 2013), Yokohama, Japan, Aug. 13-15, 2013, van den Herik, H. J., Iida, H., and Plaat, A. (Eds.), Springer International Publishing, pp. 151-162, doi: 10.1007/978-3-319-09165-5_13.
25. Chang, H.-J., Hsueh, C.-W., and Hsu, T.-s. "Convergence and correctness analysis of Monte-Carlo tree search algorithms: A case study of 2 by 4 Chinese dark chess", in Proceedings of 2015 IEEE Conference on Computational Intelligence and Games (CIG 2015), Tainan, Taiwan, Aug. 31-Sep. 2, 2015, IEEE, pp. 260-266, doi: 10.1109/CIG.2015.7317963.
26. Chaslot, G. M. J. B., "Monte Carlo tree search", Department of Knowledge Engineering, Universiteit Maastricht, Ph.D. dissertation, 144 pages, 2010, Maastricht, the Netherlands.
27. Chaslot, G. M. J. B., Fiter, C., Hoock, J.-B., Rimmel, A., and Teytaud, O. "Adding expert knowledge and exploration in Monte-Carlo tree search", in Proceedings of The 12th International Conference on Advances in Computer Games (ACG 2009), Pamplona, Spain, May 11-13, 2009, van den Herik, H. J. and Spronck, P. (Eds.), Springer Berlin Heidelberg, pp. 1-13, doi: 10.1007/978-3-642-12993-3_1.
28. Chaslot, G. M. J. B., Winands, M. H. M., van den Herik, H. J., Uiterwijk, J. W. H. M., and Bouzy, B., "Progressive strategies for Monte Carlo tree search", New Mathematics and Natural Computation, 4(3), pp. 343-357, 2008, doi: 10.1142/S1793005708001094.
29. Chen, B.-N. and Hsu, T.-s. "Automatic generation of opening books for dark chess", in Proceedings of The 8th International Conference on Computers and Games (CG 2013), Yokohama, Japan, Aug. 13-15, 2013, van den Herik, H. J., Iida, H., and Plaat, A. (Eds.), Springer International Publishing, pp. 221-232, doi: 10.1007/978-3-319-09165-5_19.
30. Chen, B.-N., Shen, B.-J., and Hsu, T.-s., "Chinese dark chess", ICGA Journal, 33(2), pp. 93-106, 2010, doi: 10.3233/ICG-2010-33204.
31. Chen, J.-C., Fan, G.-Y., Chang, H.-J., and Hsu, T.-s., "Compressing Chinese dark chess endgame databases by deep learning", IEEE Transactions on Games, 10(4), pp. 413-422, 2018, doi: 10.1109/TG.2018.2802484.
32. Chen, J.-C., Lin, T.-Y., Chen, B.-N., and Hsu, T.-s., "Equivalence classes in Chinese dark chess endgames", IEEE Transactions on Computational Intelligence and AI in Games, 7(2), pp. 109-122, 2015, doi: 10.1109/TCIAIG.2014.2317832.
33. Chen, J.-C., Lin, T.-Y., Hsu, S.-C., and Hsu, T.-s. "Design and implementation of

computer Chinese dark chess endgame database", in TCGA Workshop 2012, Hualien, Taiwan, Jun. 30-Jul. 1, 2012 of Conference. (Chinese)

34. Chu, Y.-J. R., Chen, Y.-H., Hsueh, C.-H., and Wu, I.-C. "An agent for EinStein Würfelt Nicht! using n-tuple networks", in Proceedings of 2017 Conference on Technologies and Applications of Artificial Intelligence (TAAI 2017), Taipei, Taiwan, Dec. 1-3, 2017, IEEE, pp. 184-189, doi: 10.1109/TAAI.2017.32.
35. Ciancarini, P. and Favini, G. P., "Monte Carlo tree search in Kriegspiel", Artificial Intelligence, 174(11), pp. 670-684, 2010, doi: 10.1016/j.artint.2010.04.017.
36. Cirulli, G., *2048*, retrieved Mar. 5, 2019, from: https://play2048.co/.
37. Coulom, R. "Computing elo ratings of move patterns in the game of Go", in Proceedings of Computer Games Workshop 2007 (CGW 2007), Amsterdam, the Netherlands, Jun. 15-17, 2007, van den Herik, H. J., Uiterwijk, J. W. H. M., Winands, M. H. M., and Schadd, M. P. D. (Eds.), pp. 113-124.
38. Cowling, P. I., Powley, E. J., and Whitehouse, D., "Information set Monte Carlo tree search", IEEE Transactions on Computational Intelligence and AI in Games, 4(2), pp. 120-143, 2012, doi: 10.1109/TCIAIG.2012.2200894.
39. Demediuk, S., Tamassia, M., Raffe, W. L., Zambetta, F., Li, X., and Mueller, F. "Monte Carlo tree search based algorithms for dynamic difficulty adjustment", in Proceedings of 2017 IEEE Conference on Computational Intelligence and Games (CIG 2017), New York, USA, Aug. 22-25, 2017, IEEE, pp. 53-59, doi: 10.1109/CIG.2017.8080415.
40. Došilović, F. K., Brčić, M., and Hlupić, N. "Explainable artificial intelligence: A survey", in Proceedings of The 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO 2018), Opatija, Croatia, May 21-25, 2018, Skala, K., Koricic, M., Galinac Grbac, T., Cicin-Sain, M., Sruk, V., Ribaric, S., . . . Janjic, M. (Eds.), IEEE, pp. 210-215, doi: 10.23919/MIPRO.2018.8400040.
41. Enzenberger, M. and Müller, M. "A lock-free multithreaded Monte-Carlo tree search algorithm", in Proceedings of The 12th International Conference on Advances in Computer Games (ACG 2009), Pamplona, Spain, May 11-13, 2009, van den Herik, H. J. and Spronck, P. (Eds.), Springer Berlin Heidelberg, pp. 14-20, doi: 10.1007/978-3-642-12993-3_2.
42. Enzenberger, M., Müller, M., Arneson, B., and Segal, R., "Fuego—An open-source framework for board games and Go engine based on Monte Carlo tree search", IEEE Transactions on Computational Intelligence and AI in Games, 2(4), pp. 259-270, 2010, doi: 10.1109/TCIAIG.2010.2083662.
43. Facebook, *ELF | Game Research Platform | Facebook AI*, retrieved Mar. 5, 2019, from: https://facebook.ai/developers/tools/elf.

44. Ferguson, T. S., “A Bayesian analysis of some nonparametric problems”, The Annals of Statistics, 1(2), pp. 209-230, 1973, doi: 10.1214/aos/1176342360.
45. Finnsson, H. “Generalized Monte Carlo tree search extensions for general game playing”, in Proceedings of The 26th AAAI Conference on Artificial Intelligence (AAAI-12), Toronto, Canada, Jul. 22-26, 2012, AAAI Press, pp. 1550-1556.
46. Gao, C., Hayward, R. B., and Müller, M., “Move prediction using deep convolutional neural networks in Hex”, IEEE Transactions on Games, 10(4), pp. 336-343, 2018, doi: 10.1109/TG.2017.2785042.
47. Gao, C., Müller, M., and Hayward, R. B. “Focused depth-first proof number search using convolutional neural networks for the game of Hex”, in Proceedings of The 26th International Joint Conference on Artificial Intelligence (IJCAI-17), Melbourne, Australia, Aug. 19-25, 2017, Sierra, C. (Ed.), IJCAI Organization, pp. 3668-3674, doi: 10.24963/ijcai.2017/513.
48. Gelly, S. and Silver, D. “Combining online and offline knowledge in UCT”, in Proceedings of The 24th International Conference on Machine Learning (ICML '07), Corvallis, Oregon, USA, Jun. 20-24, 2007, Ghahramani, Z. (Ed.), ACM, pp. 273-280, doi: 10.1145/1273496.1273531.
49. Gelly, S. and Silver, D., “Monte-Carlo tree search and rapid action value estimation in computer Go”, Artificial Intelligence, 175(11), pp. 1856-1875, 2011, doi: 10.1016/j.artint.2011.03.007.
50. Gelly, S., Wang, Y., Munos, R., and Teytaud, O., “Modification of UCT with patterns in Monte-Carlo Go”, Technical Report: RR-6062, INRIA, France, 2006.
51. Han, J. and Moraga, C. “The influence of the sigmoid function parameters on the speed of backpropagation learning”, in Proceedings of The International Workshop on Artificial Neural Networks (IWANN '95): From Natural to Artificial Neural Computation, Torremolinos, Spain, Jun. 7-9, 1995, Mira, J. and Sandoval, F. (Eds.), Springer Berlin Heidelberg, pp. 195-201, doi: 10.1007/3-540-59497-3_175.
52. Henderson, P., Arneson, B., and Hayward, R. B. “Solving 8×8 Hex”, in Proceedings of The 21st International Joint Conference on Artificial Intelligence (IJCAI-09), Pasadena, California, USA, Jul. 11-17, 2009, IJCAI Organization, pp. 505-510.
53. Hsueh, C.-H. and Wu, I.-C., “DarkKnight wins Chinese dark chess tournament”, ICGA Journal, 38(4), pp. 249-251, 2015, doi: 10.3233/ICG-2015-38411.
54. Hsueh, C.-H., Wu, I.-C., Chen, J.-C., and Hsu, T.-s. “AlphaZero for a non-deterministic game”, in Proceedings of 2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI 2018), Taichung, Taiwan, Nov. 30-Dec. 2, 2018, IEEE, pp. 116-121.
55. Hsueh, C.-H., Wu, I.-C., Hsu, T.-s., and Chen, J.-C., “An investigation of strength

analysis metrics for game-playing programs: A case study in Chinese dark chess", ICGA Journal, 40(2), pp. 77-104, 2018, doi: 10.3233/ICG-180046.

56. Hsueh, C.-H., Wu, I.-C., Tseng, W.-J., Yen, S.-J., and Chen, J.-C. "Strength improvement and snalysis for an MCTS-based Chinese dark chess program", in Proceedings of The 14th International Conference on Advances in Computer Games (ACG 2015), Leiden, the Netherlands, Jul. 1-3, 2015, Plaat, A., van den Herik, H. J., and Kosters, W. (Eds.), Springer International Publishing, pp. 29-40, doi: 10.1007/978-3-319-27992-3_4.
57. Hsueh, C.-H., Wu, I.-C., Tseng, W.-J., Yen, S.-J., and Chen, J.-C., "An analysis for strength improvement of an MCTS-based program playing Chinese dark chess", Theoretical Computer Science, 644, pp. 63-75, 2016, doi: 10.1016/j.tcs.2016.06.025.
58. Huang, S.-C., "New heuristics for Monte Carlo tree search applied to the game of Go", Department of Computer Science and Information Engineering, National Taiwan Normal University, Ph.D. dissertation, 91 pages, 2011, Taipei, Taiwan.
59. Hunicke, R. and Chapman, V. "AI for dynamic difficulty adjustment in games", in Proceedings of AAAI-04 workshop on Challenges in Game Artificial Intelligence, San Jose, California, USA, Jul. 25-26, 2004, AAAI Press, pp. 91-96.
60. ICGA, *Results (Computer Olympiad 2016) | ICGA*, retrieved Mar. 5, 2019, from: https://icga.org/?page_id=1764.
61. ICGA, *Results (Computer Olympiad 2017) | ICGA*, retrieved Mar. 5, 2019, from: https://icga.org/?page_id=2050.
62. Ikeda, K., Shishido, T., and Viennot, S. "Machine-learning of shape names for the game of Go", in Proceedings of The 14th International Conference on Advances in Computer Games (ACG 2015), Leiden, the Netherlands, Jul. 1-3, 2015, Plaat, A., van den Herik, H. J., and Kosters, W. (Eds.), Springer International Publishing, pp. 247-259, doi: 10.1007/978-3-319-27992-3_22.
63. Ikeda, K. and Viennot, S. "Efficiency of static knowledge bias in Monte-Carlo tree search", in Proceedings of The 8th International Conference on Computers and Games (CG 2013), Yokohama, Japan, Aug. 13-15, 2013, van den Herik, H. J., Iida, H., and Plaat, A. (Eds.), Springer International Publishing, pp. 26-38, doi: 10.1007/978-3-319-09165-5_3.
64. Ikeda, K. and Viennot, S. "Production of various strategies and position control for Monte-Carlo Go — Entertaining human players", in Proceedings of 2013 IEEE Conference on Computational Inteligence in Games (CIG 2013), Niagara Falls, ON, Canada, Aug. 11-13, 2013, IEEE, pp. 145-152, doi: 10.1109/CIG.2013.6633625.
65. Ishihara, M., Ito, S., Ishii, R., Harada, T., and Thawonmas, R. "Monte-Carlo tree search for implementation of dynamic difficulty adjustment fighting game AIs having

believable behaviors", in Proceedings of 2018 IEEE Conference on Computational Intelligence and Games (CIG 2018), Maastricht, the Netherlands, Aug. 14-17, 2018, IEEE, pp. 46-53, doi: 10.1109/CIG.2018.8490376.

66. Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., . . . Kavukcuoglu, K., "Population based training of neural networks", CoRR, abs/1711.09846, 2017.
67. Jaśkowski, W., "Systematic n-tuple networks for Othello position evaluation", ICGA Journal, 37(2), pp. 85-96, 2014, doi: 10.3233/ICG-2014-37203.
68. Jaśkowski, W., "Mastering 2048 with delayed temporal coherence learning, multistage weight promotion, redundant encoding, and carousel shaping", IEEE Transactions on Games, 10(1), pp. 3-14, 2018, doi: 10.1109/TCIAIG.2017.2651887.
69. Jouandeau, N. "Varying complexity in CHINESE DARK CHESS stochastic game", in TCGA Workshop 2014, Taipei, Taiwan, Jun. 27-28, 2014 of Conference.
70. Jouandeau, N. and Cazenave, T. "Monte-Carlo tree reductions for stochastic games", in Proceedings of 2014 Conference on Technologies and Applications of Artificial Intelligence (TAAI 2014), Taipei, Taiwan, Nov. 21-23, 2014, Cheng, S.-M. and Day, M.-Y. (Eds.), Springer International Publishing, pp. 228-238, doi: 10.1007/978-3-319-13987-6_22.
71. Jouandeau, N. and Cazenave, T. "Small and Large MCTS Playouts Applied to Chinese Dark Chess Stochastic Game", in Proceedings of The 3rd Workshop on Computer Games (CGW 2014), Held in Conjunction with the 21st European Conference on Artificial Intelligence (ECAI 2014), Prague, Czech Republic, Aug. 18, 2014, Cazenave, T., Winands, M. H. M., and Björnsson, Y. (Eds.), Springer International Publishing, pp. 78-89, doi: 10.1007/978-3-319-14923-3_6.
72. Kishimoto, A. and Müller, M., "Game solvers", in Handbook of Digital Games and Entertainment Technologies (Ch. 1), Nakatsu, R., Rauterberg, M., and Ciancarini, P. (Eds), Springer Singapore, Singapore, pp. 3-22, 2015, doi: 10.1007/978-981-4560-52-8_35-1.
73. Kocsis, L. and Szepesvári, C. "Bandit based Monte-Carlo planning", in Proceedings of The 17th European Conference on Machine Learning (ECML 2006), Berlin, Germany, Sep. 18-22, 2006, Fürnkranz, J., Scheffer, T., and Spiliopoulou, M. (Eds.), Springer-Verlag Berlin Heidelberg, pp. 282-293, doi: 10.1007/11871842_29.
74. Krawiec, K. and Szubert, M. G. "Learning n-tuple networks for Othello by coevolutionary gradient search", in Proceedings of The 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11), Dublin, Ireland, Jul. 12-16, 2011, Krasnogor, N. (Ed.), ACM, pp. 355-362, doi: 10.1145/2001576.2001626.
75. Lai, M., "Giraffe: Using deep reinforcement learning to play chess", Department of

Computing, Imperial College London, Master dissertation, 39 pages, 2015, London, UK.
76. Lanctot, M., Winands, M. H. M., Pepels, T., and Sturtevant, N. R. "Monte Carlo tree search with heuristic evaluations using implicit minimax backups", in Proceedings of 2014 IEEE Conference on Computational Intelligence and Games (CIG 2014), Dortmund, Germany, Aug. 26-29, 2014, IEEE, doi: 10.1109/CIG.2014.6932903.
77. Lane, D. M., Scott, D., Hebl, M., Guerra, R., Osherson, D., and Zimmer, H., Introduction to Statistics, ed. Lane, D. M., Rice University, Houston, Texas, USA, 692 pages, 2014.
78. Leela Zero, *Leela Zero - GitHub*, retrieved Mar. 5, 2019, from: https://github.com/leela-zero.
79. Liang, H.-M., "Study on the property of double king dark chess", Department of Computer Science and Information Engineering, National Dong Hwa University, Master dissertation, 56 pages, 2014, Hualien, Taiwan. (Chinese)
80. Lin, S.-S., Chen, C.-H., Hsu, S.-C., Wu, I.-C., Yen, S.-J., and Chen, J.-C., "TCGA 2014 Computer Game Tournament", ICGA Journal, 37(4), pp. 226-229, 2014, doi: 10.3233/ICG-2014-37408.
81. Lin, Y.-S., Wu, I.-C., and Yen, S.-J., "TAAI 2011 computer-game tournaments", ICGA Journal, 34(4), pp. 248-250, 2011, doi: 10.3233/ICG-2011-34413.
82. Liskowski, P., Jaśkowski, W., and Krawiec, K., "Learning to play Othello with deep neural networks", IEEE Transactions on Games, 10(4), pp. 354-364, 2018, doi: 10.1109/TG.2018.2799997.
83. Lorentz, R. J. "Amazons discover Monte-Carlo", in Proceedings of The 6th International Conference on Computers and Games (CG 2008), Beijing, China, Sep. 29-Oct. 1, 2008, van den Herik, H. J., Xu, X., Ma, Z., and Winands, M. H. M. (Eds.), Springer Berlin Heidelberg, pp. 13-24, doi: 10.1007/978-3-540-87608-3_2.
84. Lorentz, R. J. "An MCTS program to play EinStein Würfelt Nicht!", in Proceedings of The 13th International Conference on Advances in Computer Games (ACG 2011), Tilburg, the Netherlands, Nov. 20-22, 2011, van den Herik, H. J. and Plaat, A. (Eds.), Springer Berlin Heidelberg, pp. 52-59, doi: 10.1007/978-3-642-31866-5_5.
85. Lorentz, R. J., "Using evaluation functions in Monte-Carlo tree search", Theoretical Computer Science, 644, pp. 106-113, 2016, doi: 10.1016/j.tcs.2016.06.026.
86. Lorentz, R. J. and Horey, T. "Programming Breakthrough", in Proceedings of The 8th International Conference on Computers and Games (CG 2013), Yokohama, Japan, Aug. 13-15, 2014, van den Herik, H. J., Iida, H., and Plaat, A. (Eds.), Springer International Publishing, pp. 49-59, doi: 10.1007/978-3-319-09165-5_5.
87. Lucas, S. M., "Learning to play Othello with n-tuple systems", Australian Journal of Intelligent Information Processing, 4, pp. 1-20, 2008.

88. Matsuzaki, K. “Systematic selection of n-tuple networks with consideration of interinfluence for game 2048”, in Proceedings of 2016 Conference on Technologies and Applications of Artificial Intelligence (TAAI 2016), Hsinchu, Taiwan Nov. 25-27, 2016, IEEE, pp. 186-193, doi: 10.1109/TAAI.2016.7880154.
89. Mei, S., Montanari, A., and Nguyen, P.-M., “A mean field view of the landscape of two-layer neural networks”, Proceedings of the National Academy of Sciences of the United States of America, 115(33), pp. E7665-E7671, 2018, doi: 10.1073/pnas.1806579115.
90. Nakamichi, T. and Ito, T., “Adjusting the evaluation function for weakening the competency level of a computer shogi program”, ICGA Journal, 40(1), pp. 15-31, 2018, doi: 10.3233/ICG-180042.
91. NDHU Artificial Intelligence Laboratory, *TAAI Computer Game Tournaments*, retrieved Mar. 5, 2019, from: https://www.tcga.tw/taai2015/zh_TW/#tabout.
92. NDHU Artificial Intelligence Laboratory, *TAAI Computer Game Tournaments 2017*, retrieved Mar. 5, 2019, from: https://www.tcga.tw/taai2017/zh_TW/#tabout.
93. NDHU Artificial Intelligence Laboratory, *TCGA Computer Game Tournaments 2017*, retrieved Mar. 5, 2019, from: https://www.tcga.tw/tournament2017/en/#ranklist.
94. NDHU Artificial Intelligence Laboratory, *ICGA Computer Olympiad 2018*, retrieved Mar. 5, 2019, from: https://www.tcga.tw/icga-computer-olympiad-2018/en/.
95. NDHU Artificial Intelligence Laboratory, *TAAI Computer Game Tournaments 2018*, retrieved Mar. 5, 2019, from: https://www.tcga.tw/taai2018/en/#ranklist.
96. Nguyen, H., Viennot, S., and Ikeda, K. “Fast optimization of the pattern shapes in board games with simulated annealing”, in Proceedings of The 6th International Conference on Knowledge and Systems Engineering (KSE 2014), Hanoi, Vietnam, Oct. 9-11, 2014, Nguyen, V.-H., Le, A.-C., and Huynh, V.-N. (Eds.), Springer International Publishing, pp. 325-337, doi: 10.1007/978-3-319-11680-8_26.
97. Nijssen, J. A. M. and Winands, M. H. M. “Enhancements for multi-player Monte-Carlo tree search”, in Proceedings of The 7th International Conference on Computers and Games (CG 2010), Kanazawa, Japan, Sep. 24-26, 2010, van den Herik, H. J., Iida, H., and Plaat, A. (Eds.), Springer Berlin Heidelberg, pp. 238-249, doi: 10.1007/978-3-642-17928-0_22.
98. Oka, K. and Matsuzaki, K. “Systematic selection of n-tuple networks for 2048”, in Proceedings of The 9th International Conference on Computers and Games (CG 2016), Leiden, the Netherlands, Jun. 29-Jul. 1, 2016, Plaat, A., Kosters, W., and van den Herik, H. J. (Eds.), Springer International Publishing, pp. 81-92, doi: 10.1007/978-3-319-50935-8_8.
99. Osborne, M. J. and Rubinstein, A., A Course in Game Theory, 1st edition, MIT Press, Cambridge, Massachusetts, USA, 368 pages, 1994.

100. Oster, E., “An MCTS agent for EinStein Würfelt Nicht!”, Department of Knowledge Engineering, Maastricht University, Master dissertation, 41 pages, 2015, Maastricht, the Netherlands.
101. Paulsen, P. and Fürnkranz, J. “A moderately successful attempt to train chess evaluation functions of different strengths”, in The ICML-10 Workshop on Machine Learning and Games, Haifa, Israel, Jun. 25, 2010 of Conference, Thurau, C., Driessens, K., and Missura, O. (Eds.).
102. Pawlewicz, J. and Hayward, R. B. “Scalable parallel DFPN search”, in Proceedings of The 8th International Conference on Computers and Games (CG 2013), Yokohama, Japan, Aug. 13-15, 2014, van den Herik, H. J., Iida, H., and Plaat, A. (Eds.), Springer International Publishing, pp. 138-150, doi: 10.1007/978-3-319-09165-5_12.
103. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, É., “Scikit-learn: Machine learning in Python”, Journal of Machine Learning Research, 12(Oct), pp. 2825-2830, 2011.
104. Pepels, T., Tak, M. J. W., Lanctot, M., and Winands, M. H. M. “Quality-based rewards for Monte-Carlo tree search simulations”, in Proceedings of The 21st European Conference on Artificial Intelligence (ECAI 2014), Prague, Czech Republic, Aug. 18-22, 2014, Schaub, T., Friedrich, G., and O'Sullivan, B. (Eds.), IOS Press, pp. 705-710, doi: 10.3233/978-1-61499-419-0-705.
105. Ponsen, M., Gerritsen, G., and Chaslot, G. M. J. B. “Integrating opponent models with Monte-Carlo tree search in poker”, in Proceedings of AAAI-10 Workshop on Interactive Decision Theory and Game Theory, Atlanta, Georgia, USA, Jul. 12, 2010, AAAI Press, pp. 37-42.
106. Richards, M. and Amir, E. “Opponent modeling in Scrabble”, in Proceedings of The 20th International Joint Conference on Artificial Intelligence (IJCAI-07), Hyderabad, India, 2007, IJCAI Organization, pp. 1482-1487.
107. Robbins, H. and Monro, S., “A stochastic approximation method”, The Annals of Mathematical Statistics, 22(3), pp. 400-407, 1951, doi: 10.1214/aoms/1177729586.
108. Rosin, C. D., “Multi-armed bandits with episode context”, Annals of Mathematics and Artificial Intelligence, 61(3), pp. 203-230, 2011, doi: 10.1007/s10472-011-9258-6.
109. Runarsson, T. P. and Lucas, S. M., “Preference learning for move prediction and evaluation function approximation in Othello”, IEEE Transactions on Computational Intelligence and AI in Games, 6(3), pp. 300-313, 2014, doi: 10.1109/TCIAIG.2014.2307272.
110. Russell, S. and Norvig, P., Artificail Intelligence: A Modern Approach, 3rd edition, Pearson, 1152 pages, 2009.
111. Saffidine, A., Jouandeau, N., Buron, C., and Cazenave, T. “Material symmetry to

partition endgame tables", in Proceedings of The 8th International Conference on Computers and Games (CG 2013), Yokohama, Japan, Aug. 13-15, 2014, van den Herik, H. J., Iida, H., and Plaat, A. (Eds.), Springer International Publishing, pp. 187-198, doi: 10.1007/978-3-319-09165-5_16.

112. Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., . . . Sutphen, S., "Checkers is solved", Science, 317(5844), pp. 1518-1522, 2007, doi: 10.1126/science.1144079.
113. Schaeffer, J., Hlynka, M., and Jussila, V. "Temporal difference learning applied to a high-performance game-playing program", in Proceedings of The 17th International Joint Conference on Artificial Intelligence (IJCAI-01), Seattle, Washington, USA, Aug. 4-10, 2001, IJCAI Organization, pp. 529-534.
114. Schaeffer, J. and van den Herik, H. J., "Games, computers, and artificial intelligence", Artificial Intelligence, 134(1-2), pp. 1-7, 2002, doi: 10.1016/S0004-3702(01)00165-5.
115. Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. "Grad-CAM: Visual explanations from deep networks via gradient-based localization", in Proceedings of 2017 IEEE International Conference on Computer Vision (ICCV 2017), Venice, Italy, Oct. 22-29, 2017, IEEE, pp. 618-626, doi: 10.1109/ICCV.2017.74.
116. Sephton, N., Cowling, P. I., and Slaven, N. H. "An experimental study of action selection mechanisms to create an entertaining opponent", in Proceedings of 2015 IEEE Conference on Computational Intelligence and Games (CIG 2015), Tainan, Taiwan, Aug. 31-Sep. 2, 2015, IEEE, pp. 122-129, doi: 10.1109/CIG.2015.7317939.
117. Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N., "Taking the human out of the loop: A review of Bayesian optimization", Proceedings of the IEEE, 104(1), pp. 148-175, 2016, doi: 10.1109/JPROC.2015.2494218.
118. Shapley, L. S., "Stochastic games", Proceedings of the National Academy of Sciences, 39(10), pp. 1095-1100, 1953, doi: 10.1073/pnas.39.10.1095.
119. Silva, M. P., Silva, V. d. N., and Chaimowicz, L. "Dynamic difficulty adjustment through an adaptive AI", in Proceedings of The 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames 2015), Teresina, Piauí, Brazil, Nov. 11-13, 2015, Rodrigues, M. A. F., de Carvalho, F. G., and de Vasconcellos, M. S. (Eds.), IEEE, pp. 173-182, doi: 10.1109/SBGames.2015.16.
120. Silver, D., "Reinforcement Learning and Simulation-Based Search in Computer Go", Department of Computing Science, University of Alberta, Ph.D. dissertation, 158 pages, 2009, Edmonton, Alberta, Canada.
121. Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., . . . Hassabis, D., "Mastering the game of Go with deep neural networks and tree search", Nature, 529(7587), pp. 484-489, 2016, doi: 10.1038/nature16961.

122. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., . . . Hassabis, D., “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”, Science, 352(6419), pp. 1140-1144, 2018, doi: 10.1126/science.aar6404.
123. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., . . . Hassabis, D., “Mastering the game of Go without human knowledge”, Nature, 550(7676), pp. 354-359, 2017, doi: 10.1038/nature24270.
124. Sirvo, *The Rip-offs & Making Our Original Game*, retrieved Mar. 5, 2019, from: http://asherv.com/threes/threemails/.
125. Sirvo, *THREES-A tiny puzzle that grows on you*, retrieved Mar. 5, 2019, from: http://asherv.com/threes/.
126. Sturtevant, N. R. “An analysis of UCT in multi-player games”, in Proceedings of The 6th International Conference on Computers and Games (CG 2008), Beijing, China, Sep. 29-Oct.1, 2008, van den Herik, H. J., Xu, X., Ma, Z., and Winands, M. H. M. (Eds.), Springer Berlin Heidelberg, pp. 37-49, doi: 10.1007/978-3-540-87608-3_4.
127. Su, T.-C., Yen, S.-J., Chen, J.-C., and Wu, I.-C., “TAAI 2012 computer game tournaments”, ICGA Journal, 37(1), pp. 33-35, 2014, doi: 10.3233/ICG-2014-37107.
128. Sun, H., Shao, F., and Li, S. “Design and implementation of the EinStein Würfelt Nicht system based on multi attributes evaluation”, in Proceedings of The 2015 International Conference on Electromechanical Control Technology and Transportation (ICECTT 2015), Zhuhai City, Guangdong Province, China, Oct. 31-Nov. 1, 2015, Atlantis Press, pp. 324-331, doi: 10.2991/icectt-15.2015.62.
129. Sutton, R. S., “Learning to predict by the methods of temporal differences”, Machine Learning, 3(1), pp. 9-44, 1988, doi: 10.1007/BF00115009.
130. Sutton, R. S. and Barto, A. G., Reinforcement Learning: An Introduction, 2nd edition, MIT Press, Cambridge, Massachusetts, USA, 552 pages, 2018.
131. Szubert, M. G. and Jaśkowski, W. “Temporal difference learning of N-tuple networks for the game 2048”, in Proceedings of 2014 IEEE Conference on Computational Intelligence and Games (CIG 2014), Dortmund, Germany, Aug. 26-29, 2014, IEEE, doi: 10.1109/CIG.2014.6932907.
132. Takeuchi, S., Kaneko, T., and Yamaguchi, K., “Evaluation of game tree search methods by game records”, IEEE Transactions on Computational Intelligence and AI in Games, 2(4), pp. 288-302, 2010, doi: 10.1109/TCIAIG.2010.2102022.
133. Tensorflow, *tensorflow/minigo: An open-source implementation of the AlphaGoZero algorithm*, retrieved Mar. 5, 2019, from: https://github.com/tensorflow/minigo.
134. Tesauro, G., “Temporal difference learning and TD-Gammon”, Communications of the ACM, 38(3), pp. 58-68, 1995, doi: 10.1145/203330.203343.

135. Thill, M., Bagheri, S., Koch, P., and Konen, W. “Temporal difference learning with eligibility traces for the game connect four”, in Proceedings of 2014 IEEE Conference on Computational Intelligence and Games, Dortmund, Germany, Aug. 26-29, 2014, IEEE, doi: 10.1109/CIG.2014.6932870.
136. Thill, M., Koch, P., and Konen, W. “Reinforcement learning with n-tuples on the game Connect-4”, in Proceedings of The 12th International Conference on Parallel Problem Solving from Nature (PPSN 2012), Taormina, Italy, Sep. 1-5, 2012, Coello, C. A. C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., and Pavone, M. (Eds.), Springer Berlin Heidelberg, pp. 184-194, doi: 10.1007/978-3-642-32937-1_19.
137. Thompson, K., “Retrograde analysis of certain endgames”, ICCA Journal, 9(3), pp. 131-139, 1986.
138. Tian, Y. and Zhu, Y. “Better computer Go player with neural network and long-term prediction”, in The 4th International Conference on Learning Representations (ICLR 2016), San Juan, Puerto Rico, May 2-4, May 2-4, 2016 of Conference.
139. Trinh, T. B., Bashi, A. S., and Deshpande, N. “Temporal difference learning in Chinese Chess”, in Proceedings of The 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA-98-AIE), Benicàssim, Castellón, Spain, Jun. 1-4, 1998, del Pobil, A. P., Mira, J., and Ali, M. (Eds.), Springer Berlin Heidelberg, pp. 612-618, doi: 10.1007/3-540-64574-8_447.
140. Tseng, W.-J., Chen, J.-C., Chen, L.-P., Yen, S.-J., and Wu, I.-C., “TCGA 2013 computer game tournaments”, ICGA Journal, 36(3), pp. 166-168, 2013, doi: 10.3233/ICG-2013-36310.
141. Tseng, W.-J., Chen, J.-C., and Wu, I.-C., “DarkKnight wins Chinese dark chess tournament”, ICGA Journal, 39(2), pp. 163-165, 2017, doi: 10.3233/ICG-170023.
142. van den Dries, S. and Wiering, M. A., “Neural-fitted TD-leaf learning for playing Othello with structured neural networks”, IEEE Transactions on Neural Networks and Learning Systems, 23(11), pp. 1701-1713, 2012, doi: 10.1109/TNNLS.2012.2210559.
143. van den Herik, H. J., Iida, H., Plaat, A., and Hellemons, J., “The brain and mind sports Olympiad”, ICGA Journal, 36(3), pp. 172, 2013, doi: 10.3233/ICG-2013-36313.
144. van den Herik, H. J., Plaat, A., and Hellemons, J., “The 16th Computer Olympiad”, ICGA Journal, 35(1), pp. 50, 2012, doi: 10.3233/ICG-2012-35111.
145. van den Herik, H. J., Uiterwijk, J. W. H. M., and van Rijswijck, J., “Games solved: Now and in the future”, Artificial Intelligence, 134(1-2), pp. 277-311, 2002, doi: 10.1016/S0004-3702(01)00152-7.
146. Van Lishout, F., Chaslot, G. M. J. B., and Uiterwijk, J. W. H. M. “Monte-Carlo tree search in backgammon”, in Proceedings of Computer Games Workshop 2007 (CGW 2007), Amsterdam, the Netherlands, Jun. 15-17, 2007, van den Herik, H. J., Uiterwijk,

J. W. H. M., Winands, M. H. M., and Schadd, M. P. D. (Eds.), pp. 175-184.

147. Veness, J., Silver, D., Uther, W., and Blair, A. "Bootstrapping from game tree search", in Proceedings of The 22nd International Conference on Neural Information Processing Systems (NIPS 2009), Vancouver, British Columbia, Canada, 2009, Bengio, Y., Schuurmans, D., Williams, C. K. I., and Culotta, A. (Eds.), NIPS Foundation, Inc., pp. 1937-1945.

148. Wang, J., Zhu, T., Li, H., Hsueh, C.-H., and Wu, I.-C. "Belief-state Monte-Carlo tree search for phantom games", in Proceedings of 2015 IEEE Conference on Computational Intelligence and Games (CIG 2015), Tainan, Taiwan, Aug. 31-Sep. 2, 2015, IEEE, pp. 267-274, doi: 10.1109/CIG.2015.7317917.

149. Wang, J., Zhu, T., Li, H., Hsueh, C.-H., and Wu, I.-C., "Belief-state Monte Carlo tree search for phantom Go", IEEE Transactions on Games, 10(2), pp. 139-154, 2018, doi: 10.1109/TCIAIG.2017.2734067.

150. Wikipedia, *Tabletop game - Wikipedia*, retrieved Mar. 5, 2019, from: https://en.wikipedia.org/wiki/Tabletop_game.

151. Winands, M. H. M., Björnsson, Y., and Saito, J.-T. "Monte-Carlo tree search solver", in Proceedings of The 6th International Conference on Computers and Games (CG 2008), Beijing, China, Sep. 29-Oct. 1, 2008, van den Herik, H. J., Xu, X., Ma, Z., and Winands, M. H. M. (Eds.), Springer Berlin Heidelberg, pp. 25-36, doi: 10.1007/978-3-540-87608-3_3.

152. Winands, M. H. M., Björnsson, Y., and Saito, J.-T., "Monte Carlo tree search in lines of action", IEEE Transactions on Computational Intelligence and AI in Games, 2(4), pp. 239-250, 2010, doi: 10.1109/TCIAIG.2010.2061050.

153. Wu, I.-C., Tsai, H.-T., Lin, H.-H., Lin, Y.-S., Chang, C.-M., and Lin, P.-H. "Temporal difference learning for Connect6", in Proceedings of The 13th International Conference on Advances in Computer Games (ACG 2011), Tilburg, the Netherlands, Nov. 20-22, 2011, van den Herik, H. J. and Plaat, A. (Eds.), Springer Berlin Heidelberg, pp. 121-133, doi: 10.1007/978-3-642-31866-5_11.

154. Wu, I.-C., Wu, T.-R., Liu, A.-J., Guei, H., and Wei, T. h. "On strength adjustment for MCTS-based programs", in Proceedings of The 33rd AAAI Conference on Artificial Intelligence (AAAI-19), Honolulu, Hawaii, USA, Jan. 27-Feb. 1, 2019, AAAI Press.

155. Wu, I.-C., Yeh, K.-H., Liang, C.-C., Chang, C.-C., and Chiang, H. "Multi-stage temporal difference learning for 2048", in Proceedings of 2014 Conference on Technologies and Applications of Artificial Intelligence (TAAI 2014), Taipei, Taiwan, Nov. 21-23, 2014, Cheng, S.-M. and Day, M.-Y. (Eds.), Springer International Publishing, pp. 366-378, doi: 10.1007/978-3-319-13987-6_34.

156. Wu, T.-R., Wu, I.-C., Chen, G.-W., Wei, T. h., Wu, H.-C., Lai, T.-Y., and Lan, L.-C.,

"Multilabeled value networks for computer Go", IEEE Transactions on Games, 10(4), pp. 378-389, 2018, doi: 10.1109/TG.2018.2852806.

157. Yang, J.-K., Su, T.-C., and Wu, I.-C., "TCGA 2012 computer games tournament", ICGA Journal, 35(3), pp. 178-180, 2012, doi: 10.3233/ICG-2012-35305.

158. Yannakakis, G. N. and Hallam, J. "Evolving opponents for interesting interactive computer games", in Proceedings of The 8th International Conference on the Simulation of Adaptive Behavior, From Animals to Animats 8 (SAB 2004), Los Angeles, California, USA, Jul. 13-17, 2004, Schaal, S., Ijspeert, A. J., Billard, A., Vijayakumar, S., Hallam, J., and Meyer, J.-A. (Eds.), MIT Press, pp. 499-508.

159. Yeh, K.-H., Wu, I.-C., Hsueh, C.-H., Chang, C.-C., Liang, C.-C., and Chiang, H., "Multistage temporal difference learning for 2048-like games", IEEE Transactions on Computational Intelligence and AI in Games, 9(4), pp. 369-380, 2017, doi: 10.1109/TCIAIG.2016.2593710.

160. Yen, S.-J., Chen, J.-C., Chen, B.-N., and Tseng, W.-J., "DarkKnight wins Chinese dark chess tournament", ICGA Journal, 36(3), pp. 175-176, 2013, doi: 10.3233/ICG-2013-36315.

161. Yen, S.-J., Chiu, S.-Y., and Wu, I.-C., "MoDark wins the Chinese dark chess tournament", ICGA Journal, 33(4), pp. 230-231, 2010, doi: 10.3233/ICG-2010-33410.

162. Yen, S.-J., Chou, C.-W., Chen, J.-C., Wu, I.-C., and Kao, K.-Y., "Design and implementation of Chinese dark chess programs", IEEE Transactions on Computational Intelligence and AI in Games, 7(1), pp. 66-74, 2015, doi: 10.1109/TCIAIG.2014.2329034.

163. Yen, S.-J., Su, T.-C., and Wu, I.-C., "The TCGA 2011 computer-games tournament", ICGA Journal, 34(2), pp. 108-110, 2011, doi: 10.3233/ICG-2011-34216.

164. Yin, H.-F. and Fu, T.-T. "Applying temporal difference learning to acquire a high-performance position evaluation function", in Proceedings of The 7th International Conference on Computer Science & Education (ICCSE 2012), Melbourne, Australia, Jul. 14-17, 2012, IEEE, pp. 80-84, doi: 10.1109/ICCSE.2012.6295031.

165. Zohaib, M., "Dynamic difficulty adjustment (DDA) in computer games: A review", Advances in Human-Computer Interaction, 2018, pp. 1-12, 2018, doi: 10.1155/2018/5681652.

Appendix A. Rules for Early Playout Terminations

This section describes six manually designed rules used in Subsection 2.4.1 to judge whether a material combination has a likely outcome (win, loss, or draw). Without loss of generality, the rules are illustrated from Red's view. Totally, six rules, two for likely draws and four for likely wins/losses, are presented as follows.

Draw Rule 1: Cannons and Pawns

The games usually draws when Red only has at least one cannon and n king where n is either 0 or 1, while Black has at least one piece and the piece(s) are either pawns ($\geq n$) or cannons (≥ 0). Examples includes `Cc`, `Cp`, `CCccp`, `CCppp`, `Ccpp`, `KCp`, and `KCCccp`.

Draw Rule 2: Being Able to Escape

This rule excludes the cases that both players have cannons. The conditions for the rule includes (i) Red has at least two pieces that are only capturable by only one black piece p other than cannons, and (ii) the black piece p is not capturable by any red pieces. However, a special case is excluded that Red has only two pieces while Black has at least three pieces including at least one cannon. In this case (e.g., `GGkmc`), Black may be able to force a win. Examples of likely draws judged by rule 2 includes `GGk`, `GGMkp`, `GGCkmn`, and `GGMMRRNkmr`.

Win Rule 1: Single Color

This is a trivial rule that when the remaining pieces all have the same color (e.g., red), the player of the color wins the game. Examples includes `K`, `KGCP`, and `KGGMRN`.

Win Rule 2: Perfect Domination

This rule excludes the cases that Black has two cannons. Red can usually win when it has at least two pieces other than cannons which (i) can capture all Black's remaining pieces, and (ii) are not capturable by any of the black pieces other than cannons. Such kind of pieces are called *dominating* pieces in this thesis. Examples of likely wins judged by rule 2 includes `KGm`, `KGc`, `KGmm`, `KGMMrrnnc`, and `KGGCmmrrppp`. With at least two dominating pieces, Red can force to win by tactically cooperating two of the dominating pieces to capture black pieces one at a time.

Win Rule 3: Limited Domination

This rule also excludes the cases that Black has two cannons. When Red has only one dominating piece p, it can usually win if it has at least one *limited dominating* piece defined as follows. A limited dominating piece is a piece other than p and cannons which can also capture all Black's remaining pieces but is capturable by black pieces of the same type. Examples of likely wins judged by rule 3 includes `KGg`, `KGMPk`, `KGggm`, and `KGGgmmrc`. The limited dominating piece(s) are usually used to curb black pieces of the same type. Red can then use the dominating piece to capture the curbed pieces.

Win Rule 4: Only One and Capturable Opponent Piece

Red can usually win when Black has only one piece p that is capturable by at least one red piece q other than cannons, and Red has at least two pieces. As long as the Manhattan distance between p and q is odd when Red is going to play, Red is proven to win [30]. Red pieces other than q can be used to change the Manhattan distance from even to odd. Note that, this rule overlaps with the previous two rules. Examples of likely wins judged by rule 4 includes `KGk`, `KGg`, `KGm`, `KGc`, `KGp`, `KGMPk`, and `KGCk`.

Appendix B. Detailed Results of the Strength Analysis Metrics

Table 10 lists the win rates of the 129 program settings against five 2×4 CDC baseline programs, OPT, EMM, MCTS-1k, MCTS-5k and MCTS-30k, in the five columns WR_{OPT} (WR_{2x4}), WR_{EMM}, WR_{1k}, WR_{5k}, and WR_{30k}, and the 66 program settings against MCTS-30k in 4×8 CDC in the column WR_{4x8}. The 95% confidence bounds for win rates in WR_{OPT}, WR_{EMM}, WR_{1k}, WR_{5k}, and WR_{30k} were less than 0.59%, and those in WR_{4x8} were less than 2.89%.

Incorporated technique	Number of simulations per action	Parameters	WR_{OPT} (WR_{2x4})	WR_{EMM}	WR_{1k}	WR_{5k}	WR_{30k}	WR_{4x8}
None	1	None	11.45%	11.70%	8.92%	6.77%	6.53%	
	10		17.89%	23.71%	15.30%	13.36%	13.23%	
	100		26.15%	39.69%	29.94%	25.10%	24.48%	
	500		34.49%	53.15%	45.54%	38.14%	35.23%	
	1,000		37.67%	57.24%	50.00%	43.19%	40.09%	
	2,000		40.56%	60.09%	54.58%	46.66%	42.92%	2.45%
	5,000		43.41%	62.84%	58.27%	50.00%	46.39%	12.20%
	10,000		45.13%	64.26%	60.19%	52.46%	48.38%	24.25%
	20,000		46.35%	64.97%	61.21%	53.96%	49.98%	38.60%
	30,000		46.78%	65.45%	61.89%	54.58%	50.00%	50.00%
	50,000		47.16%	65.73%	62.42%	55.37%	51.54%	62.55%
	100,000		47.81%	66.28%	62.99%	56.03%	52.26%	74.35%
	200,000		48.60%	66.54%	63.20%	56.24%	52.75%	80.60%

Table 10: Detailed win rates for the 129 and 66 analyzed program settings in 2×4 and 4×8 CDC.

Incorporated technique	Number of simulations per action	Parameters		WR_{OPT} (WR_{2x4})	WR_{EMM}	WR_{1k}	WR_{5k}	WR_{30k}	WR_{4x8}
EPT	30,000	ε_{EPT} = 0		47.06%	65.34%	61.76%	54.60%	51.04%	60.75%
		ε_{EPT} = 0.05		46.59%	65.07%	61.05%	54.12%	50.36%	55.30%
		ε_{EPT} = 0.1		45.96%	64.92%	61.03%	53.94%	50.15%	48.25%
		ε_{EPT} = 0.2		45.41%	64.28%	60.18%	52.90%	49.25%	35.00%
		ε_{EPT} = 0.3		44.04%	63.24%	57.89%	50.70%	46.95%	7.65%
		ε_{EPT} = 0.4		41.02%	60.60%	55.01%	47.30%	43.46%	0.20%
IMB		w_{IMB} = 0.1		47.42%	66.14%	62.60%	55.43%	51.76%	62.80%
		w_{IMB} = 0.2		48.13%	66.65%	62.87%	55.87%	52.28%	68.30%
		w_{IMB} = 0.3		48.16%	66.66%	62.84%	55.98%	52.33%	70.90%
		w_{IMB} = 0.4		47.92%	66.91%	62.35%	55.95%	52.08%	69.25%
		w_{IMB} = 0.5		47.86%	66.32%	61.74%	55.24%	51.36%	65.85%
		w_{IMB} = 0.6		47.65%	65.64%	61.19%	54.44%	50.85%	64.45%
		w_{IMB} = 0.7		46.78%	64.76%	60.01%	53.36%	49.86%	59.80%
		w_{IMB} = 0.8		46.57%	63.72%	58.66%	52.02%	49.04%	55.90%
		w_{IMB} = 0.9		46.33%	62.77%	57.63%	51.21%	48.20%	49.10%
		w_{IMB} = 1		45.00%	60.36%	54.99%	48.76%	45.72%	36.35%
SL		a_L = 0.2	k_L = 3	46.28%	65.64%	61.83%	54.41%	50.55%	52.80%
			k_L = 5	46.71%	65.79%	61.95%	54.66%	51.08%	54.60%
			k_L = 7	46.41%	65.45%	61.88%	54.51%	50.66%	55.25%
			k_L = 9	46.59%	65.81%	61.55%	54.56%	50.55%	52.40%
			k_L = 11	47.10%	66.04%	62.19%	54.38%	50.65%	54.10%
		a_L = 0.25	k_L = 3	46.80%	65.95%	62.09%	54.79%	50.82%	54.75%
			k_L = 5	46.96%	65.92%	61.82%	54.84%	51.08%	57.20%
			k_L = 7	46.56%	65.62%	61.94%	54.56%	50.86%	57.50%
			k_L = 9	46.71%	65.98%	61.91%	54.41%	50.68%	54.20%
			k_L = 11	46.92%	65.62%	61.72%	54.37%	50.74%	53.25%
		a_L = 0.3	k_L = 3	46.74%	65.81%	61.76%	54.47%	50.59%	54.95%
			k_L = 5	46.63%	65.83%	61.95%	54.57%	50.74%	55.70%
			k_L = 7	46.74%	65.59%	61.94%	54.45%	50.87%	57.35%
			k_L = 9	46.76%	65.49%	61.84%	54.64%	50.81%	56.05%
			k_L = 11	46.48%	65.71%	61.78%	54.72%	50.76%	54.25%

Table 10 (Continued)

Incorporated technique	Number of simulations per action	Parameters		WR_{OPT} (WR_{2x4})	WR_{EMM}	WR_{1k}	WR_{5k}	WR_{30k}	WR_{4x8}
TSQ	30,000	a_T = 0.3, k_T = 11	c_w = 0	46.81%	66.20%	61.48%	54.48%	50.86%	52.80%
			c_w = 0.01	46.89%	66.11%	61.41%	54.76%	51.06%	54.60%
			c_w = 0.02	46.43%	66.06%	61.35%	54.54%	50.86%	55.25%
			c_w = 0.03	46.57%	65.95%	61.61%	54.46%	50.69%	52.40%
			c_w = 0.04	46.62%	65.98%	61.73%	54.58%	50.84%	54.10%
		a_T = 0.3, c_w = 0.02	k_T = 7	46.55%	65.98%	61.70%	54.39%	50.88%	54.75%
			k_T = 9	46.43%	65.89%	61.72%	54.37%	50.73%	57.20%
			k_T = 13	46.53%	65.63%	61.85%	54.56%	50.77%	57.50%
			k_T = 15	46.51%	65.87%	61.79%	54.62%	50.68%	54.20%
		k_T = 11, c_w = 0.02	a_T = 0.2	46.65%	66.26%	61.85%	54.78%	50.80%	53.25%
			a_T = 0.25	46.59%	65.91%	62.00%	54.38%	50.77%	54.95%
			a_T = 0.3	46.42%	66.01%	61.59%	54.31%	50.76%	55.70%
PB		$\mathbf{H^B}$	c_{PB} = 0.001	47.08%	66.02%	62.23%	55.09%	51.24%	59.85%
			c_{PB} = 0.01	47.20%	66.39%	62.43%	55.62%	51.77%	68.40%
			c_{PB} = 0.1	45.03%	62.94%	58.26%	50.88%	47.69%	74.45%
			c_{PB} = 1	40.41%	56.82%	49.94%	43.43%	41.23%	57.30%
			c_{PB} = 10	37.88%	50.34%	42.73%	36.99%	35.21%	50.30%
		$\mathbf{H^B + H^C}$	c_{PB} = 0.001	46.84%	66.06%	62.43%	54.99%	51.24%	59.00%
			c_{PB} = 0.01	47.44%	66.32%	62.70%	55.69%	52.06%	65.55%
			c_{PB} = 0.1	45.34%	62.98%	58.04%	51.30%	47.94%	78.90%
			c_{PB} = 1	40.79%	57.62%	50.96%	44.78%	41.78%	59.00%
			c_{PB} = 10	37.56%	50.08%	43.00%	37.85%	35.72%	38.65%
		$\mathbf{H^B + H^C + H^F}$	c_{PB} = 0.001	46.73%	65.72%	62.08%	55.05%	51.00%	59.90%
			c_{PB} = 0.01	47.62%	66.51%	63.13%	56.03%	52.30%	69.55%
			c_{PB} = 0.1	41.88%	58.45%	53.34%	46.54%	43.43%	82.10%
			c_{PB} = 1	36.52%	50.68%	44.99%	39.01%	36.35%	50.70%
			c_{PB} = 10	33.90%	43.66%	38.55%	32.98%	31.79%	32.15%

Table 10 (Continued)

Incorporated technique	Number of simulations per action	Parameters		WR_{OPT} (WR_{2x4})	WR_{EMM}	WR_{1k}	WR_{5k}	WR_{30k}	WR_{4x8}
EPT	1,000	ε_{EPT} = 0		39.33%	58.61%	52.33%	44.86%	41.71%	
		ε_{EPT} = 0.05		38.44%	57.17%	50.35%	43.18%	39.56%	
		ε_{EPT} = 0.1		37.81%	56.42%	49.17%	41.92%	38.77%	
		ε_{EPT} = 0.2		34.96%	53.11%	45.75%	38.65%	35.38%	
		ε_{EPT} = 0.3		31.45%	48.68%	39.46%	33.28%	31.08%	
		ε_{EPT} = 0.4		25.94%	40.84%	31.21%	26.33%	24.63%	
IMB		w_{IMB} = 0.1		40.63%	59.97%	54.13%	46.17%	42.28%	
		w_{IMB} = 0.2		41.66%	60.92%	55.70%	47.99%	44.20%	
		w_{IMB} = 0.3		42.60%	62.04%	56.10%	49.35%	45.52%	
		w_{IMB} = 0.4		43.10%	62.10%	56.55%	49.30%	45.72%	
		w_{IMB} = 0.5		43.14%	62.39%	56.19%	48.88%	45.65%	
		w_{IMB} = 0.6		43.31%	61.45%	55.65%	48.66%	45.47%	
		w_{IMB} = 0.7		43.04%	60.47%	54.17%	47.47%	44.42%	
		w_{IMB} = 0.8		42.13%	59.37%	53.21%	46.68%	43.49%	
		w_{IMB} = 0.9		41.96%	58.18%	51.99%	45.30%	42.49%	
		w_{IMB} = 1		40.60%	55.08%	48.85%	42.75%	40.01%	
SL		a_L = 0.2	k_L = 3	38.54%	58.03%	51.23%	43.57%	40.23%	
			k_L = 5	38.35%	58.45%	51.48%	44.12%	40.53%	
			k_L = 7	38.59%	58.15%	52.01%	43.94%	40.39%	
			k_L = 9	38.97%	58.01%	51.31%	43.84%	40.68%	
			k_L = 11	38.95%	58.18%	51.39%	43.67%	40.23%	
		a_L = 0.25	k_L = 3	38.77%	57.74%	51.43%	43.47%	40.33%	
			k_L = 5	38.87%	57.96%	52.05%	43.81%	40.57%	
			k_L = 7	38.64%	58.27%	51.20%	43.79%	40.29%	
			k_L = 9	38.70%	58.08%	51.75%	43.94%	40.36%	
			k_L = 11	38.52%	57.95%	51.33%	43.80%	40.41%	
		a_L = 0.3	k_L = 3	38.45%	58.15%	51.61%	43.71%	40.11%	
			k_L = 5	39.05%	58.27%	51.85%	43.86%	40.78%	
			k_L = 7	38.46%	58.50%	51.39%	43.91%	40.46%	
			k_L = 9	38.73%	58.01%	51.91%	44.08%	40.16%	
			k_L = 11	38.34%	58.04%	51.90%	43.89%	40.46%	

Table 10 (Continued)

Incorporated technique	Number of simulations per action	Parameters		WR_{OPT} (WR_{2x4})	WR_{EMM}	WR_{1k}	WR_{5k}	WR_{30k}	WR_{4x8}
TSQ	1,000	a_T = 0.3, k_T = 11	c_w = 0	39.13%	58.81%	52.11%	44.58%	41.39%	
			c_w = 0.01	39.31%	58.90%	52.52%	44.56%	41.10%	
			c_w = 0.02	39.20%	58.84%	52.82%	44.68%	40.88%	
			c_w = 0.03	39.49%	58.99%	52.11%	44.75%	41.19%	
			c_w = 0.04	39.24%	58.61%	52.69%	45.01%	41.10%	
		a_T = 0.3, c_w = 0.02	k_T = 7	39.58%	59.06%	52.22%	44.47%	41.21%	
			k_T = 9	39.37%	58.69%	52.96%	44.70%	40.97%	
			k_T = 13	39.41%	58.57%	52.53%	44.63%	40.81%	
			k_T = 15	39.49%	58.63%	52.27%	44.52%	41.29%	
		k_T = 11, c_w = 0.02	a_T = 0.2	38.92%	58.74%	52.37%	44.35%	40.55%	
			a_T = 0.25	39.14%	58.57%	52.17%	44.39%	40.93%	
			a_T = 0.3	39.09%	58.61%	52.22%	45.00%	41.35%	
PB		$\mathbf{H^B}$	c_{PB} = 0.001	38.68%	57.65%	51.09%	43.84%	40.93%	
			c_{PB} = 0.01	40.43%	60.13%	53.94%	46.31%	42.94%	
			c_{PB} = 0.1	38.99%	57.14%	50.44%	44.06%	40.61%	
			c_{PB} = 1	36.18%	49.43%	42.35%	36.27%	34.23%	
			c_{PB} = 10	35.84%	48.20%	41.33%	35.39%	33.38%	
		$\mathbf{H^B + H^C}$	c_{PB} = 0.001	38.92%	57.64%	51.55%	44.02%	40.76%	
			c_{PB} = 0.01	40.47%	59.96%	54.34%	46.56%	42.95%	
			c_{PB} = 0.1	39.80%	57.70%	50.94%	44.26%	41.37%	
			c_{PB} = 1	35.70%	49.00%	42.30%	36.34%	34.20%	
			c_{PB} = 10	35.39%	46.91%	40.47%	34.89%	33.12%	
		$\mathbf{H^B + H^C + H^F}$	c_{PB} = 0.001	39.08%	58.43%	51.94%	44.34%	41.04%	
			c_{PB} = 0.01	41.57%	61.06%	55.59%	47.55%	43.59%	
			c_{PB} = 0.1	38.64%	54.55%	48.75%	42.02%	38.95%	
			c_{PB} = 1	33.84%	44.52%	38.88%	33.32%	31.91%	
			c_{PB} = 10	33.21%	42.30%	37.23%	31.90%	30.64%	

Table 10 (Continued)

Table 11 lists the prediction rates of the 129 program settings for 2×4 CDC with respect to the best actions, the actions played by MCTS-1k, and the actions by MCTS-1M in the columns PR_{BEST}, PR_{1k}, and PR_{1M}, the mean squared errors with respect to the theoretical values, the game outcomes from self-plays by MCTS-1k, and the game outcomes from self-plays by MCTS-1M in the columns MSE_{TH}, $MSE_{GO\text{-}1k}$, and $MSE_{GO\text{-}1M}$ respectively. The 95% confidence bounds for prediction rates in PR_{BEST} were less than 0.42%, and those in PR_{1k} and PR_{1M} were less than 0.45%. The 95% confidence bounds for mean squared errors in MSE_{TH} were less than 0.0012, and those in $MSE_{GO\text{-}1k}$ and $MSE_{GO\text{-}1M}$ were less than 0.0017.

Incorporated technique	**Number of simulations per action**	**Parameters**	**PR_{BEST}**	**PR_{1k}**	**PR_{1M}**	**MSE_{TH}**	**$MSE_{GO\text{-}1k}$**	**$MSE_{GO\text{-}1M}$**
None	1	None	69.34%	41.98%	42.49%	0.0955	0.1521	0.1445
	10		73.86%	48.04%	47.67%	0.0464	0.1029	0.0958
	100		78.50%	56.76%	53.55%	0.0275	0.0849	0.0769
	500		82.65%	67.58%	61.16%	0.0205	0.0773	0.0699
	1,000		84.46%	71.82%	64.54%	0.0175	0.0743	0.0669
	2,000		86.08%	71.80%	67.69%	0.0145	0.0713	0.0640
	5,000		87.59%	70.76%	71.70%	0.0109	0.0679	0.0603
	10,000		88.79%	69.42%	74.49%	0.0085	0.0660	0.0579
	20,000		89.46%	68.27%	77.04%	0.0065	0.0646	0.0559
	30,000		89.94%	67.42%	78.78%	0.0055	0.0640	0.0549
	50,000		90.65%	66.84%	80.71%	0.0044	0.0634	0.0539
	100,000		91.43%	66.07%	83.36%	0.0033	0.0630	0.0529
	200,000		92.15%	65.38%	85.43%	0.0024	0.0627	0.0521

Table 11: Detailed prediction rates and mean squared errors of the 129 analyzed program settings in 2×4 CDC.

Incorporated technique	Number of simulations per action	Parameters		PR_{BEST}	PR_{1k}	PR_{1M}	MSE_{TH}	$MSE_{GO\text{-}1k}$	$MSE_{GO\text{-}1M}$
EPT	30,000	ε_{EPT} = 0		90.54%	67.27%	75.91%	0.0051	0.0637	0.0547
		ε_{EPT} = 0.05		90.23%	66.18%	76.26%	0.0060	0.0646	0.0556
		ε_{EPT} = 0.1		89.78%	65.46%	75.88%	0.0071	0.0656	0.0566
		ε_{EPT} = 0.2		88.91%	64.72%	74.62%	0.0097	0.0679	0.0591
		ε_{EPT} = 0.3		87.57%	64.18%	72.53%	0.0131	0.0712	0.0625
		ε_{EPT} = 0.4		85.62%	63.36%	69.50%	0.0175	0.0752	0.0668
IMB		w_{IMB} = 0.1		91.12%	66.84%	78.75%	0.0052	0.0640	0.0546
		w_{IMB} = 0.2		91.81%	66.49%	78.25%	0.0051	0.0641	0.0546
		w_{IMB} = 0.3		92.34%	66.17%	77.33%	0.0053	0.0644	0.0548
		w_{IMB} = 0.4		92.45%	65.68%	76.27%	0.0059	0.0650	0.0554
		w_{IMB} = 0.5		92.18%	65.24%	75.28%	0.0070	0.0659	0.0564
		w_{IMB} = 0.6		92.07%	64.95%	74.02%	0.0086	0.0672	0.0580
		w_{IMB} = 0.7		91.64%	64.85%	72.62%	0.0107	0.0692	0.0601
		w_{IMB} = 0.8		91.32%	64.28%	71.32%	0.0133	0.0715	0.0627
		w_{IMB} = 0.9		90.74%	63.83%	69.63%	0.0163	0.0744	0.0657
		w_{IMB} = 1		90.03%	58.23%	65.39%	0.0209	0.0789	0.0702
SL		a_L = 0.2	k_L = 3	89.93%	66.83%	77.88%	0.0045	0.0632	0.0540
			k_L = 5	90.27%	66.66%	77.43%	0.0045	0.0633	0.0540
			k_L = 7	89.87%	66.41%	76.91%	0.0047	0.0634	0.0541
			k_L = 9	89.97%	66.03%	76.74%	0.0049	0.0636	0.0543
			k_L = 11	89.85%	66.04%	76.55%	0.0051	0.0637	0.0544
		a_L = 0.25	k_L = 3	89.88%	66.48%	77.46%	0.0044	0.0631	0.0538
			k_L = 5	89.73%	65.81%	76.34%	0.0046	0.0633	0.0540
			k_L = 7	89.99%	65.55%	76.24%	0.0049	0.0637	0.0543
			k_L = 9	89.90%	65.34%	75.91%	0.0052	0.0640	0.0547
			k_L = 11	90.16%	65.57%	76.12%	0.0055	0.0642	0.0548
		a_L = 0.3	k_L = 3	90.34%	66.18%	76.90%	0.0044	0.0631	0.0538
			k_L = 5	89.87%	65.45%	75.81%	0.0048	0.0635	0.0541
			k_L = 7	89.71%	65.23%	75.46%	0.0052	0.0640	0.0546
			k_L = 9	90.11%	65.14%	75.38%	0.0057	0.0645	0.0551
			k_L = 11	89.94%	65.33%	75.33%	0.0061	0.0648	0.0555

Table 11 (Continued)

Incorporated technique	Number of simulations per action	Parameters		PR_{BEST}	PR_{1k}	PR_{1M}	MSE_{TH}	$MSE_{GO\text{-}1k}$	$MSE_{GO\text{-}1M}$
TSQ	30,000	a_T = 0.3, k_T = 11	c_w = 0	89.89%	64.41%	73.02%	0.0080	0.0665	0.0573
			c_w = 0.01	89.93%	64.40%	73.01%	0.0079	0.0664	0.0572
			c_w = 0.02	90.18%	64.48%	73.19%	0.0078	0.0663	0.0571
			c_w = 0.03	89.83%	64.41%	73.05%	0.0077	0.0662	0.0571
			c_w = 0.04	89.97%	64.30%	73.03%	0.0076	0.0661	0.0570
		a_T = 0.3, c_w = 0.02	k_T = 7	89.78%	64.67%	73.60%	0.0066	0.0652	0.0560
			k_T = 9	90.13%	64.46%	73.13%	0.0073	0.0658	0.0566
			k_T = 13	89.89%	64.32%	72.83%	0.0082	0.0668	0.0576
			k_T = 15	90.14%	64.46%	73.04%	0.0085	0.0670	0.0579
		k_T = 11, c_w = 0.02	a_T = 0.2	90.02%	65.31%	74.45%	0.0061	0.0646	0.0554
			a_T = 0.25	90.12%	64.83%	73.74%	0.0068	0.0654	0.0562
			a_T = 0.3	90.04%	64.09%	72.45%	0.0089	0.0674	0.0582
PB		$\mathbf{H^B}$	c_{PB} = 0.001	90.68%	67.16%	77.96%	0.0050	0.0636	0.0545
			c_{PB} = 0.01	91.04%	66.38%	77.38%	0.0041	0.0633	0.0538
			c_{PB} = 0.1	90.02%	61.95%	69.69%	0.0120	0.0691	0.0615
			c_{PB} = 1	87.69%	59.27%	65.22%	0.0195	0.0778	0.0687
			c_{PB} = 10	86.57%	57.69%	62.58%	0.0314	0.0894	0.0804
		$\mathbf{H^B + H^C}$	c_{PB} = 0.001	90.52%	66.98%	77.81%	0.0049	0.0636	0.0544
			c_{PB} = 0.01	91.36%	66.63%	77.46%	0.0041	0.0632	0.0537
			c_{PB} = 0.1	89.96%	62.03%	69.73%	0.0118	0.0688	0.0613
			c_{PB} = 1	87.58%	59.44%	64.96%	0.0176	0.0758	0.0668
			c_{PB} = 10	86.33%	57.82%	62.73%	0.0271	0.0849	0.0761
		$\mathbf{H^B + H^C + H^F}$	c_{PB} = 0.001	90.48%	67.13%	77.73%	0.0049	0.0635	0.0544
			c_{PB} = 0.01	91.68%	66.36%	77.80%	0.0040	0.0631	0.0536
			c_{PB} = 0.1	88.33%	61.23%	67.38%	0.0140	0.0706	0.0632
			c_{PB} = 1	85.25%	58.42%	62.15%	0.0201	0.0779	0.0691
			c_{PB} = 10	84.26%	57.09%	60.33%	0.0283	0.0859	0.0771

Table 11 (Continued)

Incorporated technique	Number of simulations per action	Parameters		PR_{BEST}	PR_{1k}	PR_{1M}	MSE_{TH}	$MSE_{GO\text{-}1k}$	$MSE_{GO\text{-}1M}$
EPT	1,000	ε_{EPT} = 0		85.25%	69.28%	63.88%	0.0147	0.0711	0.0641
		ε_{EPT} = 0.05		84.71%	67.50%	63.44%	0.0168	0.0734	0.0662
		ε_{EPT} = 0.1		83.90%	66.29%	63.07%	0.0197	0.0763	0.0691
		ε_{EPT} = 0.2		82.93%	64.40%	62.30%	0.0270	0.0838	0.0763
		ε_{EPT} = 0.3		81.10%	61.40%	59.96%	0.0362	0.0931	0.0855
		ε_{EPT} = 0.4		78.65%	58.25%	56.99%	0.0455	0.1025	0.0949
IMB		w_{IMB} = 0.1		85.92%	71.73%	66.01%	0.0169	0.0737	0.0663
		w_{IMB} = 0.2		86.88%	70.94%	66.79%	0.0166	0.0734	0.0659
		w_{IMB} = 0.3		87.32%	70.37%	67.12%	0.0165	0.0734	0.0658
		w_{IMB} = 0.4		87.76%	69.13%	66.66%	0.0170	0.0739	0.0663
		w_{IMB} = 0.5		87.77%	68.19%	66.44%	0.0178	0.0748	0.0671
		w_{IMB} = 0.6		87.81%	67.28%	66.16%	0.0189	0.0759	0.0681
		w_{IMB} = 0.7		87.46%	66.04%	65.69%	0.0201	0.0772	0.0694
		w_{IMB} = 0.8		87.32%	64.98%	65.04%	0.0214	0.0786	0.0707
		w_{IMB} = 0.9		87.06%	63.51%	64.09%	0.0227	0.0799	0.0720
		w_{IMB} = 1		86.03%	57.28%	61.28%	0.0243	0.0816	0.0735
SL		a_L = 0.2	k_L = 3	84.81%	69.09%	67.61%	0.0155	0.0723	0.0649
			k_L = 5	84.60%	68.50%	67.71%	0.0150	0.0717	0.0644
			k_L = 7	84.63%	68.29%	67.79%	0.0149	0.0715	0.0642
			k_L = 9	84.76%	68.37%	67.61%	0.0148	0.0714	0.0642
			k_L = 11	84.80%	68.42%	67.69%	0.0148	0.0715	0.0642
		a_L = 0.25	k_L = 3	84.82%	68.55%	68.03%	0.0150	0.0718	0.0644
			k_L = 5	84.81%	68.41%	67.93%	0.0147	0.0713	0.0639
			k_L = 7	84.76%	68.06%	68.09%	0.0147	0.0714	0.0641
			k_L = 9	84.75%	68.08%	67.94%	0.0147	0.0715	0.0640
			k_L = 11	84.80%	68.17%	68.17%	0.0148	0.0716	0.0642
		a_L = 0.3	k_L = 3	84.83%	68.18%	68.29%	0.0148	0.0715	0.0642
			k_L = 5	85.03%	67.80%	68.33%	0.0145	0.0712	0.0639
			k_L = 7	84.62%	67.55%	68.15%	0.0147	0.0714	0.0641
			k_L = 9	84.68%	67.58%	68.17%	0.0149	0.0717	0.0643
			k_L = 11	84.87%	67.65%	68.24%	0.0151	0.0718	0.0645

Table 11 (Continued)

Incorporated technique	Number of simulations per action	Parameters		PR_{BEST}	PR_{1k}	PR_{1M}	MSE_{TH}	$MSE_{GO\text{-}1k}$	$MSE_{GO\text{-}1M}$
TSQ	1,000	a_T = 0.3, k_T = 11	c_w = 0	85.05%	67.94%	65.84%	0.0175	0.0741	0.0669
			c_w = 0.01	85.07%	67.81%	65.74%	0.0174	0.0739	0.0668
			c_w = 0.02	84.84%	67.75%	65.50%	0.0172	0.0738	0.0666
			c_w = 0.03	85.13%	67.76%	65.66%	0.0171	0.0737	0.0665
			c_w = 0.04	85.11%	67.86%	65.72%	0.0168	0.0733	0.0661
		a_T = 0.3, c_w = 0.02	k_T = 7	85.19%	68.11%	65.85%	0.0165	0.0731	0.0659
			k_T = 9	85.02%	67.65%	65.70%	0.0169	0.0735	0.0662
			k_T = 13	84.81%	67.88%	65.62%	0.0175	0.0740	0.0668
			k_T = 15	85.07%	67.58%	65.54%	0.0176	0.0742	0.0670
		k_T = 11, c_w = 0.02	a_T = 0.2	84.74%	68.51%	65.40%	0.0162	0.0728	0.0656
			a_T = 0.25	84.89%	67.94%	65.67%	0.0166	0.0731	0.0659
			a_T = 0.3	84.84%	67.42%	65.46%	0.0181	0.0745	0.0674
PB		$\mathbf{H^B}$	c_{PB} = 0.001	84.92%	70.27%	64.40%	0.0161	0.0727	0.0655
			c_{PB} = 0.01	85.40%	68.54%	65.66%	0.0143	0.0708	0.0636
			c_{PB} = 0.1	85.30%	61.67%	63.21%	0.0180	0.0752	0.0671
			c_{PB} = 1	84.05%	58.46%	60.26%	0.0283	0.0857	0.0773
			c_{PB} = 10	83.63%	58.00%	59.88%	0.0312	0.0884	0.0802
		$\mathbf{H^B + H^C}$	c_{PB} = 0.001	85.09%	69.77%	64.60%	0.0160	0.0726	0.0654
			c_{PB} = 0.01	85.54%	68.60%	65.94%	0.0141	0.0706	0.0635
			c_{PB} = 0.1	85.50%	61.87%	63.40%	0.0172	0.0745	0.0664
			c_{PB} = 1	83.95%	58.33%	60.49%	0.0253	0.0829	0.0743
			c_{PB} = 10	83.72%	57.64%	59.98%	0.0289	0.0861	0.0778
		$\mathbf{H^B + H^C + H^F}$	c_{PB} = 0.001	84.62%	69.67%	64.32%	0.0159	0.0726	0.0654
			c_{PB} = 0.01	86.18%	68.39%	66.51%	0.0138	0.0702	0.0631
			c_{PB} = 0.1	85.66%	60.71%	63.12%	0.0180	0.0748	0.0667
			c_{PB} = 1	83.75%	57.68%	60.19%	0.0266	0.0839	0.0752
			c_{PB} = 10	83.95%	57.11%	59.72%	0.0298	0.0872	0.0785

Table 11 (Continued)

Appendix C. Detailed Results of AlphaZero on 2×4 CDC

Table 12 lists the numbers of training positions and the win rates obtained at the end of each iteration of the AlphaZero algorithm with settings of symmetries. The tested settings of symmetries included None, Color, Position, and Full for none, color, positional, and both color and positional symmetries respectively. The 95% confidence bounds of the win rates were less than 0.98%.

	#Training Positions				Win Rates			
	None	**Color**	**Position**	**Full**	**None**	**Color**	**Position**	**Full**
0	0	0	0	0	0.00%	0.00%	0.00%	0.00%
1	17,290	34,580	69,160	138,320	0.05%	0.23%	0.91%	2.94%
2	51,153	102,334	204,860	411,104	0.50%	1.80%	3.82%	8.61%
3	84,227	168,480	338,276	679,640	1.24%	3.67%	10.84%	16.87%
4	117,198	234,768	472,096	949,144	2.27%	5.96%	17.03%	24.60%
5	150,116	301,462	606,756	1,221,816	3.32%	9.42%	20.55%	35.11%
6	183,443	368,528	743,916	1,499,960	4.47%	11.94%	26.36%	41.72%
7	217,074	435,924	882,416	1,783,720	5.58%	15.71%	31.08%	43.63%
8	250,700	503,762	1,022,156	2,072,816	7.27%	18.98%	35.00%	46.90%
9	284,432	571,714	1,164,024	2,368,720	8.57%	24.01%	37.81%	47.22%
10	318,158	640,446	1,307,160	2,669,952	10.55%	27.95%	39.63%	48.07%
11	352,089	710,442	1,452,784	2,972,008	11.79%	30.80%	42.57%	47.67%
12	386,003	781,230	1,600,740	3,276,680	12.42%	33.17%	43.89%	47.46%
13	419,955	852,500	1,749,696	3,584,624	13.66%	35.78%	46.11%	47.52%
14	454,201	923,608	1,898,572	3,894,384	15.82%	38.57%	46.05%	48.05%
15	488,490	995,624	2,048,144	4,205,672	18.02%	39.62%	46.18%	47.83%
16	523,144	1,068,448	2,199,244	4,519,872	20.00%	40.56%	46.91%	47.68%
17	558,239	1,141,302	2,351,348	4,836,544	22.62%	41.23%	46.59%	47.87%
18	593,202	1,214,070	2,506,256	5,151,912	23.36%	42.87%	47.48%	48.69%

Table 12: Detailed numbers of training positions and win rates for applying AlphaZero on 2×4 CDC with different settings of symmetries.

	#Training Positions				Win Rates			
	None	**Color**	**Position**	**Full**	**None**	**Color**	**Position**	**Full**
19	628,298	1,287,064	2,661,892	5,467,216	25.35%	43.75%	48.17%	48.89%
20	663,832	1,360,672	2,817,156	5,786,024	27.88%	44.46%	48.20%	48.11%
21	699,363	1,434,304	2,974,020	6,108,512	29.08%	43.93%	48.14%	46.91%
22	734,898	1,508,330	3,130,408	6,428,672	32.34%	44.05%	48.03%	48.41%
23	770,388	1,583,090	3,286,036	6,748,224	33.94%	45.27%	47.35%	49.22%
24	805,859	1,657,892	3,443,060	7,068,056	35.78%	45.44%	47.65%	48.74%
25	841,396	1,732,064	3,601,396	7,387,056	35.14%	45.91%	47.87%	48.84%
26	877,264	1,806,498	3,759,392	7,706,400	36.12%	46.20%	47.31%	48.41%
27	913,091	1,881,542	3,916,640	8,027,088	37.32%	46.13%	48.07%	49.26%
28	948,938	1,956,226	4,074,536	8,353,040	38.20%	46.14%	48.38%	50.03%
29	985,397	2,031,412	4,233,332	8,677,304	38.59%	46.51%	48.48%	50.46%
30	1,021,734	2,107,040	4,392,628	9,000,632	38.94%	46.49%	48.50%	51.02%
31	1,057,667	2,182,582	4,551,860	9,327,000	39.68%	46.62%	48.81%	49.85%
32	1,093,701	2,259,034	4,709,444	9,652,728	39.99%	46.48%	48.43%	50.37%
33	1,129,778	2,336,028	4,866,976	9,978,288	40.60%	46.43%	49.01%	50.26%
34	1,165,943	2,411,894	5,024,968	10,302,944	40.86%	46.70%	48.73%	49.08%
35	1,202,290	2,487,560	5,183,748	10,626,480	41.33%	46.81%	48.61%	49.67%
36	1,238,222	2,563,520	5,342,208	10,951,952	41.82%	47.29%	48.77%	49.53%
37	1,274,245	2,639,184	5,498,408	11,280,936	42.68%	47.32%	48.75%	49.45%
38	1,310,446	2,715,054	5,655,480	11,609,656	42.77%	47.73%	48.60%	50.11%
39	1,346,353	2,789,990	5,814,908	11,941,120	42.61%	47.88%	48.80%	49.74%
40	1,382,689	2,864,536	5,974,400	12,271,304	42.74%	47.92%	48.75%	49.94%
41	1,419,033	2,940,954	6,132,260	12,598,136	43.34%	47.87%	49.01%	49.99%
42	1,455,394	3,018,132	6,289,120	12,924,312	43.05%	47.78%	48.81%	50.13%
43	1,492,000	3,094,174	6,446,296	13,246,184	43.52%	47.85%	48.45%	49.66%
44	1,528,281	3,169,908	6,604,696	13,570,000	43.80%	47.95%	48.76%	49.76%
45	1,564,665	3,246,424	6,763,440	13,899,416	43.53%	48.07%	48.77%	49.89%
46	1,601,216	3,322,502	6,921,188	14,229,416	43.82%	47.68%	48.76%	49.84%
47	1,637,561	3,398,212	7,078,312	14,556,504	43.85%	47.81%	49.04%	49.96%
48	1,673,795	3,474,214	7,236,076	14,884,768	44.03%	47.68%	48.82%	50.27%
49	1,710,068	3,550,304	7,395,952	15,217,528	44.09%	47.66%	48.86%	49.50%
50	1,746,173	3,626,748	7,555,884	15,548,312	43.88%	47.69%	48.89%	49.60%

Table 12 (Continued)

Table 13 lists the absolute errors (AEs) and the cumulative number of distinct seen positions obtained at the end of each iteration of the AlphaZero algorithm with settings of symmetries. The tested settings of symmetries included None, Color, Position, and Full.

	AEs				#Distinct Seen Positions			
	None	**Color**	**Position**	**Full**	**None**	**Color**	**Position**	**Full**
0	0.2692	0.2692	0.2692	0.2692	0	0	0	0
1	0.3380	0.1866	0.2182	0.2003	11,157	19,523	32,353	51,133
2	0.2888	0.1064	0.0584	0.1922	19,003	31,871	50,021	74,665
3	0.2132	0.1437	0.2676	0.1575	25,532	41,537	63,327	89,717
4	0.2523	0.3696	0.1615	0.4681	31,273	49,591	73,837	100,905
5	0.0995	0.1475	0.3061	0.0047	36,494	56,665	82,027	108,389
6	0.0893	0.1925	0.2483	0.2048	41,093	62,689	88,753	114,181
7	0.1728	0.2136	0.3211	0.0125	45,365	67,953	94,257	118,133
8	0.2997	0.2307	0.2434	0.0574	49,125	72,603	98,669	121,063
9	0.1564	0.3393	0.0250	0.0275	52,759	76,899	102,077	123,479
10	0.3075	0.1313	0.1161	0.0569	56,183	80,567	105,195	125,503
11	0.2726	0.1228	0.0118	0.0383	59,210	84,085	107,819	127,295
12	0.2431	0.0149	0.0262	0.0524	62,135	87,015	110,251	128,847
13	0.1254	0.0730	0.1116	0.0639	64,846	89,669	112,223	130,179
14	0.2258	0.1713	0.0227	0.0299	67,412	91,935	113,721	131,451
15	0.2651	0.1025	0.0108	0.0287	69,725	94,071	115,189	132,391
16	0.1595	0.0506	0.2102	0.0415	71,903	95,971	116,705	133,531
17	0.1301	0.0588	0.0245	0.0300	73,957	97,729	117,755	134,407
18	0.0880	0.0726	0.0319	0.0898	75,884	99,471	119,003	135,311
19	0.2237	0.0276	0.0123	0.0144	77,826	100,813	120,039	136,003
20	0.0808	0.0757	0.0412	0.0367	79,609	102,051	121,125	136,831
21	0.1129	0.0151	0.0814	0.0780	81,181	103,313	122,117	137,515
22	0.2126	0.0281	0.0452	0.0153	82,671	104,417	123,133	137,943
23	0.2640	0.0461	0.1328	0.0964	84,064	105,525	124,013	138,599
24	0.2553	0.0627	0.0724	0.2038	85,287	106,583	124,731	139,231
25	0.0306	0.1001	0.0016	0.0230	86,506	107,533	125,377	139,723

Table 13: Detailed AEs and cumulative number of distinct seen positions for applying AlphaZero on 2×4 CDC with different settings of symmetries.

	AEs				#Distinct Seen Positions			
	None	**Color**	**Position**	**Full**	**None**	**Color**	**Position**	**Full**
26	0.1081	0.0259	0.0275	0.0226	87,758	108,381	126,041	140,079
27	0.0775	0.0292	0.0285	0.0475	88,941	109,189	126,813	140,531
28	0.0922	0.0460	0.0088	0.0567	89,964	109,897	127,609	141,059
29	0.0986	0.1221	0.0210	0.0692	91,003	110,701	128,081	141,403
30	0.1012	0.0475	0.0492	0.0095	91,963	111,495	128,563	141,763
31	0.0902	0.0191	0.0129	0.0372	92,927	112,357	129,097	142,139
32	0.0827	0.0823	0.0479	0.0483	93,832	112,971	129,605	142,667
33	0.0968	0.0624	0.0056	0.0083	94,695	113,595	130,165	143,055
34	0.0833	0.0422	0.0579	0.0107	95,591	114,099	130,743	143,405
35	0.0980	0.0491	0.0220	0.0027	96,371	114,689	131,325	143,745
36	0.0972	0.0139	0.0139	0.0180	97,149	115,255	131,717	143,973
37	0.0624	0.0280	0.0117	0.0073	97,868	115,755	132,171	144,337
38	0.0577	0.0446	0.0453	0.0359	98,584	116,361	132,489	144,689
39	0.0912	0.0211	0.0143	0.0514	99,246	116,881	132,815	145,089
40	0.0874	0.0209	0.0244	0.0400	99,991	117,421	133,162	145,453
41	0.0655	0.0672	0.0482	0.0130	100,640	117,959	133,534	145,713
42	0.1195	0.0541	0.0544	0.0411	101,230	118,483	133,842	146,025
43	0.0291	0.0378	0.0051	0.0559	101,798	119,003	134,202	146,313
44	0.0375	0.0885	0.0200	0.0072	102,402	119,435	134,658	146,685
45	0.0609	0.0763	0.0056	0.0032	103,028	119,871	135,026	147,001
46	0.0573	0.0755	0.0160	0.0297	103,685	120,307	135,474	147,369
47	0.0672	0.0752	0.0474	0.0092	104,222	120,681	135,850	147,577
48	0.0830	0.0226	0.0118	0.0079	104,704	121,141	136,206	147,913
49	0.0833	0.0175	0.0085	0.0094	105,202	121,499	136,502	148,121
50	0.0864	0.0330	0.0401	0.0368	105,748	121,875	136,836	148,369

Table 13 (Continued)

Table 14 lists the win rates obtained at the end of each iteration of the AlphaZero algorithm with different c_{puct}'s. The tested values of c_{puct} included 0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, and 16. The 95% confidence bounds of the win rates were less than 0.98%.

	0.0625	**0.125**	**0.25**	**0.5**	**1**	**2**	**4**	**8**	**16**
0	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	0.94%	1.12%	2.31%	2.84%	2.94%	2.19%	2.17%	2.01%	1.59%
2	1.47%	5.26%	7.54%	7.87%	8.61%	8.92%	7.33%	4.96%	4.95%
3	1.22%	12.35%	20.37%	24.73%	16.87%	11.91%	11.35%	7.85%	7.95%
4	1.55%	19.91%	32.93%	33.18%	24.60%	20.19%	15.18%	11.50%	10.74%
5	1.64%	26.78%	37.73%	40.30%	35.11%	23.30%	18.62%	12.22%	13.65%
6	1.63%	36.22%	41.62%	43.76%	41.72%	34.68%	17.78%	13.52%	16.98%
7	1.77%	37.68%	44.29%	45.93%	43.63%	35.92%	22.28%	12.71%	13.63%
8	1.67%	37.80%	45.41%	46.06%	46.90%	39.08%	23.06%	13.55%	10.84%
9	1.66%	39.05%	46.72%	47.12%	47.22%	43.11%	24.55%	14.73%	10.64%
10	1.76%	39.77%	47.52%	46.02%	48.07%	45.83%	24.88%	19.48%	13.40%
11	1.67%	39.75%	48.42%	46.43%	47.67%	46.64%	32.12%	23.89%	13.59%
12	1.58%	39.65%	48.89%	47.00%	47.46%	47.76%	31.24%	29.66%	17.03%
13	1.52%	41.01%	49.19%	47.11%	47.52%	47.39%	33.77%	30.89%	17.00%
14	1.65%	41.19%	49.07%	46.96%	48.05%	47.49%	37.95%	34.23%	21.11%
15	1.52%	41.19%	49.03%	48.61%	47.83%	47.99%	39.29%	31.83%	18.93%
16	1.65%	41.50%	49.18%	49.70%	47.68%	48.08%	44.25%	26.57%	17.04%
17	1.67%	40.53%	49.31%	49.44%	47.87%	48.01%	45.39%	26.60%	15.36%
18	1.51%	42.06%	49.51%	49.38%	48.69%	47.54%	46.13%	28.02%	21.96%
19	1.64%	42.24%	48.94%	49.54%	48.89%	47.79%	46.74%	28.51%	21.36%
20	1.66%	42.77%	49.21%	48.73%	48.11%	47.66%	46.57%	34.45%	15.67%
21	1.65%	42.66%	48.77%	48.55%	46.91%	48.28%	47.17%	37.23%	23.16%
22	1.65%	42.63%	49.24%	48.70%	48.41%	48.22%	48.07%	37.25%	31.07%
23	1.61%	43.12%	47.91%	49.26%	49.22%	48.16%	47.68%	37.31%	35.04%
24	1.53%	45.03%	48.75%	48.61%	48.74%	48.32%	47.70%	38.76%	39.29%
25	1.65%	44.92%	48.61%	49.39%	48.84%	48.36%	48.79%	40.35%	40.41%

Table 14: Detailed win rates for applying AlphaZero on 2×4 CDC with different c_{puct}.

	0.0625	0.125	0.25	0.5	1	2	4	8	16
26	1.58%	44.91%	48.54%	49.33%	48.41%	48.11%	47.40%	39.90%	44.42%
27	1.47%	45.25%	48.40%	49.71%	49.26%	48.14%	48.54%	39.88%	44.63%
28	1.56%	44.64%	48.13%	49.92%	50.03%	48.63%	47.40%	41.19%	38.42%
29	1.41%	44.48%	48.08%	50.09%	50.46%	48.69%	47.78%	45.59%	35.47%
30	1.41%	44.49%	48.42%	50.32%	51.02%	48.36%	48.17%	47.35%	34.38%
31	1.55%	44.58%	48.89%	49.92%	49.85%	48.30%	47.21%	47.57%	34.38%
32	1.68%	44.53%	48.84%	49.30%	50.37%	48.59%	48.65%	47.10%	34.25%
33	1.49%	44.71%	48.13%	49.35%	50.26%	47.74%	48.41%	47.16%	34.80%
34	1.53%	44.46%	48.62%	49.50%	49.08%	48.05%	47.69%	47.39%	34.10%
35	1.55%	44.79%	48.58%	50.34%	49.67%	47.94%	47.60%	47.81%	36.27%
36	1.64%	44.71%	48.79%	50.21%	49.53%	47.94%	48.17%	47.43%	35.64%
37	1.61%	44.26%	48.81%	49.97%	49.45%	48.49%	48.17%	48.04%	35.80%
38	1.70%	44.42%	48.08%	50.13%	50.11%	47.75%	48.51%	47.16%	35.51%
39	1.52%	44.33%	48.58%	49.52%	49.74%	48.19%	48.29%	47.78%	35.46%
40	1.47%	44.46%	48.54%	49.50%	49.94%	47.66%	48.05%	47.96%	35.50%
41	1.44%	44.83%	48.32%	49.47%	49.99%	47.56%	48.38%	47.32%	35.13%
42	1.54%	44.50%	48.33%	49.97%	50.13%	47.94%	48.10%	47.37%	34.81%
43	1.58%	44.82%	48.29%	49.56%	49.66%	47.62%	47.69%	47.21%	37.80%
44	1.52%	44.83%	48.47%	49.62%	49.76%	47.20%	48.41%	47.24%	43.28%
45	1.58%	44.88%	48.56%	49.27%	49.89%	47.32%	48.15%	47.23%	44.19%
46	1.66%	44.62%	48.44%	50.22%	49.84%	48.28%	48.42%	47.75%	43.07%
47	1.67%	44.58%	47.86%	49.85%	49.96%	48.22%	48.46%	47.60%	40.01%
48	1.64%	44.88%	48.55%	49.76%	50.27%	48.77%	47.69%	47.50%	39.13%
49	1.72%	44.54%	47.84%	49.17%	49.50%	48.28%	48.18%	47.63%	39.63%
50	1.75%	44.57%	47.90%	49.44%	49.60%	48.18%	48.14%	48.23%	39.94%

Table 14 (Continued)

Table 15 lists the AEs obtained at the end of each iteration of the AlphaZero algorithm with different c_{puct}'s. The tested values of c_{puct} included 0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, and 16.

	0.0625	**0.125**	**0.25**	**0.5**	**1**	**2**	**4**	**8**	**16**
0	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692
1	0.5144	0.1938	0.3279	0.3620	0.2003	0.2126	0.2059	0.2610	0.2312
2	0.2567	0.2341	0.3296	0.2607	0.1922	0.2090	0.0997	0.3042	0.2725
3	0.2290	0.3704	0.2412	0.2003	0.1575	0.2568	0.3638	0.1858	0.1037
4	0.2659	0.2129	0.0550	0.0717	0.4681	0.2719	0.2383	0.2122	0.3679
5	0.1781	0.1562	0.0185	0.1120	0.0047	0.1043	0.2475	0.3401	0.3464
6	0.2823	0.0690	0.0883	0.0622	0.2048	0.2375	0.2718	0.1773	0.1537
7	0.2351	0.1137	0.1083	0.0123	0.0125	0.1877	0.1727	0.2130	0.1429
8	0.2640	0.0045	0.0826	0.0542	0.0574	0.1987	0.3956	0.3053	0.1938
9	0.2934	0.0749	0.0934	0.0872	0.0275	0.1126	0.3546	0.2808	0.1450
10	0.2610	0.0197	0.0344	0.0616	0.0569	0.0959	0.2170	0.1702	0.1095
11	0.2715	0.1093	0.0038	0.0233	0.0383	0.1447	0.1266	0.2663	0.0357
12	0.2550	0.0564	0.0441	0.1122	0.0524	0.2826	0.1344	0.0619	0.4249
13	0.2714	0.0188	0.0489	0.0466	0.0639	0.1186	0.2334	0.2448	0.3716
14	0.2714	0.0030	0.0327	0.0172	0.0299	0.0236	0.3397	0.2872	0.2420
15	0.2714	0.0217	0.0131	0.0153	0.0287	0.0238	0.1650	0.3196	0.2121
16	0.2714	0.0507	0.1515	0.0433	0.0415	0.1430	0.1197	0.2593	0.2591
17	0.2714	0.0336	0.2014	0.0836	0.0300	0.0674	0.1984	0.3015	0.2112
18	0.2714	0.0433	0.0672	0.0501	0.0898	0.0829	0.1886	0.2481	0.2414
19	0.2714	0.0071	0.0001	0.0430	0.0144	0.1881	0.0573	0.2132	0.2024
20	0.2714	0.0197	0.0434	0.0335	0.0367	0.0035	0.2853	0.0828	0.1856
21	0.2714	0.0210	0.0479	0.0571	0.0780	0.0828	0.1216	0.1681	0.1593
22	0.2714	0.0244	0.1517	0.0149	0.0153	0.1362	0.1460	0.3217	0.2878
23	0.2714	0.0412	0.0056	0.0005	0.0964	0.0080	0.2015	0.1013	0.2260
24	0.2714	0.0404	0.0235	0.0938	0.2038	0.0153	0.0239	0.1623	0.1873
25	0.2714	0.0253	0.0910	0.0901	0.0230	0.0034	0.0139	0.1396	0.2989

Table 15: Detailed AEs for applying AlphaZero on 2×4 CDC with different c_{puct}.

	0.0625	0.125	0.25	0.5	1	2	4	8	16
26	0.2714	0.0370	0.0114	0.0021	0.0226	0.0273	0.1507	0.2928	0.2623
27	0.2714	0.0451	0.0048	0.0114	0.0475	0.0889	0.1310	0.2612	0.2467
28	0.2714	0.0150	0.0210	0.0053	0.0567	0.1063	0.1791	0.2707	0.2631
29	0.2714	0.0296	0.0098	0.0312	0.0692	0.0752	0.1758	0.2073	0.2439
30	0.2714	0.0218	0.0211	0.0212	0.0095	0.0674	0.1110	0.2650	0.2963
31	0.2714	0.0236	0.0250	0.0072	0.0372	0.0302	0.1024	0.2275	0.1962
32	0.2714	0.0058	0.0309	0.0102	0.0483	0.0379	0.1468	0.2537	0.2267
33	0.2714	0.0240	0.0346	0.0135	0.0083	0.1307	0.2009	0.2515	0.2079
34	0.2714	0.0238	0.0151	0.0005	0.0107	0.0430	0.1295	0.2283	0.2512
35	0.2714	0.0526	0.0008	0.0167	0.0027	0.0064	0.1363	0.2027	0.2758
36	0.2714	0.0316	0.0231	0.0203	0.0180	0.0406	0.1171	0.2068	0.2361
37	0.2714	0.0135	0.0196	0.0600	0.0073	0.0558	0.1304	0.1622	0.2407
38	0.2714	0.0482	0.0256	0.0307	0.0359	0.0698	0.1304	0.1647	0.2515
39	0.2714	0.0368	0.0099	0.0251	0.0514	0.0705	0.1044	0.2487	0.2939
40	0.2714	0.0167	0.0167	0.0002	0.0400	0.0815	0.1682	0.2333	0.2991
41	0.2714	0.0156	0.0177	0.0065	0.0130	0.0466	0.1222	0.2521	0.2361
42	0.2714	0.0200	0.0309	0.0120	0.0411	0.0821	0.1496	0.2050	0.2478
43	0.2714	0.0053	0.0090	0.0207	0.0559	0.0405	0.1078	0.2206	0.2216
44	0.2714	0.0374	0.0195	0.0210	0.0072	0.0573	0.1038	0.2688	0.2606
45	0.2714	0.0011	0.0164	0.0377	0.0032	0.0582	0.1209	0.2444	0.2733
46	0.2714	0.0296	0.0106	0.0126	0.0297	0.1035	0.1554	0.1877	0.2833
47	0.2714	0.0272	0.0367	0.0152	0.0092	0.0386	0.1717	0.2450	0.2971
48	0.2714	0.0012	0.0127	0.0262	0.0079	0.0612	0.1176	0.2996	0.2748
49	0.2714	0.0261	0.0173	0.0085	0.0094	0.0693	0.1425	0.1891	0.2448
50	0.2714	0.0311	0.0021	0.0031	0.0368	0.0761	0.1360	0.2239	0.2096

Table 15 (Continued)

Table 16 lists the cumulative number of distinct seen positions obtained at the end of each iteration of the AlphaZero algorithm with different c_{puct}'s. The tested values of c_{puct} included 0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, and 16.

	0.0625	**0.125**	**0.25**	**0.5**	**1**	**2**	**4**	**8**	**16**
0	0	0	0	0	0	0	0	0	0
1	54,451	54,921	54,221	51,765	51,133	51,577	51,927	53,367	54,131
2	68,271	79,811	78,933	76,333	74,665	75,045	75,481	76,445	77,385
3	70,699	95,035	93,453	91,033	89,717	90,131	91,741	91,981	92,469
4	71,071	105,291	103,061	99,725	100,905	101,271	102,513	103,049	102,249
5	71,167	112,023	109,673	105,333	108,389	110,275	111,341	111,289	110,873
6	71,207	116,663	113,905	109,825	114,181	117,087	118,425	118,109	117,241
7	71,207	119,587	116,349	113,081	118,133	123,107	124,129	123,449	122,233
8	71,207	121,499	118,469	116,145	121,063	127,403	128,625	128,329	126,145
9	71,207	122,819	119,949	118,041	123,479	131,067	132,941	131,933	129,625
10	71,207	123,715	121,389	119,769	125,503	133,989	136,645	135,261	133,125
11	71,207	124,339	122,265	120,921	127,295	136,693	139,685	138,561	135,725
12	71,207	125,087	122,913	122,169	128,847	138,889	142,309	141,353	138,309
13	71,207	126,199	123,493	123,197	130,179	140,813	144,877	143,613	140,721
14	71,207	126,699	123,997	124,117	131,451	142,509	147,105	145,653	142,953
15	71,207	127,179	124,437	125,037	132,391	143,793	149,169	147,677	144,537
16	71,207	127,403	124,857	125,825	133,531	145,073	150,861	149,477	146,153
17	71,207	127,731	125,261	126,497	134,407	146,241	152,033	150,965	147,617
18	71,207	128,267	125,773	127,089	135,311	147,309	153,457	152,533	149,029
19	71,207	128,727	126,093	127,777	136,003	148,149	154,441	153,853	150,249
20	71,207	128,903	126,529	128,381	136,831	148,933	155,565	155,073	151,661
21	71,207	129,099	126,833	128,761	137,515	149,685	156,701	156,113	152,793
22	71,207	129,323	127,321	129,341	137,943	150,349	157,557	157,201	153,977
23	71,207	129,643	127,665	129,685	138,599	151,073	158,393	158,181	154,873
24	71,207	129,763	128,097	130,293	139,231	151,709	159,393	159,181	155,857
25	71,207	129,979	128,345	130,885	139,723	152,517	160,165	160,085	156,669

Table 16: Detailed cumulative number of distinct seen positions for applying AlphaZero on 2×4 CDC with different c_{puct}.

	0.0625	0.125	0.25	0.5	1	2	4	8	16
26	71,207	130,115	128,537	131,293	140,079	153,249	160,917	160,777	157,465
27	71,207	130,271	128,869	131,781	140,531	153,813	161,381	161,529	158,149
28	71,207	130,327	129,213	132,161	141,059	154,341	162,069	162,429	158,865
29	71,207	130,415	129,429	132,477	141,403	154,769	162,733	163,009	159,509
30	71,207	130,519	129,709	132,885	141,763	155,425	163,333	163,817	160,237
31	71,207	130,551	129,901	133,181	142,139	155,993	163,777	164,561	161,101
32	71,207	130,711	130,089	133,525	142,667	156,457	164,489	165,317	161,677
33	71,207	130,783	130,449	133,857	143,055	156,969	164,881	165,825	162,321
34	71,207	130,967	130,569	134,201	143,405	157,425	165,309	166,261	162,977
35	71,207	131,047	130,801	134,505	143,745	157,777	165,781	166,853	163,525
36	71,207	131,111	131,001	134,793	143,973	158,137	166,369	167,341	164,049
37	71,207	131,191	131,161	135,089	144,337	158,641	166,649	167,785	164,597
38	71,207	131,271	131,353	135,425	144,689	158,953	167,045	168,209	165,145
39	71,207	131,351	131,529	135,625	145,089	159,329	167,325	168,537	165,605
40	71,207	131,407	131,729	135,845	145,453	159,621	167,673	168,873	165,973
41	71,207	131,527	131,889	136,145	145,713	160,029	167,965	169,377	166,237
42	71,207	131,655	131,977	136,413	146,025	160,301	168,325	169,809	166,561
43	71,207	131,735	132,241	136,621	146,313	160,653	168,669	170,317	166,889
44	71,207	131,863	132,401	136,757	146,685	160,853	168,905	170,557	167,237
45	71,207	131,927	132,577	137,053	147,001	161,133	169,229	170,957	167,557
46	71,207	131,991	132,741	137,233	147,369	161,317	169,537	171,309	167,997
47	71,207	132,071	132,969	137,493	147,577	161,653	169,849	171,629	168,365
48	71,207	132,095	133,129	137,821	147,913	161,921	170,081	171,953	168,753
49	71,207	132,135	133,237	138,049	148,121	162,297	170,389	172,233	169,101
50	71,207	132,183	133,429	138,285	148,369	162,501	170,605	172,481	169,453

Table 16 (Continued)

Table 17 lists the win rates obtained at the end of each iteration of the AlphaZero algorithm with different Dirichlet α's. The tested values of Dirichlet α included 0.03, 0.15, 0.3, 0.75, 1.5, 3, 6, 9, and 12. The 95% confidence bounds of the win rates were less than 0.98%.

	0.03	**0.15**	**0.3**	**0.75**	**1.5**	**3**	**6**	**9**	**12**
0	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	2.91%	2.30%	2.06%	1.97%	2.94%	2.38%	2.25%	2.77%	2.89%
2	6.41%	8.03%	2.65%	8.01%	8.61%	9.07%	9.63%	10.07%	10.03%
3	12.62%	16.03%	8.06%	13.26%	16.87%	12.70%	19.20%	13.89%	18.48%
4	25.13%	17.75%	30.47%	23.38%	24.60%	22.31%	27.68%	19.28%	26.61%
5	31.89%	25.40%	34.96%	33.95%	35.11%	35.32%	38.21%	29.65%	35.26%
6	38.84%	33.45%	38.12%	42.21%	41.72%	43.22%	41.08%	42.36%	43.83%
7	39.69%	40.86%	44.02%	44.50%	43.63%	44.94%	44.63%	45.65%	46.24%
8	45.44%	42.80%	46.02%	46.41%	46.90%	45.52%	46.93%	46.75%	46.41%
9	46.89%	45.49%	46.87%	46.45%	47.22%	47.37%	46.63%	46.82%	47.09%
10	47.04%	46.97%	46.92%	47.08%	48.07%	47.33%	47.93%	47.61%	48.24%
11	46.94%	47.15%	46.96%	47.43%	47.67%	47.69%	47.56%	47.78%	47.65%
12	47.54%	48.21%	47.56%	47.40%	47.46%	48.12%	47.67%	47.55%	47.57%
13	47.76%	47.68%	48.25%	47.49%	47.52%	48.24%	48.29%	48.19%	47.78%
14	48.16%	48.09%	47.79%	47.14%	48.05%	47.87%	48.10%	48.05%	47.83%
15	48.60%	48.73%	48.16%	47.11%	47.83%	48.16%	48.60%	47.92%	48.61%
16	48.58%	47.68%	48.55%	47.48%	47.68%	48.55%	47.76%	47.93%	48.76%
17	47.63%	48.40%	48.44%	48.31%	47.87%	48.27%	48.20%	48.37%	47.68%
18	47.49%	47.67%	48.24%	47.34%	48.69%	48.55%	47.73%	48.19%	48.28%
19	47.33%	48.84%	47.66%	47.71%	48.89%	47.89%	47.75%	48.66%	48.34%
20	48.28%	48.15%	48.83%	47.82%	48.11%	48.58%	47.81%	47.88%	48.19%
21	47.83%	48.02%	47.67%	47.98%	46.91%	48.06%	48.05%	48.07%	48.80%
22	47.64%	48.86%	48.08%	47.93%	48.41%	47.70%	47.75%	48.08%	48.63%
23	47.93%	47.86%	47.40%	48.10%	49.22%	48.61%	48.01%	48.24%	48.55%
24	48.98%	49.53%	47.89%	48.14%	48.74%	48.26%	48.57%	47.28%	48.47%
25	47.56%	49.72%	48.82%	48.42%	48.84%	47.97%	48.48%	48.18%	48.89%

Table 17: Detailed win rates for applying AlphaZero on 2×4 CDC with different Dirichlet α.

	0.03	0.15	0.3	0.75	1.5	3	6	9	12
26	47.67%	48.50%	47.85%	48.13%	48.41%	48.41%	47.60%	47.27%	48.86%
27	47.49%	49.26%	47.81%	47.72%	49.26%	48.71%	49.53%	48.03%	48.78%
28	48.12%	49.10%	48.02%	48.44%	50.03%	48.24%	48.79%	48.29%	49.25%
29	48.41%	49.25%	49.47%	47.96%	50.46%	49.01%	48.95%	48.71%	49.17%
30	47.35%	49.73%	48.82%	48.00%	51.02%	48.44%	48.68%	49.59%	49.26%
31	49.32%	49.56%	49.09%	48.63%	49.85%	50.28%	50.18%	49.90%	49.16%
32	48.35%	48.97%	48.14%	47.79%	50.37%	49.57%	48.94%	49.78%	49.25%
33	48.49%	49.02%	48.21%	48.22%	50.26%	50.72%	48.98%	49.63%	49.33%
34	48.05%	50.32%	48.56%	47.29%	49.08%	49.69%	49.00%	50.06%	49.48%
35	46.87%	50.16%	48.73%	47.57%	49.67%	50.21%	48.72%	49.60%	49.46%
36	48.53%	50.15%	48.40%	47.57%	49.53%	49.85%	49.03%	49.74%	49.02%
37	48.56%	49.79%	48.20%	48.76%	49.45%	49.73%	49.12%	50.03%	49.06%
38	48.70%	49.70%	49.79%	48.07%	50.11%	49.85%	49.83%	50.21%	49.11%
39	49.15%	49.54%	48.67%	48.26%	49.74%	50.09%	49.12%	50.21%	49.89%
40	49.12%	49.21%	50.01%	49.39%	49.94%	50.07%	49.23%	49.74%	49.69%
41	47.71%	49.40%	49.58%	49.32%	49.99%	49.95%	50.08%	49.76%	49.38%
42	48.89%	49.53%	50.25%	48.07%	50.13%	50.15%	49.18%	49.23%	49.61%
43	48.72%	49.41%	50.27%	47.32%	49.66%	49.61%	49.15%	50.03%	49.30%
44	49.07%	49.41%	50.51%	48.31%	49.76%	50.03%	49.81%	49.98%	49.57%
45	48.33%	49.76%	49.94%	48.86%	49.89%	50.43%	49.26%	50.16%	49.60%
46	48.76%	49.61%	50.76%	49.01%	49.84%	49.58%	49.33%	50.19%	49.66%
47	48.97%	49.62%	50.32%	48.81%	49.96%	49.67%	48.83%	49.85%	49.83%
48	48.24%	49.56%	49.94%	49.53%	50.27%	49.38%	49.19%	49.86%	49.28%
49	48.44%	49.43%	49.61%	49.44%	49.50%	49.42%	49.28%	49.75%	49.23%
50	48.44%	49.48%	49.94%	49.27%	49.60%	49.16%	50.09%	49.70%	49.44%

Table 17 (Continued)

Table 18 lists the AEs obtained at the end of each iteration of the AlphaZero algorithm with different Dirichlet α's. The tested values of Dirichlet α's included 0.03, 0.15, 0.3, 0.75, 1.5, 3, 6, 9, and 12.

	0.03	**0.15**	**0.3**	**0.75**	**1.5**	**3**	**6**	**9**	**12**
0	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692
1	0.3176	0.1900	0.0455	0.2075	0.2003	0.1777	0.3399	0.4109	0.3274
2	0.4152	0.3386	0.2271	0.1783	0.1922	0.2876	0.0054	0.2228	0.2594
3	0.2615	0.2273	0.0247	0.1737	0.1575	0.2751	0.2891	0.3383	0.3031
4	0.2459	0.3333	0.2158	0.2320	0.4681	0.0890	0.1304	0.1609	0.0148
5	0.1009	0.0893	0.0377	0.1889	0.0047	0.2006	0.2438	0.0639	0.0984
6	0.1274	0.1367	0.1523	0.2743	0.2048	0.0155	0.0391	0.0541	0.0053
7	0.1344	0.2152	0.1216	0.1295	0.0125	0.0543	0.0656	0.0682	0.0131
8	0.1232	0.2791	0.0147	0.1140	0.0574	0.0531	0.0029	0.0282	0.0496
9	0.1256	0.0983	0.0291	0.0112	0.0275	0.0349	0.0611	0.0215	0.1571
10	0.1674	0.0025	0.0216	0.1155	0.0569	0.0357	0.0387	0.0628	0.0017
11	0.0774	0.0110	0.0485	0.0007	0.0383	0.0097	0.0017	0.0067	0.0379
12	0.0806	0.0152	0.0005	0.0685	0.0524	0.0139	0.0748	0.0381	0.0095
13	0.2269	0.1466	0.0164	0.0962	0.0639	0.0177	0.0148	0.0180	0.1043
14	0.0947	0.0690	0.1212	0.1920	0.0299	0.0115	0.0077	0.0538	0.0249
15	0.0931	0.0888	0.0362	0.0034	0.0287	0.0090	0.0246	0.0295	0.0136
16	0.1032	0.1420	0.0540	0.1037	0.0415	0.0604	0.0617	0.0038	0.0558
17	0.0031	0.0284	0.1524	0.0676	0.0300	0.0128	0.0047	0.0534	0.0034
18	0.0386	0.1227	0.0202	0.0236	0.0898	0.0850	0.1761	0.0364	0.0017
19	0.1721	0.1533	0.0798	0.0145	0.0144	0.0350	0.0341	0.0792	0.0190
20	0.0807	0.2068	0.0849	0.0058	0.0367	0.0296	0.0228	0.0515	0.0396
21	0.0587	0.0011	0.1320	0.0053	0.0780	0.0076	0.0070	0.2031	0.0400
22	0.0027	0.0296	0.0002	0.1659	0.0153	0.0034	0.0108	0.0183	0.0692
23	0.3768	0.1583	0.1300	0.1213	0.0964	0.1089	0.0413	0.0444	0.0502
24	0.0245	0.0320	0.0343	0.0647	0.2038	0.0235	0.0384	0.0044	0.0156
25	0.0783	0.0764	0.1228	0.0202	0.0230	0.0185	0.0146	0.0046	0.0386

Table 18: Detailed AEs for applying AlphaZero on 2×4 CDC with different Dirichlet α.

	0.03	**0.15**	**0.3**	**0.75**	**1.5**	**3**	**6**	**9**	**12**
26	0.0755	0.0215	0.0632	0.0520	0.0226	0.0427	0.0093	0.0113	0.0170
27	0.0422	0.0576	0.0424	0.0144	0.0475	0.0436	0.0259	0.0091	0.0132
28	0.0672	0.0458	0.0361	0.0313	0.0567	0.0323	0.0160	0.0400	0.0085
29	0.0163	0.0703	0.0527	0.0349	0.0692	0.0370	0.0282	0.0056	0.0143
30	0.0830	0.0839	0.0862	0.0378	0.0095	0.0016	0.0082	0.0061	0.0189
31	0.0489	0.0640	0.0751	0.0464	0.0372	0.0180	0.0194	0.0250	0.0033
32	0.0561	0.0697	0.0375	0.0451	0.0483	0.0099	0.0121	0.0052	0.0006
33	0.0626	0.0586	0.0256	0.0468	0.0083	0.0376	0.0063	0.0688	0.0091
34	0.0600	0.0395	0.0347	0.0478	0.0107	0.0027	0.0038	0.0020	0.0164
35	0.0735	0.0828	0.0694	0.0254	0.0027	0.0049	0.0388	0.0245	0.0197
36	0.0809	0.0928	0.0573	0.0555	0.0180	0.0003	0.0166	0.0169	0.0542
37	0.0929	0.0671	0.0412	0.0661	0.0073	0.0183	0.0342	0.0291	0.0278
38	0.0688	0.0350	0.0225	0.0516	0.0359	0.0375	0.0034	0.0060	0.0247
39	0.0655	0.0225	0.0302	0.0202	0.0514	0.0374	0.0155	0.0320	0.0095
40	0.0422	0.0740	0.0569	0.0138	0.0400	0.0086	0.0081	0.0118	0.0028
41	0.0804	0.0736	0.0503	0.0458	0.0130	0.0531	0.0011	0.0181	0.0194
42	0.0341	0.0609	0.0701	0.0405	0.0411	0.0532	0.0540	0.0667	0.0212
43	0.0609	0.0902	0.1298	0.0368	0.0559	0.0181	0.0038	0.0238	0.0223
44	0.0880	0.0825	0.0555	0.0612	0.0072	0.0322	0.0138	0.0279	0.0187
45	0.0588	0.0321	0.0493	0.0962	0.0032	0.0475	0.0193	0.0167	0.0324
46	0.0901	0.0731	0.0081	0.0401	0.0297	0.0306	0.0114	0.0507	0.0016
47	0.0675	0.0429	0.0069	0.0780	0.0092	0.0183	0.0247	0.0406	0.0240
48	0.0671	0.0752	0.0790	0.0471	0.0079	0.0189	0.0103	0.0219	0.0070
49	0.0460	0.0825	0.0560	0.0755	0.0094	0.0126	0.0354	0.0206	0.0107
50	0.0581	0.0712	0.0579	0.0511	0.0368	0.0071	0.0134	0.0498	0.0068

Table 18 (Continued)

Table 19 lists the cumulative number of distinct seen positions obtained at the end of each iteration of the AlphaZero algorithm with different Dirichlet α's. The tested values of Dirichlet α included 0.03, 0.15, 0.3, 0.75, 1.5, 3, 6, 9, and 12.

	0.03	**0.15**	**0.3**	**0.75**	**1.5**	**3**	**6**	**9**	**12**
0	0	0	0	0	0	0	0	0	0
1	50,611	51,729	51,287	51,681	51,133	50,229	50,325	51,187	49,749
2	73,949	74,483	73,479	75,245	74,665	74,097	73,563	73,669	72,721
3	89,301	89,365	88,815	89,709	89,717	89,661	88,239	87,773	86,941
4	100,055	100,237	100,209	99,433	100,905	99,749	98,523	97,581	96,921
5	108,235	108,641	108,425	107,209	108,389	106,853	106,119	104,913	104,377
6	115,127	115,169	114,161	112,793	114,181	112,473	111,079	110,249	109,317
7	120,403	119,909	119,081	117,029	118,133	115,877	114,915	114,305	112,689
8	124,559	123,665	123,013	120,385	121,063	118,981	118,383	117,073	115,353
9	128,227	127,265	126,085	123,609	123,479	120,781	120,027	119,193	117,433
10	130,963	130,117	128,585	125,861	125,503	122,561	121,595	120,693	118,921
11	133,111	132,377	130,721	128,001	127,295	124,213	123,167	122,065	120,269
12	135,159	134,281	132,769	129,717	128,847	125,717	124,455	123,573	121,717
13	136,803	136,025	134,377	131,233	130,179	126,801	125,595	124,665	122,821
14	138,463	137,393	135,881	132,601	131,451	127,797	126,459	125,705	123,945
15	139,711	138,829	137,129	133,953	132,391	128,777	127,315	126,813	124,957
16	141,203	140,121	138,057	135,045	133,531	129,657	128,147	127,701	125,917
17	142,459	141,209	139,325	135,985	134,407	130,285	128,743	128,645	126,889
18	143,563	142,257	140,441	137,089	135,311	131,065	129,471	129,341	127,737
19	144,519	143,101	141,469	137,993	136,003	131,729	130,095	129,989	128,441
20	145,379	144,057	142,421	138,753	136,831	132,333	130,591	130,645	129,037
21	146,251	144,901	143,125	139,545	137,515	132,885	131,227	131,265	129,561
22	147,191	145,717	143,925	140,161	137,943	133,549	131,755	131,749	130,101
23	147,951	146,481	144,733	140,861	138,599	134,217	132,163	132,389	130,581
24	148,675	147,169	145,541	141,585	139,231	134,817	132,687	132,897	131,117
25	149,371	147,917	146,277	142,545	139,723	135,377	133,299	133,425	131,609

Table 19: Detailed cumulative number of distinct seen positions for applying AlphaZero on 2×4 CDC with different Dirichlet α.

	0.03	0.15	0.3	0.75	1.5	3	6	9	12
26	149,963	148,625	146,893	143,153	140,079	135,957	133,691	133,849	132,025
27	150,543	149,241	147,329	143,677	140,531	136,369	134,091	134,153	132,333
28	151,319	149,841	147,849	144,177	141,059	136,793	134,623	134,505	132,577
29	151,863	150,369	148,573	144,613	141,403	137,121	135,063	134,925	132,945
30	152,575	150,917	149,129	144,989	141,763	137,601	135,643	135,233	133,257
31	153,135	151,401	149,493	145,437	142,139	138,037	135,919	135,761	133,633
32	153,559	151,909	149,833	145,869	142,667	138,557	136,229	136,209	133,977
33	153,999	152,417	150,329	146,285	143,055	138,893	136,613	136,505	134,241
34	154,543	152,817	150,817	146,653	143,405	139,189	136,901	136,805	134,569
35	154,967	153,241	151,233	147,073	143,745	139,405	137,293	137,117	134,885
36	155,483	153,689	151,649	147,345	143,973	139,701	137,589	137,365	135,205
37	155,843	154,117	152,057	147,737	144,337	140,109	138,081	137,705	135,521
38	156,067	154,553	152,553	148,233	144,689	140,377	138,341	138,001	135,753
39	156,403	154,909	152,869	148,533	145,089	140,593	138,637	138,165	135,993
40	156,767	155,381	153,233	148,905	145,453	140,921	138,889	138,413	136,253
41	157,115	155,785	153,529	149,261	145,713	141,241	139,145	138,821	136,577
42	157,471	156,145	153,729	149,513	146,025	141,537	139,437	139,173	136,817
43	157,735	156,441	153,977	149,801	146,313	141,825	139,701	139,425	137,121
44	157,959	156,737	154,313	150,245	146,685	142,065	140,021	139,733	137,321
45	158,407	157,041	154,529	150,505	147,001	142,421	140,373	140,005	137,537
46	158,663	157,185	154,745	150,821	147,369	142,705	140,709	140,357	137,705
47	158,959	157,493	155,109	151,117	147,577	143,033	140,941	140,637	137,909
48	159,199	157,785	155,381	151,445	147,913	143,273	141,141	140,885	138,057
49	159,483	157,989	155,749	151,701	148,121	143,529	141,341	141,077	138,249
50	159,747	158,185	156,061	151,981	148,369	143,745	141,501	141,245	138,441

Table 19 (Continued)

Table 20 lists the win rates obtained at the end of each iteration of the AlphaZero algorithm with different Dirichlet ε's. The tested values of Dirichlet ε included 0, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, and 0.5. The 95% confidence bounds of the win rates were less than 0.98%.

	0	**0.1**	**0.15**	**0.2**	**0.25**	**0.3**	**0.35**	**0.4**	**0.5**
0	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	1.89%	2.67%	2.29%	2.48%	2.94%	2.38%	2.76%	2.25%	2.42%
2	11.47%	11.22%	13.56%	10.56%	8.61%	7.45%	10.47%	13.27%	7.40%
3	23.73%	19.06%	21.89%	23.70%	16.87%	16.51%	14.19%	26.71%	15.03%
4	30.23%	30.48%	31.82%	32.58%	24.60%	32.01%	29.86%	22.32%	28.13%
5	38.95%	41.04%	39.56%	38.43%	35.11%	34.47%	36.17%	26.47%	34.15%
6	43.70%	44.58%	42.69%	42.96%	41.72%	41.41%	42.87%	39.09%	40.37%
7	46.62%	46.77%	44.94%	45.77%	43.63%	43.73%	43.66%	41.34%	44.23%
8	47.48%	46.69%	45.63%	46.50%	46.90%	44.42%	45.87%	44.85%	45.60%
9	47.70%	47.23%	47.60%	46.94%	47.22%	47.90%	46.90%	44.90%	46.70%
10	47.84%	47.98%	48.16%	47.65%	48.07%	47.36%	47.62%	47.39%	47.16%
11	47.92%	47.52%	48.66%	47.81%	47.67%	47.45%	47.52%	47.34%	47.24%
12	48.01%	47.97%	47.81%	47.85%	47.46%	48.11%	47.52%	47.87%	46.49%
13	48.42%	47.54%	48.71%	48.37%	47.52%	48.06%	48.08%	47.73%	47.77%
14	48.17%	48.07%	47.69%	48.97%	48.05%	48.68%	47.67%	48.59%	48.57%
15	48.60%	48.20%	48.47%	48.58%	47.83%	48.26%	48.03%	47.60%	46.52%
16	47.49%	48.11%	48.16%	48.27%	47.68%	48.14%	47.92%	47.80%	48.52%
17	47.78%	49.17%	48.00%	48.17%	47.87%	48.10%	48.07%	49.10%	48.45%
18	47.49%	49.55%	47.84%	48.39%	48.69%	47.91%	48.41%	48.31%	48.52%
19	47.33%	48.78%	48.39%	47.86%	48.89%	48.02%	48.41%	47.95%	47.73%
20	46.85%	47.93%	47.63%	48.17%	48.11%	47.66%	47.40%	48.41%	48.17%
21	47.36%	48.86%	47.84%	48.61%	46.91%	47.97%	48.40%	48.19%	48.40%
22	47.39%	47.35%	48.58%	48.32%	48.41%	48.20%	48.90%	48.50%	48.04%
23	47.51%	47.83%	48.30%	48.34%	49.22%	48.50%	47.90%	47.26%	48.89%
24	47.47%	47.15%	47.99%	47.43%	48.74%	48.20%	48.70%	49.33%	48.40%
25	47.40%	48.20%	48.27%	47.91%	48.84%	47.99%	49.04%	48.59%	48.59%

Table 20: Detailed win rates for applying AlphaZero on 2×4 CDC with different Dirichlet ε.

	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.5
26	46.95%	47.51%	48.11%	48.39%	48.41%	48.07%	49.16%	49.37%	49.40%
27	47.48%	47.76%	47.98%	47.89%	49.26%	48.91%	49.75%	49.78%	48.36%
28	46.82%	48.10%	48.86%	48.55%	50.03%	48.74%	48.88%	49.51%	48.52%
29	46.90%	48.20%	48.22%	48.15%	50.46%	48.99%	48.94%	49.52%	48.27%
30	47.49%	48.00%	48.88%	49.32%	51.02%	48.44%	49.51%	49.76%	47.76%
31	47.49%	47.94%	49.63%	49.20%	49.85%	48.47%	49.97%	49.50%	48.77%
32	47.52%	47.62%	49.65%	48.82%	50.37%	48.73%	49.49%	49.12%	47.26%
33	47.54%	47.91%	50.07%	48.81%	50.26%	48.46%	49.91%	49.52%	47.77%
34	47.59%	47.89%	49.79%	48.97%	49.08%	48.63%	49.58%	50.17%	47.17%
35	47.46%	48.04%	49.43%	49.47%	49.67%	48.82%	49.59%	49.64%	48.31%
36	47.52%	49.43%	48.73%	49.32%	49.53%	48.35%	49.89%	49.26%	48.75%
37	47.37%	49.12%	48.39%	49.26%	49.45%	47.88%	49.38%	49.97%	47.56%
38	47.50%	48.96%	48.07%	49.43%	50.11%	48.35%	49.05%	50.20%	48.05%
39	47.46%	49.37%	47.91%	48.90%	49.74%	48.63%	49.78%	50.02%	48.21%
40	47.44%	49.15%	48.15%	49.35%	49.94%	48.25%	49.93%	49.86%	48.67%
41	46.85%	49.35%	47.82%	49.64%	49.99%	48.44%	49.66%	49.44%	47.68%
42	47.67%	49.35%	47.99%	49.20%	50.13%	48.34%	49.43%	49.81%	48.47%
43	47.56%	49.30%	48.55%	49.23%	49.66%	48.45%	49.19%	49.69%	48.69%
44	46.92%	49.26%	47.99%	49.67%	49.76%	49.86%	49.10%	49.35%	48.73%
45	46.90%	49.54%	47.99%	49.53%	49.89%	49.50%	49.54%	49.22%	49.26%
46	47.07%	48.90%	47.85%	48.75%	49.84%	48.40%	49.57%	49.63%	49.41%
47	46.93%	49.36%	48.59%	49.51%	49.96%	49.85%	49.56%	49.99%	49.44%
48	47.47%	47.96%	47.84%	49.81%	50.27%	50.35%	49.70%	50.09%	48.07%
49	47.57%	48.90%	48.66%	48.84%	49.50%	50.10%	49.67%	49.84%	48.49%
50	46.93%	49.21%	48.01%	49.94%	49.60%	49.55%	49.72%	49.86%	48.12%

Table 20 (Continued)

Table 21 lists the AEs obtained at the end of each iteration of the AlphaZero algorithm with different Dirichlet ε's. The tested values of Dirichlet ε's included 0, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, and 0.5.

	0	**0.1**	**0.15**	**0.2**	**0.25**	**0.3**	**0.35**	**0.4**	**0.5**
0	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692
1	0.3389	0.2456	0.3089	0.3721	0.2003	0.2741	0.3129	0.3878	0.1576
2	0.1571	0.2916	0.3805	0.2370	0.1922	0.3033	0.2056	0.2211	0.2161
3	0.2331	0.1883	0.2471	0.2276	0.1575	0.1880	0.0895	0.3399	0.1252
4	0.1972	0.0696	0.3197	0.2710	0.4681	0.2086	0.0830	0.1591	0.3074
5	0.1989	0.1326	0.0957	0.0695	0.0047	0.0398	0.0772	0.2628	0.2892
6	0.0517	0.0362	0.0286	0.0323	0.2048	0.0708	0.0751	0.2008	0.0031
7	0.0059	0.0901	0.0043	0.0464	0.0125	0.0252	0.0096	0.0584	0.0567
8	0.0333	0.0726	0.0155	0.0880	0.0574	0.1135	0.0473	0.2030	0.0849
9	0.0496	0.0629	0.0267	0.1090	0.0275	0.1590	0.0113	0.0383	0.1112
10	0.0491	0.0876	0.0610	0.0073	0.0569	0.0434	0.0956	0.0663	0.1414
11	0.0542	0.0883	0.0171	0.0428	0.0383	0.0603	0.0216	0.1080	0.1332
12	0.0346	0.0556	0.0353	0.0017	0.0524	0.0064	0.0610	0.0282	0.0919
13	0.1088	0.0657	0.0669	0.0881	0.0639	0.0465	0.0208	0.0536	0.0272
14	0.0067	0.0084	0.0042	0.0634	0.0299	0.1754	0.0978	0.2001	0.0737
15	0.0094	0.1390	0.0894	0.0707	0.0287	0.0263	0.1470	0.2867	0.0437
16	0.0433	0.0331	0.0186	0.0392	0.0415	0.1974	0.0179	0.0040	0.0402
17	0.0763	0.0127	0.0187	0.0010	0.0300	0.0181	0.0027	0.0047	0.1788
18	0.0300	0.0717	0.0306	0.0290	0.0898	0.0521	0.0986	0.0489	0.0025
19	0.0554	0.0388	0.0158	0.1253	0.0144	0.0096	0.1484	0.0604	0.1185
20	0.0237	0.0205	0.0199	0.0527	0.0367	0.0493	0.0751	0.0259	0.0815
21	0.0331	0.0311	0.0427	0.0424	0.0780	0.0716	0.0412	0.0268	0.0146
22	0.0188	0.0336	0.0248	0.0158	0.0153	0.0864	0.0365	0.0054	0.1781
23	0.1249	0.0169	0.0547	0.1044	0.0964	0.0018	0.0186	0.0071	0.0392
24	0.0047	0.0396	0.0154	0.0212	0.2038	0.0084	0.0828	0.1138	0.0709
25	0.0393	0.0440	0.0155	0.0075	0.0230	0.0562	0.0457	0.0338	0.0343

Table 21: Detailed AEs for applying AlphaZero on 2×4 CDC with different Dirichlet ε.

	0	**0.1**	**0.15**	**0.2**	**0.25**	**0.3**	**0.35**	**0.4**	**0.5**
26	0.0286	0.0080	0.0053	0.0027	0.0226	0.0258	0.0258	0.0588	0.0935
27	0.0331	0.0166	0.0117	0.0077	0.0475	0.0204	0.0304	0.0750	0.0322
28	0.0163	0.0200	0.0174	0.0128	0.0567	0.0004	0.0655	0.0216	0.0528
29	0.0196	0.0307	0.0247	0.0029	0.0692	0.0797	0.0725	0.0239	0.0273
30	0.0197	0.0016	0.0250	0.0367	0.0095	0.0235	0.0761	0.0431	0.0620
31	0.0263	0.0024	0.0506	0.0231	0.0372	0.0281	0.0407	0.0447	0.0778
32	0.0486	0.0084	0.0012	0.0430	0.0483	0.0663	0.0227	0.0716	0.0866
33	0.0060	0.0162	0.0393	0.0286	0.0083	0.0406	0.0583	0.0569	0.0585
34	0.0084	0.0097	0.0325	0.0390	0.0107	0.0464	0.0369	0.0805	0.0641
35	0.0378	0.0003	0.0430	0.0158	0.0027	0.0476	0.0539	0.0637	0.0625
36	0.0081	0.0140	0.0348	0.0052	0.0180	0.0244	0.0525	0.1041	0.0489
37	0.0273	0.0179	0.0324	0.0279	0.0073	0.0314	0.0655	0.0774	0.0771
38	0.0002	0.0036	0.0024	0.0158	0.0359	0.0260	0.0684	0.0405	0.1025
39	0.0179	0.0008	0.0402	0.0310	0.0514	0.0565	0.0592	0.0260	0.0950
40	0.0194	0.0060	0.0388	0.0124	0.0400	0.0607	0.0762	0.0948	0.0538
41	0.0285	0.0014	0.0061	0.0227	0.0130	0.0233	0.0278	0.0373	0.0702
42	0.0198	0.0264	0.0022	0.0320	0.0411	0.0353	0.0678	0.0639	0.0597
43	0.0139	0.0071	0.0186	0.0334	0.0559	0.0317	0.0688	0.0688	0.0889
44	0.0119	0.0017	0.0171	0.0303	0.0072	0.0233	0.0428	0.0666	0.0753
45	0.0216	0.0151	0.0026	0.0085	0.0032	0.0400	0.0352	0.0633	0.0846
46	0.0020	0.0310	0.0009	0.0120	0.0297	0.0347	0.0546	0.0181	0.0519
47	0.0030	0.0443	0.0059	0.0174	0.0092	0.0436	0.0235	0.0025	0.0725
48	0.0224	0.0072	0.0056	0.0005	0.0079	0.0234	0.0584	0.0700	0.0931
49	0.0003	0.0250	0.0281	0.0251	0.0094	0.0316	0.0914	0.0736	0.0666
50	0.0008	0.0163	0.0567	0.0236	0.0368	0.0300	0.0625	0.0522	0.0816

Table 21 (Continued)

Table 22 lists the cumulative number of distinct seen positions obtained at the end of each iteration of the AlphaZero algorithm with different Dirichlet ε's. The tested values of Dirichlet ε's included 0, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, and 0.5.

	0	**0.1**	**0.15**	**0.2**	**0.25**	**0.3**	**0.35**	**0.4**	**0.5**
0	0	0	0	0	0	0	0	0	0
1	51,123	50,291	51,447	50,993	51,133	51,017	51,263	51,319	50,721
2	73,635	73,105	73,971	73,867	74,665	74,573	74,227	74,175	74,573
3	88,047	87,885	89,579	89,483	89,717	89,541	88,449	89,255	90,689
4	98,147	97,603	99,611	99,175	100,905	100,017	99,709	100,195	101,605
5	105,347	104,711	107,031	106,315	108,389	108,069	107,789	109,001	109,601
6	110,139	109,171	112,367	111,411	114,181	113,761	114,253	114,729	115,877
7	113,391	112,303	115,627	115,287	118,133	118,221	118,517	119,689	120,517
8	115,223	114,635	118,235	118,591	121,063	121,513	122,169	123,801	124,069
9	116,207	116,359	120,243	120,999	123,479	124,657	124,773	126,933	127,489
10	116,935	117,623	121,859	122,987	125,503	126,753	127,269	129,517	130,169
11	117,455	118,707	122,979	124,439	127,295	128,761	129,201	131,765	132,471
12	117,875	119,575	123,955	125,755	128,847	130,089	131,253	133,697	134,595
13	118,139	120,587	125,059	127,243	130,179	131,565	132,805	135,541	136,443
14	118,371	121,311	125,967	128,183	131,451	132,909	134,329	136,869	137,939
15	118,627	122,215	126,767	129,179	132,391	134,309	135,853	138,005	139,269
16	118,899	122,959	127,571	129,975	133,531	135,305	137,033	139,029	140,665
17	119,131	123,707	128,247	130,935	134,407	136,721	138,065	140,061	141,833
18	119,299	124,347	129,111	131,591	135,311	137,817	138,897	141,037	143,041
19	119,539	124,871	129,931	132,339	136,003	138,661	139,749	141,961	143,893
20	119,651	125,275	130,691	132,927	136,831	139,361	140,325	142,885	144,713
21	119,739	125,755	131,383	133,567	137,515	140,005	140,901	143,833	145,585
22	119,839	126,179	131,947	134,063	137,943	140,645	141,729	144,701	146,221
23	119,879	126,563	132,575	134,671	138,599	141,449	142,357	145,257	146,965
24	119,903	126,951	133,103	135,159	139,231	142,073	142,985	145,717	147,677
25	119,911	127,479	133,567	135,703	139,723	142,633	143,649	146,257	148,309

Table 22: Detailed cumulative number of distinct seen positions for applying AlphaZero on 2×4 CDC with different Dirichlet ε.

	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.5
26	119,919	127,791	134,211	136,475	140,079	143,113	144,229	146,777	148,993
27	119,935	128,319	134,707	136,831	140,531	143,717	144,825	147,337	149,565
28	119,943	128,759	135,151	137,315	141,059	144,349	145,157	147,861	150,133
29	120,015	128,983	135,575	137,803	141,403	144,837	145,725	148,433	150,641
30	120,031	129,343	135,975	138,163	141,763	145,425	146,357	148,845	151,217
31	120,055	129,831	136,387	138,559	142,139	145,833	146,733	149,333	151,725
32	120,083	130,233	136,647	139,147	142,667	146,261	147,341	149,813	152,229
33	120,131	130,533	136,975	139,571	143,055	146,557	147,633	150,141	152,725
34	120,139	130,853	137,295	139,995	143,405	146,821	148,193	150,573	153,257
35	120,163	131,097	137,595	140,407	143,745	147,205	148,729	150,989	153,713
36	120,187	131,413	138,003	140,771	143,973	147,789	149,133	151,297	154,209
37	120,219	131,613	138,287	141,123	144,337	148,249	149,501	151,561	154,529
38	120,227	131,861	138,583	141,463	144,689	148,625	149,965	151,897	154,833
39	120,251	132,205	138,719	141,803	145,089	148,821	150,173	152,265	155,205
40	120,291	132,477	139,015	142,051	145,453	149,005	150,509	152,537	155,637
41	120,303	132,741	139,247	142,267	145,713	149,437	150,849	152,809	155,949
42	120,335	132,937	139,551	142,625	146,025	149,705	151,189	153,117	156,393
43	120,351	133,153	139,815	142,837	146,313	149,985	151,501	153,373	156,785
44	120,359	133,277	140,023	143,197	146,685	150,281	151,829	153,705	157,117
45	120,383	133,437	140,303	143,453	147,001	150,577	152,245	154,017	157,325
46	120,391	133,629	140,479	143,781	147,369	150,913	152,521	154,305	157,697
47	120,399	133,877	140,735	144,053	147,577	151,185	152,749	154,621	158,129
48	120,399	134,133	140,951	144,269	147,913	151,481	152,997	154,925	158,449
49	120,407	134,349	141,135	144,465	148,121	151,681	153,265	155,197	158,641
50	120,431	134,517	141,343	144,781	148,369	151,913	153,497	155,429	158,917

Table 22 (Continued)

Table 23 lists the win rates obtained at the end of each iteration of the AlphaZero algorithm with different temperature τ's. The tested values of temperature τ included 0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, and 16. The 95% confidence bounds of the win rates were less than 0.98%.

	0.0625	**0.125**	**0.25**	**0.5**	**1**	**2**	**4**	**8**	**16**
0	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1	1.80%	1.83%	1.74%	1.40%	2.94%	2.37%	2.12%	1.84%	2.51%
2	10.31%	9.01%	7.17%	8.06%	8.61%	9.31%	10.22%	8.73%	4.23%
3	18.31%	16.61%	16.07%	19.84%	16.87%	12.36%	18.39%	12.83%	8.26%
4	28.83%	32.31%	35.00%	32.61%	24.60%	25.39%	24.14%	18.77%	11.00%
5	32.89%	42.62%	42.09%	39.03%	35.11%	33.18%	31.29%	20.98%	14.53%
6	38.69%	44.56%	43.49%	41.78%	41.72%	39.28%	36.17%	26.95%	23.20%
7	43.14%	45.39%	44.25%	45.44%	43.63%	41.16%	38.43%	27.64%	24.21%
8	44.16%	44.36%	47.44%	46.66%	46.90%	41.80%	40.21%	25.58%	26.85%
9	45.27%	44.54%	47.02%	46.94%	47.22%	44.23%	41.86%	27.31%	25.95%
10	46.66%	44.49%	47.43%	49.28%	48.07%	45.91%	42.69%	37.20%	28.17%
11	45.61%	44.14%	47.87%	49.04%	47.67%	45.71%	41.90%	40.14%	29.14%
12	45.97%	44.06%	46.23%	45.97%	47.46%	45.82%	42.81%	40.60%	29.10%
13	45.92%	43.97%	47.00%	46.82%	47.52%	46.51%	43.22%	41.31%	31.97%
14	45.89%	46.85%	47.16%	48.04%	48.05%	47.40%	43.39%	42.33%	35.26%
15	45.39%	48.12%	47.17%	47.57%	47.83%	47.06%	43.52%	42.18%	34.62%
16	45.96%	46.14%	47.79%	47.64%	47.68%	47.34%	44.03%	42.35%	33.97%
17	47.72%	48.73%	46.14%	47.74%	47.87%	48.61%	43.20%	43.43%	36.97%
18	47.62%	48.30%	48.09%	48.47%	48.69%	48.45%	44.00%	43.51%	43.77%
19	47.97%	48.83%	47.94%	48.99%	48.89%	48.20%	43.81%	43.55%	43.27%
20	46.70%	46.40%	48.13%	48.00%	48.11%	48.47%	43.94%	44.23%	43.41%
21	46.94%	46.34%	47.71%	47.59%	46.91%	48.48%	46.22%	43.85%	44.17%
22	48.03%	49.10%	47.80%	48.78%	48.41%	48.82%	46.19%	43.66%	44.49%
23	46.95%	47.70%	48.22%	48.14%	49.22%	49.11%	46.45%	44.72%	45.22%
24	47.66%	48.39%	47.79%	48.14%	48.74%	48.45%	47.05%	44.75%	45.98%
25	48.12%	48.79%	47.54%	47.95%	48.84%	48.93%	46.41%	45.39%	46.08%

Table 23: Detailed win rates for applying AlphaZero on 2×4 CDC with different temperature τ.

	0.0625	0.125	0.25	0.5	1	2	4	8	16
26	47.05%	48.94%	47.77%	48.20%	48.41%	48.46%	46.57%	45.48%	46.89%
27	48.35%	49.00%	47.81%	48.91%	49.26%	48.74%	46.02%	45.99%	46.29%
28	47.06%	49.15%	47.44%	48.54%	50.03%	48.97%	46.47%	45.99%	47.02%
29	47.14%	49.10%	47.39%	48.91%	50.46%	49.35%	46.55%	45.92%	47.40%
30	46.38%	48.93%	47.62%	49.45%	51.02%	48.81%	46.50%	46.19%	47.59%
31	46.31%	49.00%	47.59%	49.93%	49.85%	48.50%	46.21%	45.88%	47.20%
32	47.82%	48.96%	47.67%	49.59%	50.37%	48.45%	46.22%	45.90%	47.29%
33	46.08%	48.85%	47.71%	49.39%	50.26%	48.67%	47.02%	45.71%	47.17%
34	46.30%	48.38%	47.50%	49.77%	49.08%	48.67%	46.83%	45.88%	47.39%
35	46.45%	48.69%	47.73%	49.06%	49.67%	48.67%	47.10%	46.00%	47.44%
36	46.35%	48.18%	47.67%	49.40%	49.53%	48.57%	47.45%	45.89%	46.90%
37	46.34%	47.91%	47.79%	49.36%	49.45%	49.10%	47.24%	45.72%	46.90%
38	46.31%	47.93%	47.80%	49.55%	50.11%	48.73%	47.33%	46.00%	46.80%
39	46.36%	48.57%	47.59%	48.73%	49.74%	48.54%	47.67%	46.36%	47.70%
40	47.75%	48.18%	47.74%	48.67%	49.94%	48.48%	47.11%	46.42%	47.04%
41	45.86%	48.28%	47.57%	49.11%	49.99%	48.19%	47.38%	46.41%	47.98%
42	46.08%	47.93%	47.50%	49.35%	50.13%	49.05%	47.97%	46.17%	47.97%
43	46.32%	48.54%	47.64%	49.84%	49.66%	49.11%	48.03%	46.58%	47.19%
44	47.62%	48.87%	48.04%	50.00%	49.76%	48.52%	47.14%	46.57%	47.22%
45	46.37%	49.10%	47.59%	49.45%	49.89%	48.74%	47.03%	46.91%	47.53%
46	46.48%	48.82%	47.14%	48.91%	49.84%	48.73%	46.87%	47.11%	47.58%
47	46.11%	49.28%	47.50%	48.48%	49.96%	48.62%	47.02%	47.24%	47.85%
48	45.97%	48.74%	47.26%	48.94%	50.27%	49.21%	46.93%	47.10%	47.84%
49	47.54%	48.79%	47.86%	48.97%	49.50%	48.92%	47.28%	46.58%	47.94%
50	46.11%	48.92%	47.50%	48.79%	49.60%	48.68%	46.91%	46.80%	47.24%

Table 23 (Continued)

Table 24 lists the AEs obtained at the end of each iteration of the AlphaZero algorithm with different temperature τ's. The tested values of temperature τ included 0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, and 16.

	0.0625	**0.125**	**0.25**	**0.5**	**1**	**2**	**4**	**8**	**16**
0	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692	0.2692
1	0.3424	0.1038	0.0836	0.1444	0.2003	0.2976	0.1796	0.1489	0.3967
2	0.1435	0.1208	0.2043	0.0520	0.1922	0.3558	0.3425	0.3476	0.3693
3	0.1200	0.2245	0.2604	0.3227	0.1575	0.3357	0.4140	0.1878	0.2590
4	0.0601	0.0127	0.0648	0.1974	0.4681	0.3575	0.1562	0.4050	0.2209
5	0.0840	0.0654	0.0250	0.0828	0.0047	0.4408	0.4293	0.3274	0.2738
6	0.0475	0.0666	0.0255	0.0066	0.2048	0.2071	0.3558	0.1373	0.3237
7	0.0631	0.0933	0.0101	0.0124	0.0125	0.3383	0.3720	0.2709	0.2859
8	0.0026	0.0787	0.0403	0.0539	0.0574	0.3266	0.2162	0.4059	0.2598
9	0.0182	0.0499	0.0495	0.0428	0.0275	0.1117	0.4307	0.3835	0.2184
10	0.1037	0.0494	0.0032	0.0240	0.0569	0.2363	0.1806	0.3949	0.3097
11	0.0739	0.0525	0.0791	0.0593	0.0383	0.2137	0.1724	0.1228	0.4123
12	0.0403	0.0272	0.0025	0.0680	0.0524	0.2550	0.2647	0.2401	0.2607
13	0.0732	0.0652	0.0509	0.0433	0.0639	0.2243	0.3421	0.2088	0.1947
14	0.0353	0.0368	0.0637	0.0186	0.0299	0.2277	0.1966	0.2890	0.3504
15	0.0843	0.0054	0.0347	0.0942	0.0287	0.2142	0.2885	0.4164	0.2089
16	0.0907	0.0241	0.0869	0.0793	0.0415	0.1658	0.2294	0.1741	0.1151
17	0.0412	0.0324	0.0411	0.0517	0.0300	0.2042	0.1672	0.2159	0.4485
18	0.0113	0.0195	0.0499	0.0129	0.0898	0.3734	0.2148	0.1718	0.3470
19	0.0401	0.0266	0.0215	0.0512	0.0144	0.1626	0.2589	0.1220	0.2795
20	0.0025	0.0181	0.0024	0.0539	0.0367	0.3306	0.2172	0.2280	0.2291
21	0.0305	0.0891	0.0769	0.0215	0.0780	0.0307	0.3167	0.2356	0.4659
22	0.0292	0.0511	0.0299	0.0267	0.0153	0.0497	0.1257	0.4128	0.1747
23	0.0106	0.0675	0.0699	0.0864	0.0964	0.3504	0.3936	0.2483	0.2703
24	0.0079	0.0756	0.0292	0.0211	0.2038	0.1225	0.2396	0.2643	0.3345
25	0.0362	0.0228	0.0179	0.0581	0.0230	0.1183	0.3498	0.3185	0.2059

Table 24: Detailed AEs for applying AlphaZero on 2×4 CDC with different temperature τ.

	0.0625	0.125	0.25	0.5	1	2	4	8	16
26	0.0104	0.0030	0.0030	0.0385	0.0226	0.2011	0.2328	0.2740	0.2399
27	0.0070	0.0042	0.0014	0.0023	0.0475	0.1924	0.2335	0.2865	0.2269
28	0.0210	0.0022	0.0456	0.0006	0.0567	0.1767	0.2244	0.2372	0.3060
29	0.0083	0.0086	0.0090	0.0386	0.0692	0.1728	0.2332	0.2562	0.2830
30	0.0341	0.0208	0.0000	0.0178	0.0095	0.1867	0.2682	0.2928	0.2709
31	0.0380	0.0100	0.0261	0.0257	0.0372	0.2408	0.2014	0.2586	0.2419
32	0.0029	0.0093	0.0355	0.0071	0.0483	0.1963	0.2430	0.2414	0.2369
33	0.0102	0.0009	0.0021	0.0238	0.0083	0.2449	0.2689	0.2466	0.2557
34	0.0061	0.0263	0.0104	0.0354	0.0107	0.2262	0.2317	0.2720	0.2630
35	0.0017	0.0238	0.0064	0.0008	0.0027	0.1874	0.2439	0.2828	0.2622
36	0.0067	0.0303	0.0217	0.0259	0.0180	0.1677	0.2277	0.3166	0.2571
37	0.0315	0.0305	0.0219	0.0103	0.0073	0.2551	0.1694	0.2165	0.3106
38	0.0012	0.0164	0.0101	0.0196	0.0359	0.1699	0.2208	0.2619	0.2565
39	0.0208	0.0117	0.0033	0.0057	0.0514	0.2000	0.2209	0.3302	0.2735
40	0.0173	0.0287	0.0123	0.0111	0.0400	0.1933	0.2408	0.2970	0.3391
41	0.0104	0.0071	0.0362	0.0291	0.0130	0.2241	0.2521	0.2446	0.2417
42	0.0110	0.0023	0.0169	0.0010	0.0411	0.1730	0.2537	0.2429	0.2317
43	0.0329	0.0235	0.0104	0.0036	0.0559	0.1901	0.2562	0.2202	0.2297
44	0.0022	0.0304	0.0139	0.0059	0.0072	0.2028	0.2502	0.2595	0.2630
45	0.0009	0.0088	0.0106	0.0287	0.0032	0.1556	0.2539	0.2779	0.2390
46	0.0176	0.0148	0.0012	0.0546	0.0297	0.2034	0.2976	0.2537	0.2385
47	0.0136	0.0071	0.0238	0.0273	0.0092	0.1733	0.2850	0.2335	0.3095
48	0.0029	0.0033	0.0251	0.0215	0.0079	0.1884	0.2275	0.2501	0.2365
49	0.0119	0.0154	0.0352	0.0129	0.0094	0.2219	0.2538	0.2708	0.2734
50	0.0193	0.0289	0.0290	0.0136	0.0368	0.1946	0.2479	0.2949	0.2626

Table 24 (Continued)

Table 25 lists the cumulative number of distinct seen positions obtained at the end of each iteration of the AlphaZero algorithm with different temperature τ's. The tested values of temperature τ included 0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, and 16.

	0.0625	**0.125**	**0.25**	**0.5**	**1**	**2**	**4**	**8**	**16**
0	0	0	0	0	0	0	0	0	0
1	46,367	48,161	48,025	50,185	51,133	52,893	52,673	53,939	53,443
2	68,395	70,793	72,221	74,093	74,665	75,633	76,497	76,845	77,299
3	78,447	81,713	86,843	88,429	89,717	91,069	92,213	91,261	92,227
4	84,095	89,149	95,047	97,621	100,905	102,445	102,745	102,145	102,749
5	87,231	92,857	98,347	103,429	108,389	110,951	111,429	109,657	110,813
6	90,179	95,069	100,187	106,509	114,181	117,659	118,417	116,113	116,877
7	92,223	96,845	101,287	108,621	118,133	122,971	124,017	121,721	121,873
8	93,419	98,245	102,107	110,089	121,063	127,723	128,893	126,241	125,909
9	94,079	99,229	102,667	111,329	123,479	131,611	132,449	130,153	129,413
10	94,783	99,773	103,475	112,293	125,503	134,727	135,917	133,445	132,573
11	95,335	100,037	104,295	113,089	127,295	137,091	138,561	136,249	135,433
12	95,867	100,321	104,955	113,589	128,847	139,495	140,949	138,517	137,977
13	96,715	100,621	105,675	114,105	130,179	141,679	143,193	140,773	139,997
14	97,543	100,949	106,091	114,561	131,451	143,631	145,033	142,677	141,837
15	97,935	101,085	106,531	114,805	132,391	145,459	146,717	144,361	143,565
16	98,195	101,229	106,931	115,093	133,531	147,127	148,501	146,029	144,953
17	98,559	101,421	107,195	115,341	134,407	148,567	149,993	147,785	146,473
18	98,631	101,665	107,307	115,565	135,311	149,911	151,453	149,085	147,841
19	98,711	101,769	107,495	115,757	136,003	151,047	152,809	150,641	149,177
20	98,735	101,801	107,759	115,985	136,831	152,425	153,941	151,813	150,433
21	98,759	101,821	107,815	116,121	137,515	153,425	154,953	153,073	151,437
22	98,791	101,853	107,839	116,305	137,943	154,561	156,069	154,165	152,449
23	98,831	101,881	107,911	116,469	138,599	155,309	156,989	155,045	153,321
24	98,863	101,889	107,975	116,573	139,231	156,113	158,001	155,945	154,033
25	98,991	101,909	108,047	116,717	139,723	157,001	158,981	156,825	154,881

Table 25: Detailed cumulative number of distinct seen positions for applying AlphaZero on 2×4 CDC with different temperature τ.

	0.0625	0.125	0.25	0.5	1	2	4	8	16
26	99,087	101,957	108,071	116,893	140,079	157,729	159,613	157,721	155,617
27	99,159	101,989	108,131	116,965	140,531	158,633	160,261	158,429	156,237
28	99,203	102,005	108,187	117,109	141,059	159,249	161,013	159,245	156,917
29	99,243	102,029	108,227	117,221	141,403	159,797	161,609	159,869	157,525
30	99,291	102,053	108,251	117,337	141,763	160,433	162,073	160,517	158,069
31	99,291	102,069	108,259	117,465	142,139	160,933	162,649	161,141	158,669
32	99,315	102,117	108,299	117,569	142,667	161,653	163,181	161,825	159,077
33	99,347	102,141	108,339	117,765	143,055	162,193	163,721	162,349	159,785
34	99,371	102,157	108,363	117,873	143,405	162,673	164,389	162,737	160,329
35	99,379	102,173	108,371	118,017	143,745	163,169	165,133	163,297	160,777
36	99,379	102,197	108,387	118,097	143,973	163,645	165,617	163,793	161,321
37	99,387	102,213	108,419	118,241	144,337	164,001	166,133	164,245	161,729
38	99,403	102,237	108,435	118,305	144,689	164,513	166,553	164,625	162,113
39	99,411	102,245	108,495	118,385	145,089	165,013	166,837	165,053	162,589
40	99,483	102,253	108,511	118,497	145,453	165,249	167,313	165,521	163,017
41	99,507	102,285	108,519	118,569	145,713	165,569	167,693	166,021	163,449
42	99,531	102,309	108,535	118,705	146,025	166,065	168,161	166,365	163,801
43	99,555	102,325	108,559	118,801	146,313	166,417	168,549	166,733	164,137
44	99,579	102,333	108,631	118,885	146,685	166,725	168,881	167,081	164,413
45	99,595	102,333	108,655	118,937	147,001	167,045	169,257	167,481	164,989
46	99,619	102,349	108,703	118,961	147,369	167,337	169,469	167,773	165,285
47	99,619	102,349	108,711	119,001	147,577	167,561	169,725	168,089	165,597
48	99,651	102,365	108,751	119,041	147,913	167,901	170,005	168,353	166,101
49	99,659	102,373	108,791	119,097	148,121	168,209	170,357	168,721	166,381
50	99,675	102,381	108,807	119,201	148,369	168,457	170,677	169,057	166,693

Table 25 (Continued)

Curriculum Vitae

Chu-Hsuan Hsueh was born in Taipei, Taiwan in 1992. After graduating from Taipei First Girls High School in 2010, she started to study Bachler program in Computer Science at National Chiao Tung University. She received the B.Sc. degree in 2014 and started to study Master program in the same place. In the next year, July 2015, she applied for direct pursuit of the Ph.D. degree. Her research interests include computer games, artificial intelligence, and machine learning. The Ph.D. research resulted in several publications and this thesis.