國立交通大學

資訊工程學系

碩士論文

象棋棋形辨識之研究

Pattern Recognition in Chinese Chess

研 究 生：許俊彬

指導教授：吳毅成　教授

中華民國九十五年六月

象棋棋形辨識之研究

Pattern Recognition in Chinese Chess

研 究 生：許俊彬　　Student：Chun-Bin Hsu

指導教授：吳毅成　　Advisor：I-Chen Wu

國 立 交 通 大 學
資 訊 工 程 學 系
碩 士 論 文

A Thesis
Submitted to Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science and Information Engineering

June 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年六月

象棋棋形辨識之研究

研究生：許俊彬　　指導教授：吳毅成

國立交通大學 資訊工程學系

摘要

象棋是一個相當受歡迎的棋類遊戲，在世界各地有廣大的玩家，象棋 AI 程式更是近代一個相當熱門的領域，許多人投入這方面的研究，不論在學術或商業界，都有很多傑出的作品。如果能針對象棋 AI 程式的設計做探討，使 AI 能夠提高棋力，對於棋與資訊界都將是很大幫助。

本篇論文研究象棋類 AI，把用在西洋棋、將棋與圍棋上的棋形辨識技術，做些探討與改善應用在象棋上面，並提出一套有系統的棋形建置方式，產生出能快速比對棋形的程式碼，最後評估並測量出象棋棋形辨識之效能及其實際應用於象棋 AI 程式的效果。

Pattern Recognition in Chinese Chess.

Student: Chun-Bin Hsu Advisor: I-Chen Wu

Department of Computer Science and Information Engineering

National Chiao Tung University

ABSTRACT

Chinese Chess is a very popular game played by many people in the world. And Chinese Chess AI program is researched widely nowadays.

This thesis studies AI Programming for Chinese Chess-like Games and improves the original technology used in Chess, Shogi and Go in order to apply it on Chinese Chess. It also proposes a new systematical process to build patterns. We evaluate and measure the efficiency of Chinese Chess pattern recognition and the actual effect when using it in the Chinese Chess AI program.

誌謝

首先要感謝我的指導教授，吳毅成博士，由於他不厭其煩的細心指導，這篇論文才得以順利完成。

此外特別要感謝黃德彥學長和翁仕全學長、王志祥學長，他們在研究的過程中給予我許多寶貴的意見和指導，還有家齊學姊之前的研究給了我很大的幫助，以及同實驗室裡一起奮鬥的夥伴育嘉、怡良、承翰的協助。還有很多其他我忘了提到的同學及朋友，在我的研究期間，給了我的關懷與鼓勵，陪我度過這段最值得回憶的學生生活。

最後，我要感謝我的父母、弟弟，在我的求學生涯中給了我最大的支持和照顧。還有，感謝我自己，你終於辦到了。謹以此論文，獻給我最摯愛的家人。

目錄

圖表目錄

第一章　緒論

本章第 1.1 節會介紹各種關於電腦象棋程式的背景知識，第 1.2 節介紹本篇論文的研究動機，最後在 1.3 節中簡單介紹全篇的大綱。

1.1 背景說明

本節中的第 1.1.1 小節會簡單介紹一下象棋，第 1.1.2 小節簡介紹電腦棋類程式的相關背景知道，最後在 1.1.3 小節會詳細的介紹電腦象棋程式的歷史。

1.1.1 象棋

象棋(Xiang Qi)，是一個在二維棋盤上雙方輪流走子的遊戲，先吃掉對方將(帥)者獲勝。它起源於中國古代，亦稱為 Chinese Chess。南宋的棋壇可說是中國象棋的第一個高峰期，從民間到宮中廷，處處風靡。在中國的文化裡，倍受人們的青睞。

象棋是一個規則簡單、複雜度極高，相當受歡迎的遊戲。隨著時代的變遷，象棋因為電腦科技的進步，新的演算法不斷開發，電腦象棋程式實力逐漸凌駕人腦之上，迄今亦有像西洋棋、圍棋一樣有國際性職業棋會與比賽。

1.1.2 電腦棋類程式

電腦下棋的主要思考方式是利用搜尋樹，展開所有可能著手，並從底部所取得的審局分數，利用 MiniMax 概念 由底向上，讓根部取得最佳棋步[26]。

棋類人工智慧一向都是電腦科技發展的重要指標，電腦對局(Computer Game)在人工智慧領域裡, 是一門既有趣又引人注目的學問，五子棋、圍棋、西洋棋，各式各樣的棋類人工智慧發展時有所聞，至今已有不少專家設計出相當優秀的程式。最引人關注的是 IBM 團隊研製的西洋棋程式 Deep Blue，當時打敗西洋棋棋王，從此棋類人工智慧如雨後春筍般澎湃發展起來。

這裡簡單介紹一下一些目前有定期電腦 AI 棋賽的棋類發展近況，以下會介紹西洋棋、象棋、將棋、圍棋。上述四種棋類遊戲的盤面複雜度從低到高為西洋棋、象棋、將棋、圍棋[7]，盤面愈複雜電腦 AI 的棋力愈難提升，因此他們的電腦 AI 程度也是依照這個順序由強到弱排下。目前西洋棋 AI 的棋力已經超越人腦，並努力追求更高的自我突破。象棋 AI 目前的棋力已達八段左右，足以抗衡特級大師，假以時日，必能超越人類。將棋因為可將吃掉的棋子放回盤面上，所以盤面複雜度又較西洋棋及象棋高一些，目前將棋 AI 棋力約有五段，算是高手級的實力。最後圍棋因為其盤面複雜度實在太高，棋盤大小有 361 格，其電腦 AI 實力始終沒有顯著的提升，目前圍棋 AI 棋力僅有五級，只能算是新手的程度。

西洋棋: 棋力已超越人腦
象棋: 棋力達八段, 已可抗衡特級大師
將棋: 棋力達五段, 約爲高手程度
圍棋: 棋力達五級, 約爲新手程度

圖 1-1　各種棋類遊戲棋力現況

1.1.3 電腦象棋程式

電腦象棋程式的研究起步於1980年代初期[22]。到了1985年，第一個中國象棋知識庫誕生，黃東輝以樹狀資料結構的基礎，框架基底表示方式，使用雜湊函數(Hash function)的技巧，開發出一個利於維護及取用的電腦象棋開局庫[28]。到了1987年宏碁舉辦了電腦象棋大賽，1989年電腦奧林匹亞加設象棋比賽，引起電腦象棋的研究風氣[25]。80年代末及90年代初的這段時期裡，比較重要的電腦象棋程式有虞希舜的”特級大師”及”將族”，曹國明的”象棋專家”[23]、鄭武堯的“象棋明星(ELP)”，其中虞希舜的”特級大師”雖然曾經獲得第一屆電腦奧林匹亞的冠軍，當時已有至少四段的棋力[24]，但因其為商業軟體，並未發表過任何關於象棋程式設計及製作的論文[25]；曹與鄭都是屬於台灣大學許舜欽教授所帶領的電腦象棋研究團隊，國內有關電腦象棋的論文多出於此，1991年，許舜欽教授更總結了自1944年以來，Von Neumann和Morgenstern提出的MiniMax搜尋樹方法到當時發展的所有相關棋類遊戲演算法，包括了“前向修剪”、“$\alpha-\beta$切割”、“目標導向搜尋”、“殺手經驗法則”、“剃刀切割”、“SSS*搜索”等十五種演算法進行全面的剖析[26]。其後不斷有從各種角度研究電腦象棋程式的論文發表，如平行處理、象棋殘局系統[19][30][33]、開局知識庫系統[31]。一直到1998年，當時電腦象棋中的佼佼者ELP正式取得六段証書[27]。之後更多的理論與實作學術論文出現，如新的加速搜尋方式[35]、高效率停著殺[34]、棋規的判斷[20]、結合開局中局與殘局資料庫[32]、以回溯(Retrograde)方式建構殘局資料庫[2][3][10][11][12][13]等，許多象棋商業軟體也在此時出現，電腦象棋程式正快速的發展。2005年第十屆電腦奧林匹亞象棋程式的冠軍“象棋奇兵(xqmaster)”，棋力已達八段左右，足以抗衡特級大師，假以時日，必能超越人類。

1.2 研究動機

許舜欽教授曾指出，“審局函數偏差”為電腦象棋中的一個障礙[29]。除了子力及位置分數，即使再加上自由度及保護度等屬性、審局仍然難以完善。基本審局無法辨識出如空頭炮及雙車銼(圖1-2)等大幅優勢的盤面，當空頭炮棋形成立時，任何黑方棋子皆不能移動到炮與將之間，黑方棋子行動範圍嚴重受限，並且假如紅方任一棋子移動到炮與將之間，還能做抽子，使黑方受到相當大的威脅。雙碑銼的棋形成立時，因為黑將在邊上，無法尋得其它棋子的保護，並且兩雙紅碑可以先一隻佔據九宮格的中路，使黑將只能逃到上路或下路，另一隻紅碑再催殺，黑將即無生路。因此為了能辨識出此類優勢盤面，仍需考慮棋子各種相對位置的擺設，即”棋形”。

圖 1-2(a) 空頭炮

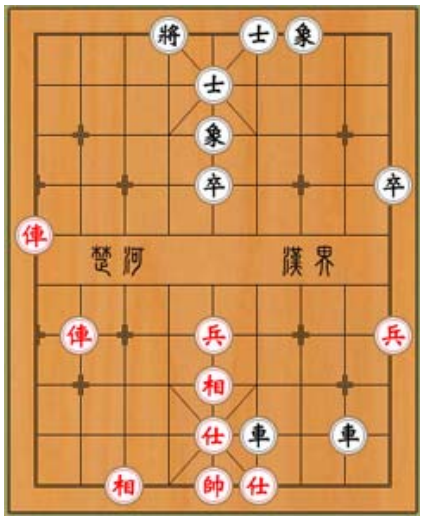


圖 1-2(b) 雙車銼

然而加強審局後的象棋程式，其搜尋深度會因審局所花費的時間而減少[7]，必需盡量減少棋形比對的時間成本。

因此，本篇論文研究象棋類 AI，把用在西洋棋、將棋與圍棋上的棋形辨識技術，做些探討與改善應用在象棋上面，並提出一套有系統的棋形建置方式，產生出能快速比對棋形的程式碼，最後評估並測量出象棋棋形辨識之效能及其實際應用於象棋 AI 程式的效果。

1.3 論文大綱

在本論文第二章的背景說明裡，介紹一些前人各種棋類遊戲棋形辨識的研究情況。第三章介紹我們所提出的象棋棋形辨識方法與流程。第四章介紹象棋棋形辨識方法中所使用的編譯 MFDA 程式碼演算法。第五章評估並測量各種編譯出的 MDFA 程式碼，及實際應用編譯出的程式碼於本實驗室開發之象棋 AI 程式－Chimo 上，觀察其實用價值。最後第六章是結論與未來展望。

第二章　背景說明

本章介紹各種棋形辨識之目的，最後介紹各種棋類 AI 棋形辨識的發展背景，各別介紹其棋形辨識技術。目前有使用到棋形辨識的棋類遊戲有西洋棋，將棋及圍棋。以下第 2.1 節簡單介紹各種棋類 AI 發展現況，第 2.2 節介紹西洋棋使用過的棋形辨識技術。第 2.3 節介紹將棋使用過的棋形辨識技術。第 2.4 節介紹圍棋使用過的棋形辨識技術。

2.1 棋形辨識之目的

雖然本篇主要應用棋形辨識於審局函式之改善，實際上棋形辨識之目的繁多，常見的有以下五種：

1.改進審局函式(Evaluation Function)

棋類 AI 之審局函式，僅靠子力、自由度及保護度等基本屬性、審局難以完善，基本審局無法辨識出如空頭炮及雙車銼(圖 1-2)等大幅優勢的盤面，因此為了能辨識出此類優勢盤面，仍需考慮棋子各種相對位置的擺設。

2.行棋計畫(Plan)

棋類 AI 與人類思考的差別是電腦不知道究竟哪一些棋步比較重要，而必需做廣泛的全域搜尋，才能找出最佳的棋步。而行棋計畫的概念就是要電腦 AI 做出如人類計畫般的思考模式，在判斷出某一特定盤面後，設定單一目標，僅做此目標的搜尋，可以節省大量不必要浪費的時間，加快搜尋速度[15][16]。像是圖 2-1 的盤面中，紅方可思考如何將紅碑移至一到三路，再往底線催殺，或是考慮如何捉死底線的黑象。

圖 2-1 行棋計畫例子

3.產生選擇性棋步(Candidate Generation)

將棋因為吃掉的棋子可以重新放回盤面上，它的盤面複雜度更高於西洋棋及象棋，因此在實在將棋 AI 時，無法完整的搜尋所有的可能著手，必需從中挑選比較重要的棋步來搜尋[6]。

4.著手順序調整(Move Ordering)

著手順序在棋類 AI 中是很重要的一環，因為 Alpha-beta Pruning 的概念，愈早搜尋好的著手，所需要的搜尋時間愈短，棋形辨識也可以用來改善著手順序。

5.選擇性加深搜尋(Selective Deepening)

選擇性加深搜尋深度，亦為棋類 AI 中重要的一個概念，比較重要的棋步可以搜尋深一點，比較不重要的棋步就淺一點，這種搜尋方式，可以讓 AI 的思考更快更精準，不會浪費時間在無關緊要的棋步，棋形辨識可以在特定的盤面，告訴 AI 哪些棋步比較重要，需要加深搜尋[5]。

2.2 西洋棋使用過的棋形辨識技術

西洋棋 AI 是最早開始使用棋形辨識技術的棋類 AI，西洋棋 AI 使用過的棋形辨識技術有以下三種：棋子在棋盤上的絕對位置[8]，棋子之間的攻擊與保護關係[4][8]，雙方剩餘棋子的種類及數量[8]。

棋形辨識最基本會考慮到某些棋子在盤面上的絕對位置，當各棋子的絕對位置排列成一特定情形時，某棋形即成立。例如圖 2-2 即代表一單碑牽制車馬棋形，若紅碑的絕對位置在座標(2,8)，黑馬的絕對位置在座標(1,8)，黑車的絕對位置在座標(1,9)，即符合本棋形，可對紅方做適當加分。

圖 2-2　單碑困住車馬的棋形

棋子之間還有攻擊與保護的關係，辨別出哪些棋子可以攻擊到哪些棋子，哪些棋子正保護哪些棋子，有助於我們對盤面做進一步的分析。圖 2-3 即代表一棋子之間互相攻擊與保護的棋形。圖中紅碑可以攻擊黑馬，黑馬可以攻擊紅炮，黑車可以攻擊紅炮，雙紅炮互相保護，紅仕保護紅炮。將此關係做近一步分析：雖然紅碑可以攻擊黑馬，但黑馬有黑車保護，故紅碑實際上不能吃黑馬；黑馬及黑車雖然兩子皆可攻擊紅炮，但紅炮亦受另一紅炮及紅仕兩子之保護，實際上，黑方也無法拿下紅炮。

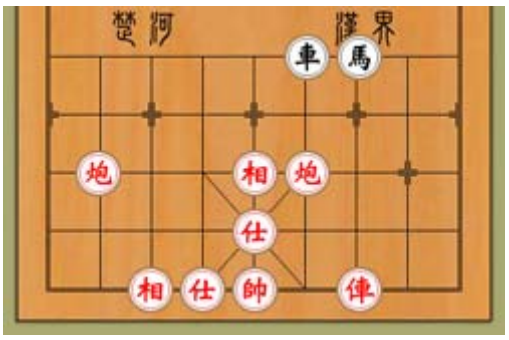


圖 2-3 棋子間互相攻擊與保護關係圖

盤面上剩餘棋子的種類及數量，也是一種重要的棋形，各式各樣的棋類遊戲舉辦著成千上萬的比賽，如果每一場棋賽都必需走到最後一步，拼盡最後一子，才能分出勝負，會浪費許多的時間，經驗是可以累積的，前人的經驗告訴我們，什麼樣的盤面，最後勝負必定是什麼樣的結果，棋局可以就此打住，像常聽到的單砗難勝士象全，三高兵勝士象全，皆為此類棋形的範疇。這類的訊息不僅可以做為比賽中止的參考，也可以拿來增強棋類 AI。例如我們知道三高兵勝士象全，因此 AI 只要能想盡辦法取得己方三高兵及對方僅剩士象全的盤面，也就等於取得勝利了。下表略舉了一些盤面剩餘棋子種類及數量與勝負之間的關係。

1.孤兵擒王
2.高低兵例勝雙士
3.高低兵例勝雙象
4.雙高兵例勝單炮
5.三兵例勝卒單缺士
6.三兵例勝士象全
7.三兵例勝炮雙象
8.三兵例勝馬雙象
9.高兵例和單士
10.高兵和炮卒

表 2-4　數種盤面剩餘棋子種類及數量與勝負之關係

2.3 將棋使用過的棋形辨識技術

將棋使用過的棋形辨識技術有以下二種：棋子之間的相對位置關係與錨(Anchor)的使用[5][6]。

棋子間相對位置關係，被廣範應用在王(King)活動範圍不受限，及棋子能長距離移動的棋類遊戲中。因為以優先殺死對方的王為勝方的棋類遊戲，棋形多以王為中心，若王的活動範圍不受限，隨著王各種不同的位置，用絕對位置方式所產生出來的棋形將會過多，圖 2-5 表示三種不同的絕對位置棋形，若不考慮將受九宮格的活動限制，三種棋形中將與傌的相對位置是相同的，可用一相對位置棋形所取代，所以在王可以任意自由活動的情況下，相對位置棋形有其必要性。

圖 2-5 三種相對位置相同的棋形

棋子能長距離移動的棋類遊戲中，在意的是棋子能否在一次移動中直接抵達某棋子的位置，即兩棋子間是否沒有任何棋子，所以是此二子間的相對位置關係。圖 2-6 為一空頭炮棋形，棋形重點即為炮和將之間是否沒有任何棋子。

圖 2-6 空頭炮棋形

錨(Anchor)的概念為，比對某棋形前，先檢查此棋形的錨(某一棋子)是否存在盤面上，如果存在再做完整棋形比對，如果不存在，則此棋形不可能成立，即不用再做完整棋形比對。例如空頭炮的棋形可以以炮做錨。

2.4 圍棋使用過的棋形辨識技術

圍棋使用過的棋形辨識技術有以下四種：類神經網路(Neural Network)棋形辨識[6]，DFA (Deterministic Finite-state Automaton) [9][14]，Patricia Tree[9]。

文亞成採用類神經網路原理設計棋形辨識處理系統，將類神經網路的方法做修正應用在圍棋棋形辨識上。

GNU Go 使用 DFA 的概念來完成圍棋棋形辨識，此方法能快速的比對多重棋形。圖 2-7 為 GNU GO 棋形辨識的路徑，從中心黑子出發，一路以逆時針方向旋轉出去，做棋形辨識。

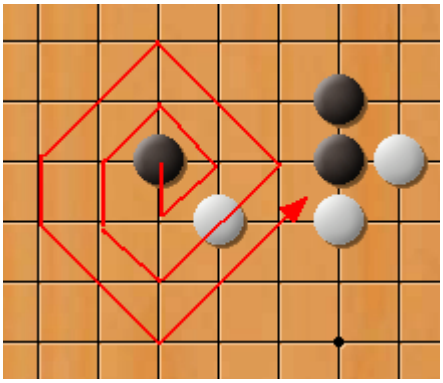


圖 2-7 圍棋棋形 DFA 辨識路徑

圖 2-8 為依照圖 2-7 路徑辨識出棋形“??.X”的 DFA 圖形(“?”代表任意棋子，”．”代表無棋子，”X”代表黑棋，”O”代表白棋。)

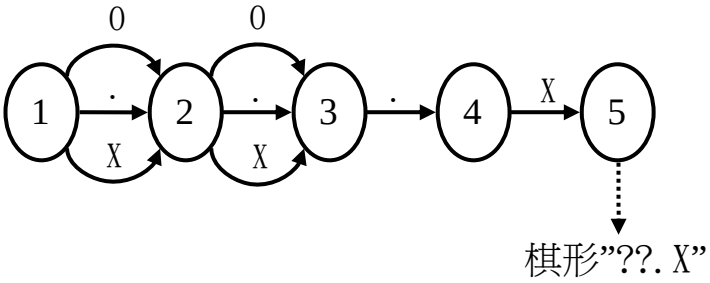


圖 2-8　圍棋棋形“??.X”之 DFA 圖形

圖 2-9 為圖 2-8 圍棋 DFA 棋形“??.X”再加入棋形”XO”後所形成的 DFA 圖形。

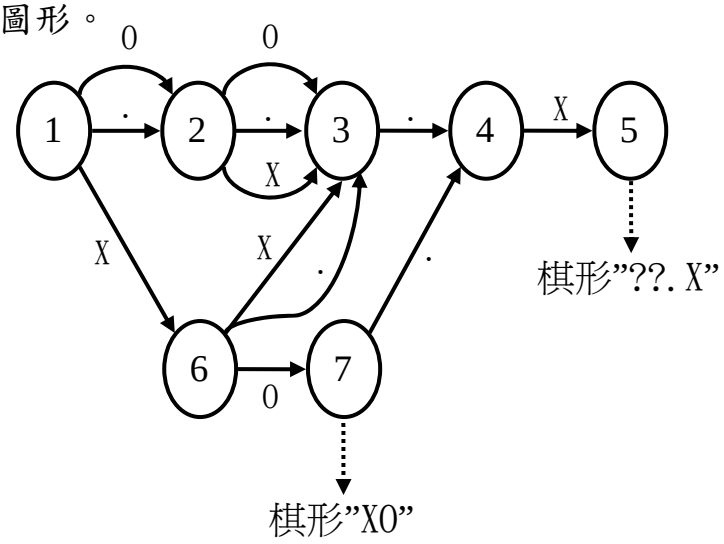


圖 2-9　圍棋棋形“??.X”加入 XO”後之 DFA 圖形

李佑堂將圍棋棋形比對樹化簡為一 Patricia Tree，能更快速篩選出所有可能符合的棋形，再將可能符合的棋形用 bitmap 的方式做完整比對，看這些棋形是否真的符合。

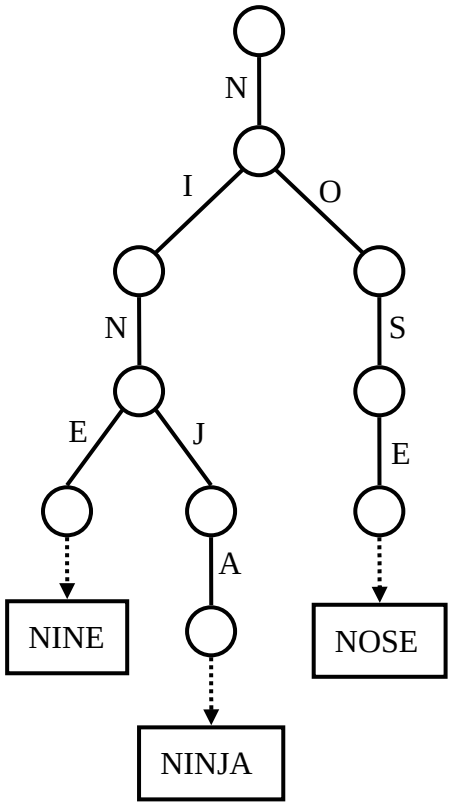


圖 2-10　化簡前的比對樹

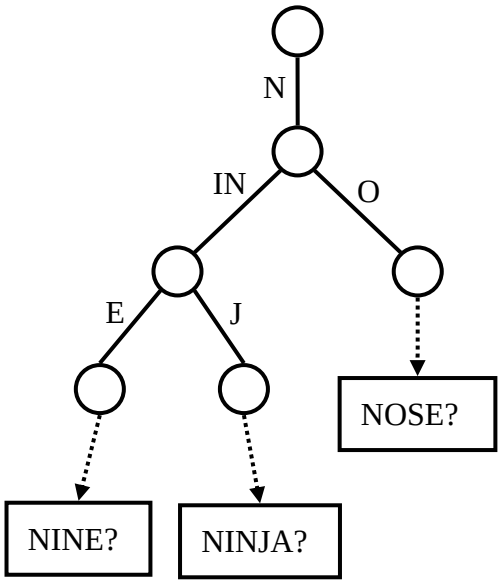


圖 2-11　化簡後的比對 Patricia Tree

圖 2-10 是以辨識文字 NINE、NINJA、NOSE 為例所成之比對樹，圖 2-11 為化簡比對樹後所得之比對 Patricia Tree。

第三章 本篇的象棋棋形辨識方法

在這一章中，主要介紹改善現有的棋形辨識技術，推廣原有的各種棋形辨識方法，應用在象棋上面。本篇所使用的棋形辨識方法有下列三者:基本採用 MDFA(Modified Deterministic Finite-state Automaton)棋形辨識技術(MDFA 為一類似 DFA 之架構，以符合本篇棋形比對系統之需求)，以絕對位置比對為主，相對位置比對為輔，完整的建置及使用棋形流程。首先在第 3.1 節中介紹在象棋上應用 MDFA 棋形辨識技術。接下來在第 3.2 節講解象棋的絕對位置及相對位置比對。第 3.3 節介紹完整的建置及使用棋形流程。

3.1 在象棋上應用 MDFA 棋形辨識技術

一棋形由數個子棋形構成，子棋形為某一棋子在盤面上的位置，若此棋子位於指定位置，則稱此子棋形成立，一棋形成立與否的充要條件為他的所有子棋形皆成立。

所有子棋形構成節點點產生出的 MDFA，即用來判斷所有棋形中，哪些成立，哪些不成立。圖 3-1 為一象棋 MDFA 例子，左右兩棋盤分別代表兩種棋形，兩種棋形比對可合成一個 MDFA。

子棋形分為絕對位置及相對位置兩種，接下來會在 3.1.1 介紹棋子絕對位置比對，在 3.1.2 中介紹棋子相對位置比對。

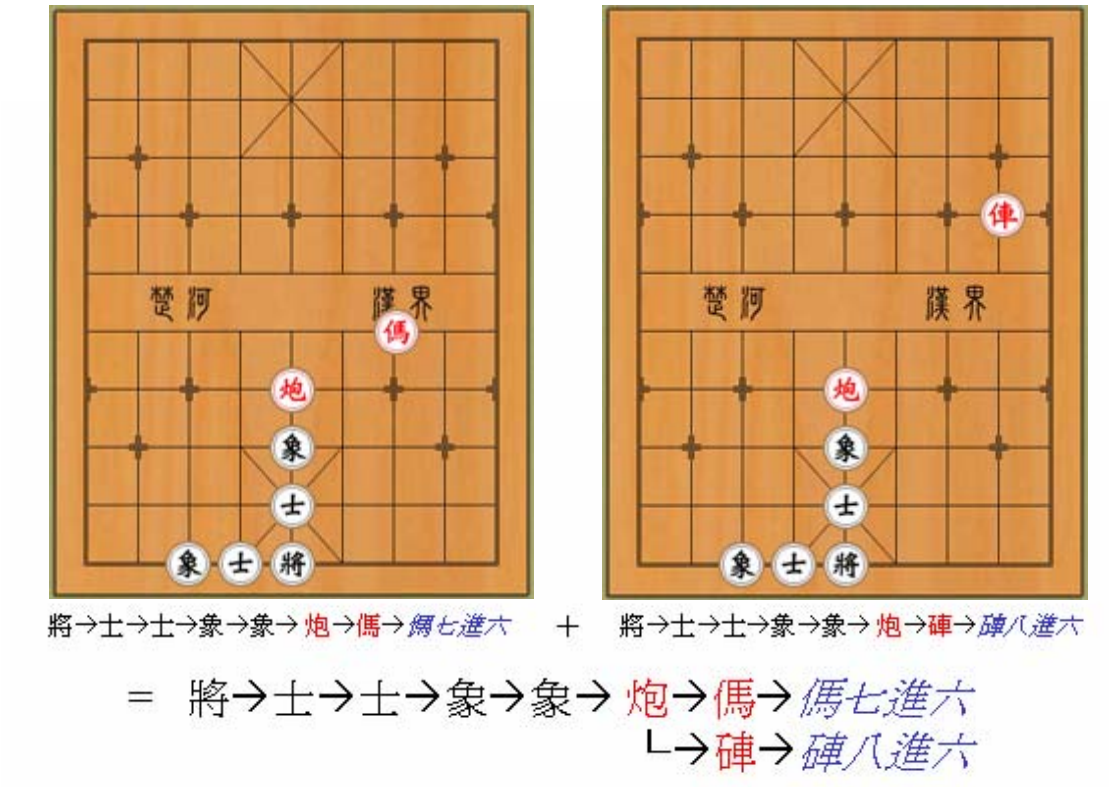


圖 3-1 象棋 MDFA 實例

3.1.1 棋子絕對位置比對

本篇定義棋子絕對位置的方法為－某特定棋子是否有在一特定區域裡出現。其格式為(MatchLoc 區域 A 棋子 B)，此格式代表棋子 B 是否有在區域 A 中出現。

舉個例子，(MatchLoc (Column7 Column8 Column9) 碑)，此子棋形代表碑是否有在第 7 到第 9 行中出現，即為圖 3-2 所表之區域。

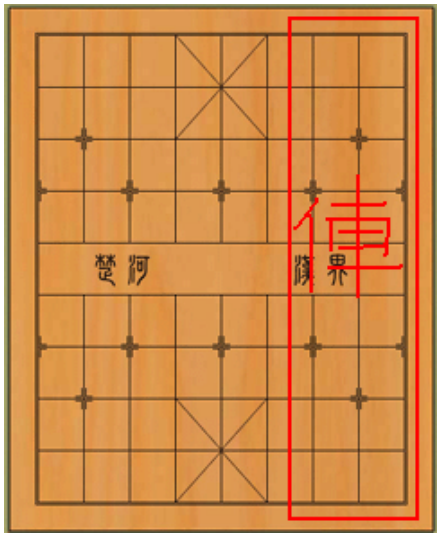


圖 3-2 棋子絕對位置

3.1.2 棋子相對位置比對

本篇定義棋子相對位置比對的方式為－從某一特定或不特定位置，向一特定方向看去，沿途是否依順序出現指定棋子。其格式為(MatchSeq 位置 A　方向 B　棋子 C　棋子 D ... 棋子 N)，此格式代表從位置 A 往方向 B 看去，必需先經過棋子 C，再來棋子 D，依此類推，最後棋子 N。

舉個例子，(MatchSeq LA5 UP 將 馬 * ANY * 炮)，此棋形代表從 LA5 往上方的序列是否為: 黑將－黑馬－任意數量空格－任意棋子－任意數量空格－紅炮。即為圖 3-3 所示圖形。

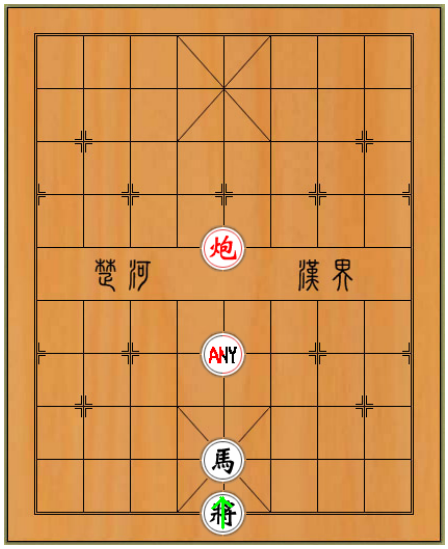


圖 3-3 棋子相對位置

3.2 完整的建置及使用棋形流程

本篇所提出的方法，為使用一圖形化介面棋形編輯器，來產生並且讀取代儲存棋形，儲存起來的棋形原始檔案，可以再經由編譯器轉換成一 MDFA，再依照此 MDFA 產生比對棋形所需的程式碼。流程如圖 3-4 所示。

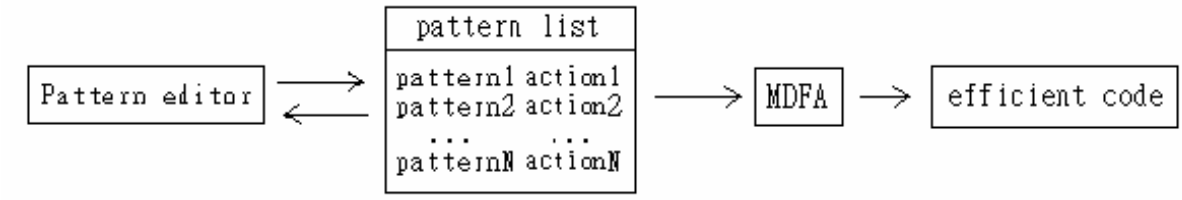


圖 3-4 完整的建置及使用棋形流程

本篇所使用之圖形化介面棋形編輯器可完整編輯所有想表示的棋形，具備的功能有－讀取棋形檔案、儲存棋形檔案、開啟新的棋形、複製棋形、刪除棋形、上下選擇棋形、編輯絕對位置子棋形、編輯相對位置子棋形、編輯棋形符合動作。下圖為此圖形化介面棋形編輯器截圖。

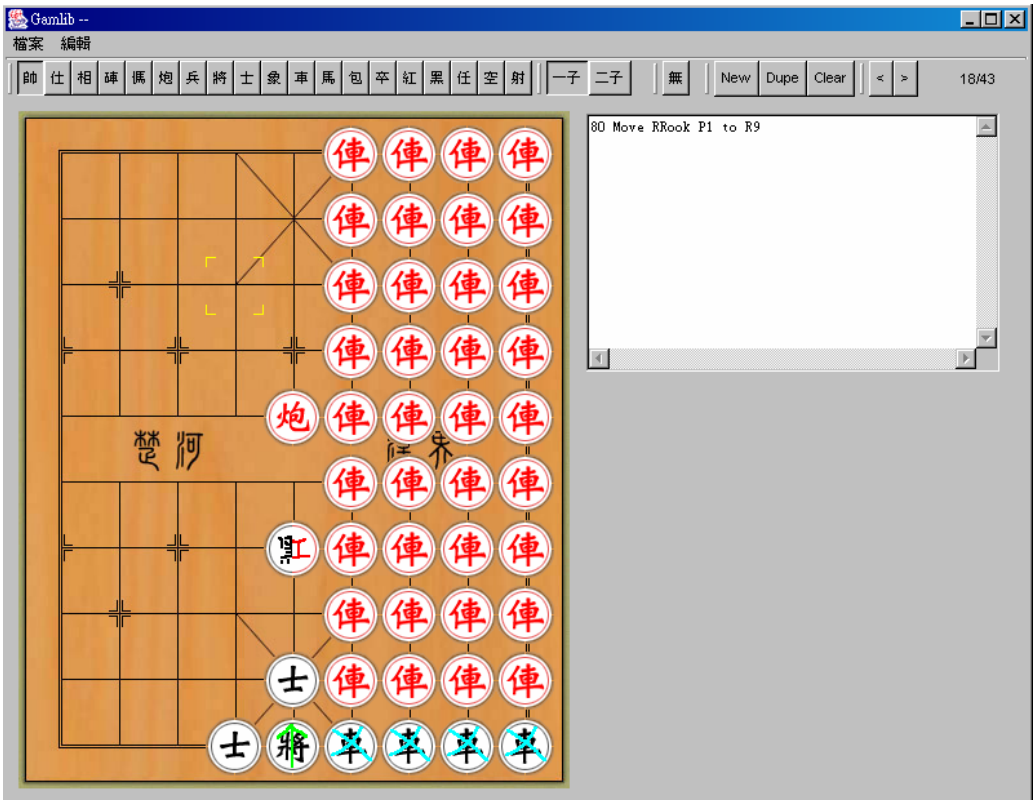


圖 3-5 圖形化介面棋形編輯器

第四章　編譯高效率 MDFA 程式碼

在這一章，我們介紹如何編譯出高效率的 MDFA 並轉成程式碼前，會先定義何謂高效率，再來介紹各種我們提出的編譯方法。

接下來分別在 4.1 節介紹何謂高效率，4.2 節介紹棋形辨識效率的定義，4.3 節當中介紹基本的編譯方法，4.4 節介紹 Switch 法，4.5 節介紹 IF 法。

4.1 何謂高效率

比對棋形程式碼所著重的要點有二，第一，時間成本，即平均執行時間要短，平均執行時間若太長，則會拖垮 AI 程式的執行效能，所以是棋形程式碼的重點之一。第二，空間成本，即程式碼行數要少，程式碼行數如過多，AI 程式在編譯及執行時效能都會降低，因此也要限制在一定的範圍之內。本篇所定義高效率為此二著皆著重所取得之平衡點。

4.2 棋形辨識效率的定義

以下先對各符號做說明，盤面集合B = { b_1 , b_2 , ... , b_n }，b_i為第i個盤面，1≦i≦n；棋形集合P = { p_1 , p_2 , ... , p_m }，p_j為第j個棋形，1≦j≦m；prob(b_i)為盤面b_i發生的機率，T(b_i, p_j)為在盤面b_i，比對棋形p_j所花費時間；$P_{match}(b_i)$為在盤面b_i，實際符合的棋形集合；A(p_j)為執行棋形p_j符合動作Action(j)的時間。則時間成本為：

$$T_{total} = \sum_{\forall b_i \in B} (prob(b_i) \times (\sum_{\forall p_j \in P} T(b_i, p_j) + \sum_{\forall p_j \in P_{match}(b_i)} A(p_j)))$$

本節定義時間成本為在所有可能盤面下，依照盤面出現的機率，及在此盤面下比對所有棋形，還有棋形符合時所執行相對動作所花時間，所得之平均執行時間。

4.3 基本編譯方法

最基本的編譯棋形方法為各別檢查每個棋形(如圖 4-1) 。此外有一改良方式：檢查每個棋形時，從發生機率最低的子棋形開始檢查，因為先檢查的子棋形如果不通過，此棋形即不成立，後面的子棋形也不用再檢查，所以花費的平均檢查時間會較少。

```
If(硨在右邊 && 傌在下面) Action(1);
If(硨在右邊 && 炮在中間 && 將在原位) Action(2);
If(硨在左邊 && 士在上面&& 傌在下面) Action(3);
If(傌在右邊 && 象在下面) Action(4);
            ......
If(炮在右邊 && 象在左邊) Action(n);
```

圖 4-1　基本編譯方法

4.4 Switch 法

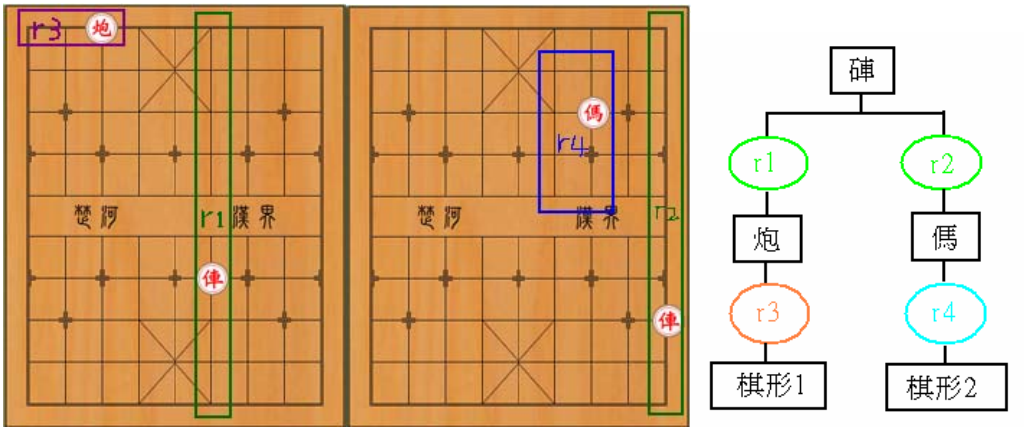
本篇所提之 Switch 編譯方法，為一尋找由 switch 組成的高效率程式碼之方法，其理論基礎為：使用一層 switch 能定位一顆棋子，象棋總共有 32 顆棋子，所以最多只要花費 32 層 switch 的時間就能辨識完所有的棋形。

以下會先在 4.4.1 小節中簡單介紹 switch 程式碼之架構，接下來4.4.2小節會介紹switch程式碼的平均時間成本公式，再來4.4.3小節介紹產生出平均時間成本最小 switch 程式碼之方法，本篇稱之為 Switch 法。最後在 4.4.4 小節中會介紹 Switch 法的問題及可能解決的方法。

4.4.1 switch 程式碼

switch 程式碼為一完全由 switch 所構成，能完全辨識所有棋形之程式碼，每使用一個 switch 能定位一個棋子的座標，並由此棋子的座標所可能存在各種不同位置，接著在適當的地方使用 switch 定位另一棋子，直到所有該定位的棋子皆為成為止。

圖 4-2 為一簡單 switch 程式碼的例子，此程式碼能辨識出棋形 A 和棋形 B 兩種棋形，棋形 A 代表硨在 r1 內，炮在 r3 內，所成之棋形；棋形 B 代表硨在 r2 內，傌在 r4 內，所成之棋形。程式碼架構圖先用一個 switch 定位車的位置，如果硨在位置在 r1，則棋形 A 尚成可能成立，棋形 B 已不可能成立，所以只需再比對棋形 A 中的另一子棋形，炮是否在 r3 中即可；如果硨的位置在 r2，棋形 B 不可能成立，棋形 B 尚可能成立，所以只需再比對棋形 B 中的另一子棋形，傌是否在 r4 中即可。



棋形 A　　棋形 B　　程式碼架構圖

圖 4-2 switch 程式碼簡單範例

4.4.2 switch 程式碼平均時間成本公式

switch 程式碼為一類似 Optimal Binary Search Tree 之樹狀結構程式碼，其程式碼平均時間成本公式如下()：

$$T(R,P)=\min_{\forall p_i \in P}\Big(\sum_{\forall R' \in S(p_i,R)}(prob(p_i,R')\times T(R',P-\{p_i\}))\Big)$$

其中，R為剩餘棋形集合，P為剩餘棋子集合, T(R, P)為剩餘棋形集合為R，且剩餘棋形集合P沒檢查完時的平均成本，prob(p_i, R’)為檢查棋子p_i時R’ 出現的機率，S(p_i, R)為以棋子p_i分類，R剩餘的各種棋形子集合。圖 4-3 中，程式碼平均時間成本公式即為檢查砩所花費成本加上砩位置在r1 的機率乘上砩在r1 時接下來需要花的時間成本，再加上砩位置在r2 的機率乘上砩在r2 時接下來需要花的時間成本。往下不斷遞迴時間成本公式，直至該檢查的棋子皆檢查完為止。

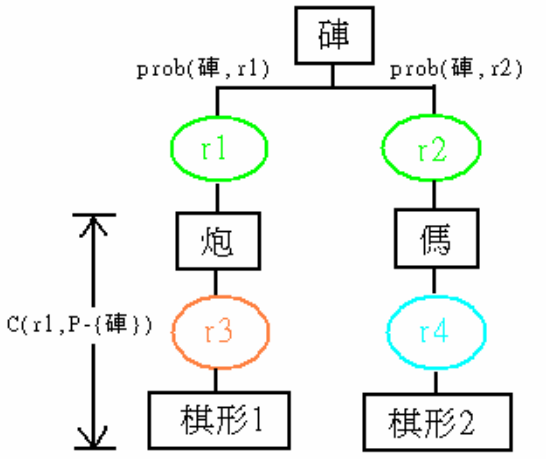


圖 4-3 switch 程式碼平均時間成本示意圖

4.4.3 Switch 法之演算法

為產生平均時間成本最短之 switch 程式碼，本篇提出一種 Switch 法，利用用搜尋樹，根據棋子在各個位置出現的機率表(圖 4-4), 在每個節點嘗試以每個棋子放置 switch，找出各個 switch 位置最適當的棋子，求出平均深度最小的樹。

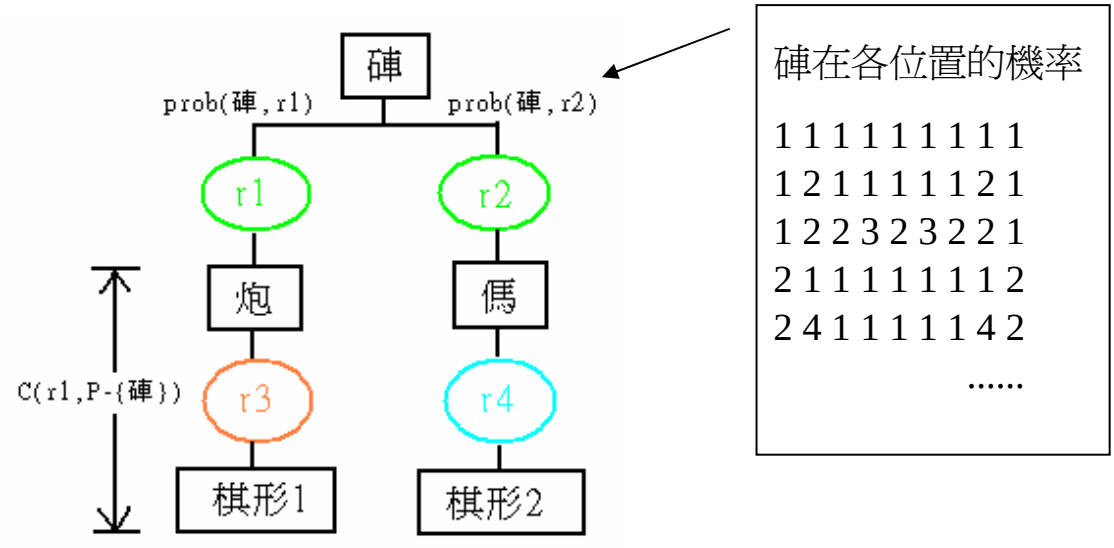


圖 4-4 依棋子在各位置的機率表計算時間成本

圖 4-5 為產生圖 4-2 最少平均時間成本 switch 程式碼之 Switch 法搜尋樹：

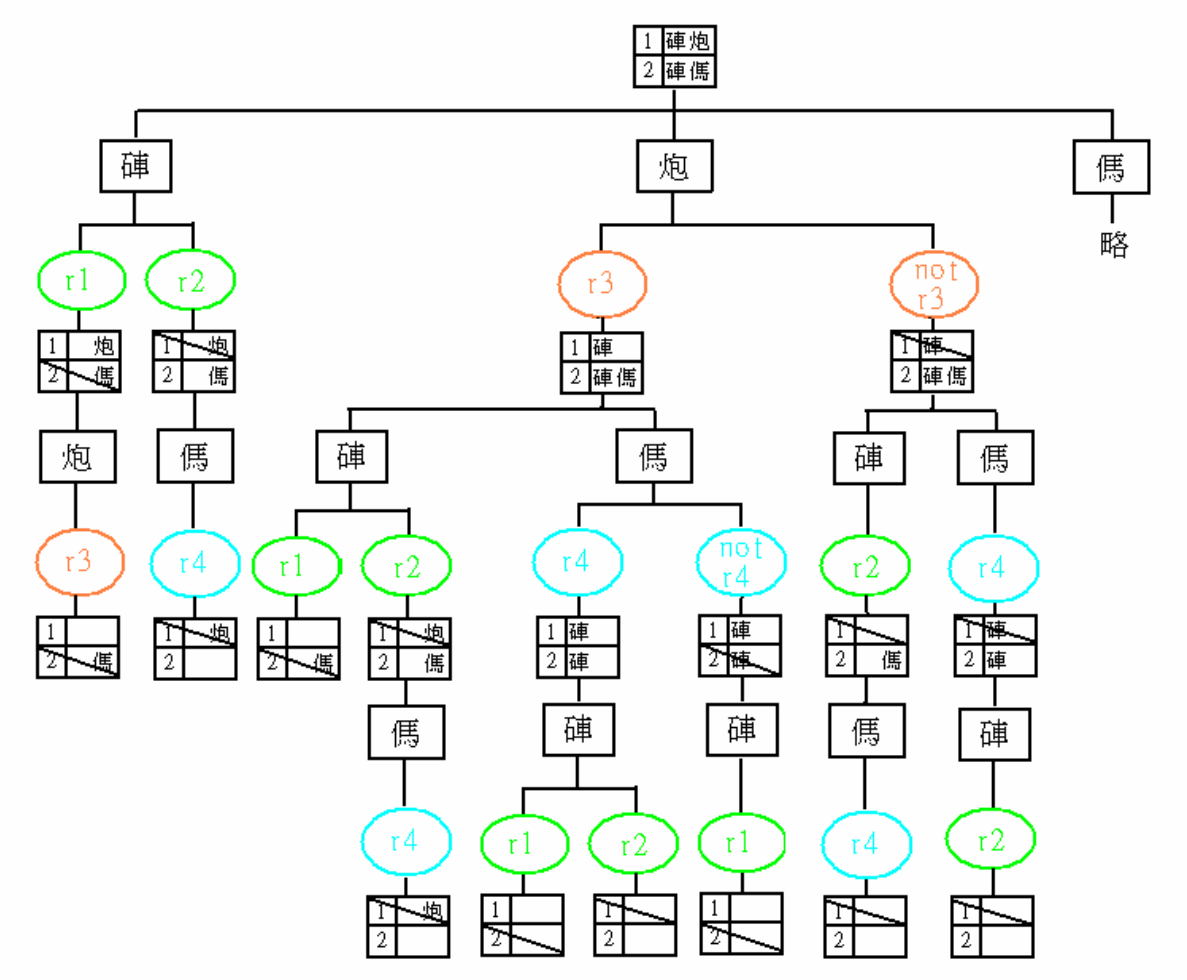


圖 4-5 Switch 法搜尋樹

4.4.4 switch 程式碼之問題與改良方式

隨著棋形愈加愈多，switch 程式碼中每檢查一個棋子所產生的分枝數量會愈來愈多，節點數量成指數成長，最後程式碼將過於龐大。在加入 50 種棋形下，switch 程式碼約要檢查 6 個棋子，而每檢查一個棋子平均產生約 10 個分支，節點數約為 10 的 6 次方 =

100,000 個，這麼龐大的程式碼是無法被接受的。

以下提出一種可以稍微解決 switch 程式碼過於龐大問題之方法。因為在 switch 程式碼中，某節點之平均時間成本公式為到達此節點之機率乘上此節點以下之成本，因為在到達機率很低的節點，即使此節點以下的成本上升，對整體成本的影響仍然不大。本方法為：到達機率過低的節點，直接用時間成本高但空間成本低的 if 來比對剩下的子棋形。使用此方法的結果為時間成本小幅上升，程式碼大幅縮小。

4.5 IF 法

本篇所提之 IF 編譯方法，為一利用貪婪演算法產生由 if 組成的高效率程式碼之方法，其理論基礎為：數行 if 程式碼中，如果各行程式碼之間有相同的比較運算，可以簡單的利用一額外的 if 提出相同的部份。依照此理論基礎，各棋形中相同子棋形之檢查, 可以一次提出做完。以下會先簡單介紹 if 程式碼之架構，接下來介紹產生出平均時間成本低的 if 程式碼之方法，本篇稱之為 IF 法。

4.5.1 if 程式碼

if 程式碼之基本架構為不斷做提出子棋形動作直至無法再做作何提出，所產生之程式碼。圖 4-6 為一提出之簡單例子：

```
if(硨在右邊  &&  傌在下面&&炮在中間) Action(1);
if(硨在右邊  &&  炮在中間) Action(2);
```

↓

```
if(硨在右邊)
{
      if(傌在下面&&炮在中間) Action(1);
      if(炮在中間) Action(2);
}
```

↓

```
if(硨在右邊)
{
      if(炮在中間)
      {
            if(傌在下面) Action(1);
            Action(2);
      }
}
```

圖 4-6 IF 法子棋形提出之基本方式

4.5.2 IF 法之演算法

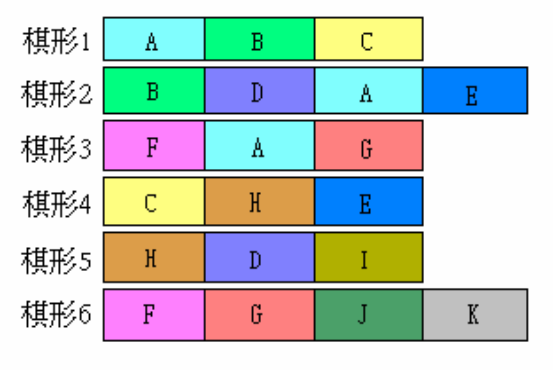
IF 法為一利用貪婪演算法產生由 if 組成的高效率程式碼之方法。首先會先提出為何無法使用搜尋樹找出最佳 if 程式碼，接下來會介紹本篇如何使用貪婪演算法來產生效能佳之 if 程式碼。

若要使用搜尋樹找出最佳 if 程式碼，則需要提出之次數與總棋形數量成正比，則不論搜尋的分枝為全部剩下可提出之子棋形，或是評估最佳的 2 或 3 個分枝，其時間複雜度最終必為 b 的 n 次方，其中 n 與棋形總數成正比。指數成長之執行時間是無法被接受的，所以目前認為無法有效使用搜尋樹找出最佳 if 程式碼。

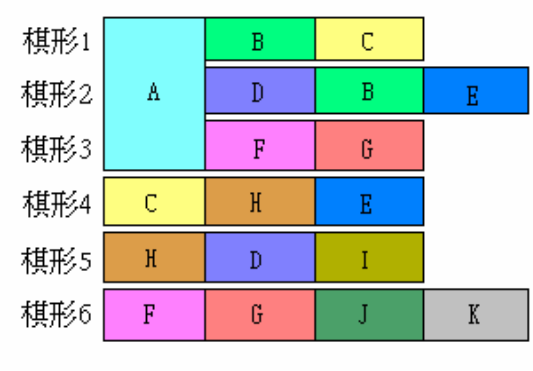
本篇使用之貪婪演算法為：檢查同一組之棋形中所有的子棋形，提出出現次數最多之子棋形，出現次數相同者，比較其達成機率，機率低者先提出。同一組之棋形，在提出出現之數最多之子棋形，分為被提出組與剩剩餘組，兩組再分別遞迴，找出各自出現次數最高之子棋形・・・。

圖 4-7 為 IF 法提出子棋形的例子。圖中 A, B , C, ... ,K 為子棋形，圖 1 為未做作何提出前的基本圖形；圖 2 首先找出棋形 1~6 中所有子棋形出現最多次的 A，將其調至最前方並提出；圖 3 找出提出 A 後的棋形 1~3 中出現最多的子棋形 B，並將其提出，圖 4 找出之前未提出任何子棋形之棋形 4~6 中出現最多的子棋形 H 並提出，最後圖四所成的圖形再無任何子棋形可提出。

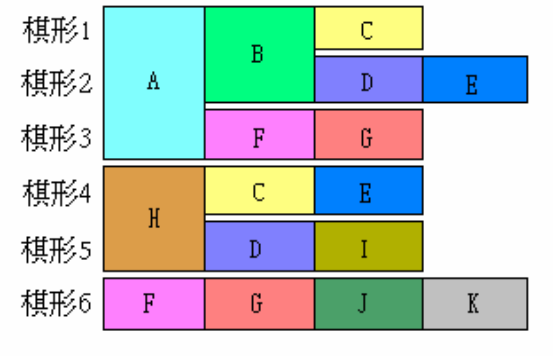
1



2



3



4

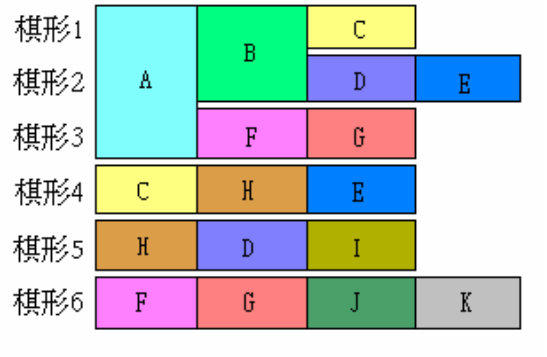


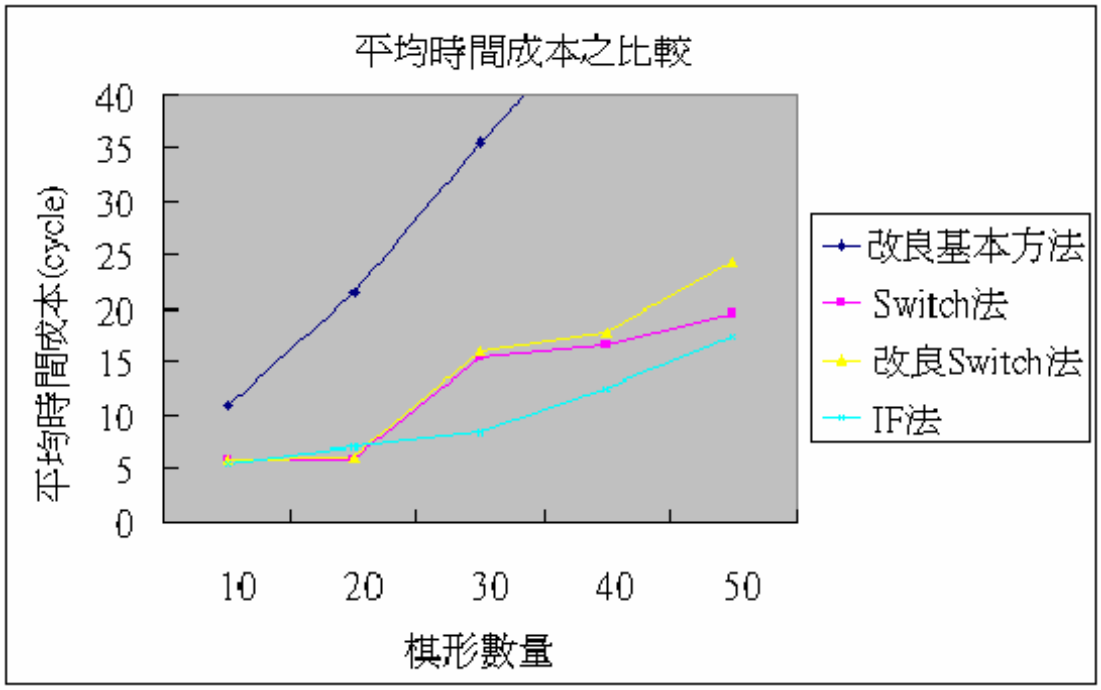
圖 4-7 IF 法子棋形提出範例

第五章　實驗數據及實作結果

本章實作出各種棋形辨識程式碼，並比較其平均時間成本及空間成本。接著將棋形辨識程式碼實際應用於象棋 AI 程式中，測驗其效能及實際應用價值。

5.1 平均時間成本之比較

在圖 5-1 中可發現，基本方法的平均時間成本非常高，難以使用，其它三種方法的平均時間成本皆比基本方法的少很多，彼此差距不大，皆為可接受範圍。觀察 Switch 法與改良 Switch 法的曲線我們可以發現，改良 Switch 法因為在某些節點使用成本較高的 if 來代替 switch，成本較 Switch 法有些微提高，而因為 if 的執行時間較 switch 來得少，使得 if 程式碼的時間成本比前述兩種 switch 程式碼都低一些。上述為棋形數量少於 50 個時所觀察得的結果，當棋形數量再提高時，由曲線圖走勢，預測 Switch 法的時間成本會低於 IF 法。



假設 if 執行時間為 1 cycle, switch 執行時間為 3 cycles

圖 5-1 平均時間成本之比較

5.2 空間成本之比較

圖 5-2 中可發現，四種方法的程式碼行數差距明顯，Swich 法在 50 種棋形時程式碼行數高達近百萬，不可能實際使用，改良過的 switch 程式碼行數已經控制在萬行以內，雖然已可勉接受，但仍然偏高，若棋形數量再提高，表現會愈來愈差。僅有基本方法及程式碼行數最低的 IF 法，在實作上能有良好的表現，依照曲線走勢，即使棋形數量再增加，程式碼行數仍然能穩定的掌握，但基本方法的平均時間成本過高，因此真正能實際應用的方法只有 IF 法。

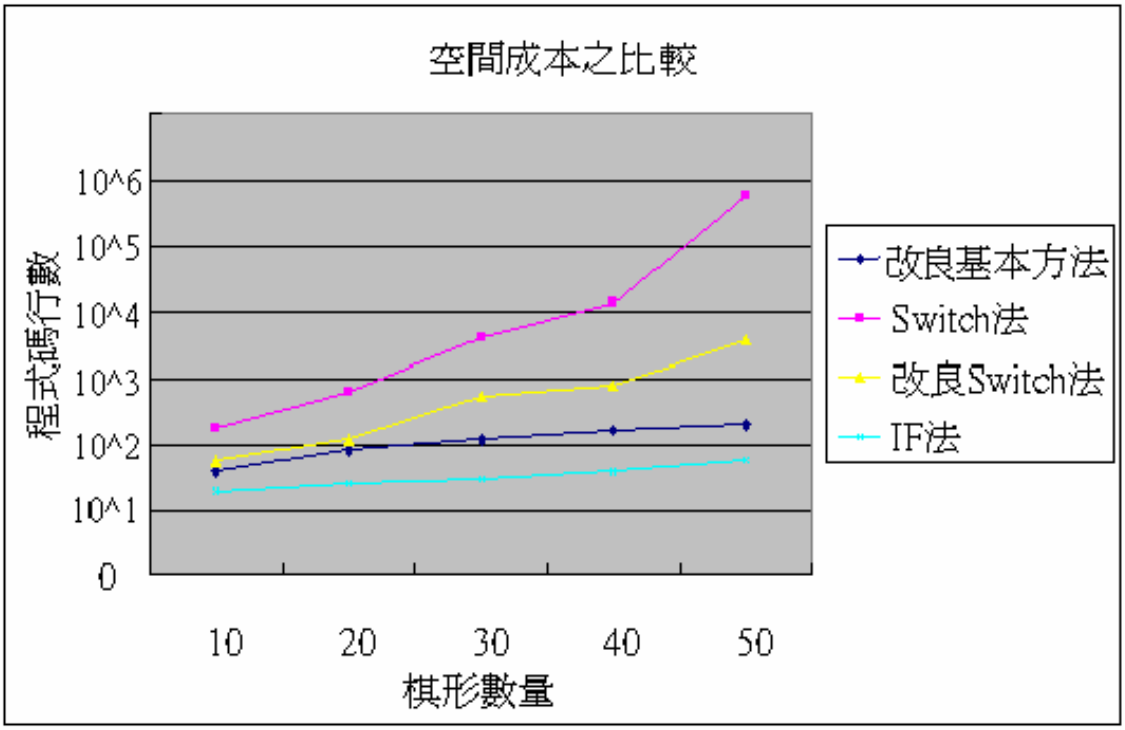


圖 5-2 平均空間成本之比較

5.3 實作結果

本篇將平均時間成本及空間成本最佳之 IF 程式碼應用於本實驗室開發之象棋 AI 程式-Chimo 上，並測試其實際運行效能，最後發現其實用價值。

實驗用之機器為 AMD Athlon(tm) XP 2500+ 1.84 GHz，在使用基本審局函式下(子力＋棋子自由度＋基本牽制)，Chimo 之運行效能原本為 1.12 million nodes/sec，在加入 50 種棋形之改良基本方法程式碼後，為 0.39 million nodes/sec；加入 50 種棋形之改良 Switch 法程式碼後，為 0.66 million nodes/sec；加入 50 種棋形之 IF 程式碼後，為 0.74 million nodes/sec。使用效能最佳之 IF 程式碼，效能僅損失 34%(不到半層的搜尋深度)，並且成功辨識出各種棋形，並且獲得棋力的提升。

只要有加入棋形的盤面，Chimo 皆能有效的依照給予盤面分數的高低，判斷優劣，選擇避開或驅往此盤面，因此加入棋形的質與量，以及給予棋形的相對應分數，將是棋形辨識系統能否有效幫助棋力提升的關鍵。若加入的棋形不夠全面，棋形的數量不夠多，或是給予的相對應分數不夠準確甚至是錯誤，皆會導致棋力無法有效的提升，甚至是降低棋力。因此本棋形辨識系統的運用方式，會是另一個重要的關鍵。

圖 5-3 左邊為一基本殘局－海底撈月，為硨炮必勝單車的殘局，在僅使用基本搜尋技巧之象棋 AI 程式下，需要搜尋 26 層，花費時間超過一小時，才能正確解出殺法，在加入右邊的棋形辨識加分後，僅需 7 層，花費時間一秒以內便能正確求得解法。

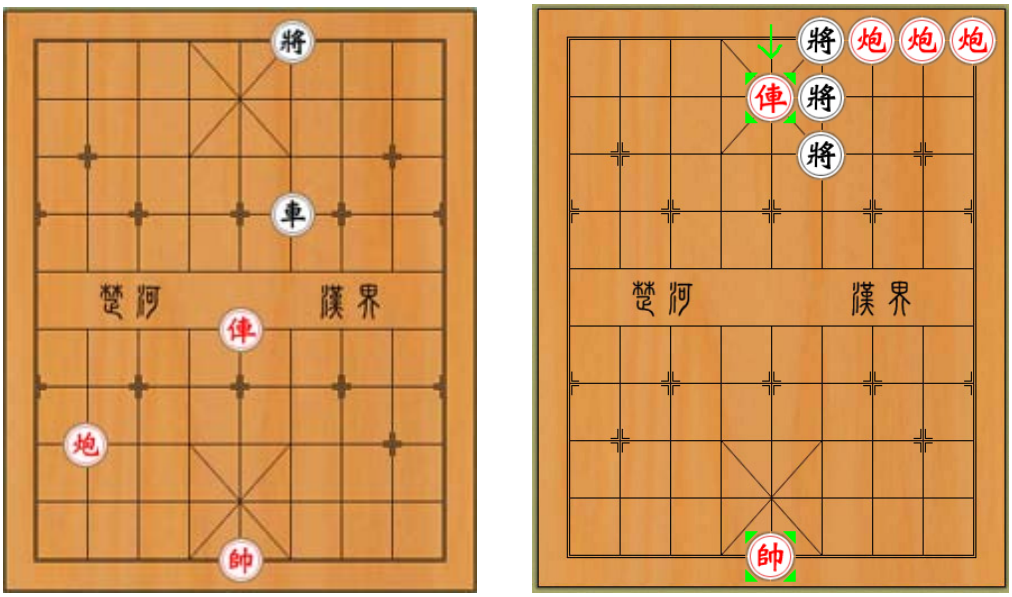


圖 5-3 海底撈月盤面及棋形

第六章　結論與未來展望

本篇論文主要的研究成果為，研究並改良象棋相關棋類的棋形辨識方法，評估並測量其效能，實際應用於象棋 AI 程式上，獲得棋力上的提升。

在未來的發展上，有以下幾個方向可以持續進行：

1. 將棋形檔編譯成更有效率的程式碼
2. 加入更多的棋形並測量其辨識效能
3. 棋形辨識應用方向之研究

以象棋作為研究背景，我們嘗試推廣原有的棋形辨識技術應用在象棋上面，獲得不錯的結果，並且以這些成果直接或間接的了解棋形辨識的重要性。另外一方面，我們也期望能延伸類似的想法到其餘的棋類遊戲上，對人工智慧的研究上也有相當大的幫助。

– 參考文獻

[1] P. Drake, J. Levenick, L. Veenstra, and A. Venghaus. (2004). Pattern matching in the game of Go.

[2] Fang, H., Hsu, T., and Hsu, S.C. (2001). Construction of Chinese Chess Endgame Databases by Retrograde Analysis. Computers and Games, Second International Conference, CG 2000 (eds. T.A. Marsland and I. Frank), pp. 96-114, Vol. 2063 of Lecture Notes in Computer Search. Springer-Verlag, Berlin. ISBN 3-540-43080-6.

[3] Fang, H., Hsu, T., and Hsu, S.C. (2003). Indefinite Sequence of Moves in Chinese Chess Endgames. Lecture Notes in Computer Science: Proceedings of the 3rd International Conference on Computers and Games (eds. J. Schaeffer and M. Muller). Springer-Verlag, New York, N.Y. (to be published).

[4] J. Gould and R. Levinson. (1992). *Experience-based adaptive search*. In Machine Learning:A Multi-Strategy Approach, volume 4. Morgan Kauffman.

[5] Reijer Grimbergen. (1996). Using Pattern Recognition and Selective Deepening to Solve Tsume Shogi , Game Programming Workshop in Japan '96 .

[6] Reijer Grimbergen, Hitoshi Matsubara. (1997). Pattern Recognition for Candidate Generation in the Game of Shogi, Proceedings of the Workshop on Computer Games (W31) at IJCAI-97.

[7] Jaap van den Herik H, Jos W Uiterwijk, and Jack van Rijswijck. (2002). *Games solved: Now and in the future*. Artificial Intelligence, 134:277—311.

[8] R.A. Levinson and R. Snyder. (1991). Adaptive pattern-oriented chess. In L. Birnbaum and G. Collins, editors, Proceedings of the 8th International Workshop on Machine Learning, pages 85--89.

[9] M. Muller. (1991). Pattern matching in explorer. In In Proceedings of the Game Playing System Workshop, Tokyo, Japan, ICOT.

[10] Wu, R. (2001). A Memory Efficient Retrograde Algorithm and its Application to Chinese Chess Endgames. Ph.D. thesis. (Submission due December 2001).

[11] Wu, R. and Beal, D.F. (2001a). Computer Analysis of some Chinese Chess Endgames. Advances in Computer Games 9 (eds. H.J. van den Herik and B. Monien), pp. 261-273. IKAT, Universiteit Maastricht, The Netherlands. ISBN 90 6216 5761.

[12] Wu, R. and Beal, D.F. (2001b). Solving Chinese Chess Endgames by Database Construction. Information Sciences, Vol. 135, Nos. 3-4, pp. 207-228. ISSN 0020-0255.

[13] Wu R, Beal D.F. (2001c). Fast, memory-efficient retrograde algorithms ICGA J 24 (3): 147-159 SEP 2001.

[14] Tanguy Urvoy and gnugo team. Pattern Matching in Go with DFA.

[15] Walczak, S. (1992). Pattern-based tactical planning. International Journal of Pattern Recognition and Artificial Intelligence, 6(5), 955988.

[16] D. Wilkins. (1980). Using patterns and plans in chess. Artificial Intelligence, 14:165--203.

[17] Shi-Jim Yen, Jr-Chang Chen, Tai-Ning Yang, Shun-Chin Hsu, (March 2004), “Computer Chinese Chess,” ICGA Journal, Vol. 27, No.1, pp. 3-18, ISSN 1389-6911.

[18] 文亞成. (1990). 電腦圍棋程式中棋形辨識處理之研究，淡江大學資訊工程研究所，碩士論文.

[19] 方浩任, (1997). 電腦象棋殘局知識庫系統之製作與應用，台灣大學資訊工程研究所，碩士論文.

[20] 吳光哲, (2005). 電腦象棋搜尋圖歷史交互作用問題之研究，臺灣大學資訊工程學研究所，碩士論文.

[21] 李佑堂. (2004). 應用Patricia Tree做爲圍棋棋形辨識處理之研究，國立東華大學，碩士論文.

[22] 張躍騰, (1981). 人造智慧在電腦象棋上的應用，台灣大學電機工程研究所，碩士論文.

[23] 曹國明, (1988). 智慧型中國象棋程式的設計與製作，台灣大學資訊工程研究所，碩士論文.
[24] 許舜欽, (1990). 電腦西洋棋和電腦象棋的回顧與前瞻，電腦學刊，第二卷，第二期，頁1—8.
[25] 許舜欽, (1990). 電腦象棋程式的設計與製作，電腦學刊，第二卷，第四期, pp.1-11.
[26] 許舜欽, (1991). 電腦對局的搜尋技巧，台大工程學刊，第51期, pp.17-31.
[27] 許舜欽，陳志昌，李天佑, (1999). 網路應用於電腦象棋學習系統之設計與製作，高速計算世界，第七卷，第四期, pp.12-16.
[28] 許舜欽，黃東輝, (1985). 中國象棋知識庫之設計與製作, 1985年全國計算機會議論文集，頁505—509.
[29] 許舜欽、林益興, (1991). 電腦象棋的盲點解析，電腦學刊，第三卷，第四期, pp.1-6.
[30] 陳世和, (1999). 象棋實用殘局知識庫系統之設計與製作，八十八年全國計算機會議論文集，淡水, pp. A103-107.
[31] 陳志昌, (1998). 電腦象棋開局知識庫系統之設計與製作，國立台灣大學資訊工程研究所，碩士論文.
[32] 陳志昌, (2005). 電腦象棋知識庫系統之研製，臺灣大學資訊工程學研究所，博士論文.
[33] 詹杰文, (1991). 一個基於知識的象棋殘局系統之研製，交通大學資訊科學研究所，碩士論文.
[34] 蔡旭程, (2005). 高效率電腦象棋停著殺程式之設計與製作，臺灣大學資訊工程學研究所，碩士論文.
[35] 顏士淨，楊泰寧，董昱騰, (2003). 電腦象棋程式達奕之設計與製作，第八屆人工智慧與應用研討會.