國立交通大學

資訊科學與工程研究所

碩士論文

象棋殘局庫之研究

The Study of Chinese Chess Endgame Databases

研 究 生：曾汶傑

指導教授：吳毅成　教授

中 華 民 國 九 十 七 年 八 月

象棋殘局庫之研究

The Study of Chinese Chess Endgame Databases

研 究 生：曾汶傑　　Student：Wen-Jie Tseng

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
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

August 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年八月

象棋殘局庫之研究

研究生：曾汶傑　　指導教授：吳毅成

國立交通大學 資訊科學與工程研究所

摘要

回溯分析演算法已經應用在許多棋類 AI 程式，中國象棋因為棋規問題無法直接套用傳統的回溯分析演算法，最近幾年方浩任等人研究出建立包含亞洲棋規的象棋殘局庫，使象棋殘局庫變的完整。為了提升象棋 AI 程式的棋力，需要花費的大量的時間與空間來建立象棋殘局庫。

本篇論文研究象棋殘局庫，利用象棋的兵卒走法的特性分割象棋殘局庫，並提出使用映射法進一步壓縮象棋殘局庫，最後評估不同方法對於象棋殘局庫的建立時間與使用空間的影響。

Pattern Recognition in Chinese Chess.

Student: Wen-Jie Tseng Advisor: I-Chen Wu

Department of Computer Science and Information Engineering

National Chiao Tung University

ABSTRACT

Chinese Chess endgame databases including Asia Rule were constructed several years ago. In order to enhance Chinese Chess AI program at the endgame stage, it needs to spend a lot of time and space to create Chinese Chess endgame databases.

This paper uses the characteristic of pawn of Chinese Chess to split Chinese Chess endgame databases and then use a mapping technique to compress them. And evaluate the impact of the time and space cost in different methods for the construction of them.

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。

此外特別要感謝汪益賢學長和群想網路科技的林育用，他們在研究的過程中給予我許多寶貴的意見和指導，以及同實驗室裡一起奮鬥的夥伴威霖、哲毅、秉儒、榮欽、典餘、偉傑、鼎量和學弟們的協助。當然，還有在我遇到瓶頸時給我鼓勵的朋友們。在我的研究期間，給了我相當多的關懷與鼓勵，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、妹妹，在我的求學生涯中給了我最大的支持和照顧。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

1.1 前言

象棋是一種在華人世界較廣為流傳的二人棋盤遊戲，一般分為紅方和黑方；在縱九路、橫十路的棋盤上，雙方各有十六顆棋子，分別有將(帥)、士(仕)、象(相)、車(俥)、馬(傌)、包(炮)、卒(兵)七種子力，遊戲進行時雙方運籌帷幄輪流走子，目標是吃掉對手的王(將帥)，當其中一方的王被吃後遊戲就結束。

象棋棋局進行時，人類棋手大致上會把棋局分成三個階段(Stage)：開局(Opening stage)、中局(Middle game stage)和殘局(Endgame stage)，但三個階段彼此之間並無明顯的分界；而象棋 AI 程式在不同的階段通常會使用不同的方法找出最佳棋步。

開局階段一般為從遊戲開始到布局結束，此時雙方大都不急著吃對手的棋子，而著重於把自己的子力走到有利位置，以利其後中局的攻殺，人類棋手已經研究出許多常走的布局類型，而且布局類型不多，所以可用人為判斷的方式，或使用統計的方式，計算出有利的布局棋步，再儲存到資料庫中，此資料庫即為「象棋開局庫」(Chinese Chess Opening Databases)[19][21]，因為這是用人工或統計的方式所建立，所以記錄的棋步不一定是最佳的，但大多是「正著」，而象棋 AI 程式在開局時就可直接使用象棋開局庫中記錄的棋步，除了節省計算時間以外，也防止象棋 AI 程式因為計算錯誤走出「劣著」。

中局階段一般為從布局結束而雙方開始攻殺至殘餘子力不多時，此階段著重於如何攻殺及吃到對手的王，並且承上啟下，不僅銜接開局布下的基礎，還要將局勢往有利的殘局發展；中局盤面複雜多變，象棋 AI 程式通常使用「審局函式」(Evaluation Function)[15][22]

計算盤面分數判斷孰優孰劣，並搭配「Alpha-beta 搜尋」(Alpha-Beta Search)[15][22]演算法來找出最佳棋步，一般中局搜尋引擎大多可計算出十多步內的棋步，但搜尋方向及結果都是依賴「審局函數」的判斷，因此審局愈精準則搜尋結果愈精準；但搜尋的深度也會影響到搜尋結果，因為中局階段經常發生兌子、棄子、奪勢、牽制、爭先…等等，這些都會影響到勝負，搜尋愈深就能看到愈多步之後的結果，而審局判斷的愈多就愈花時間，搜尋的深度也就會因此而變淺，因此象棋 AI 程式必須在審局與搜尋深度之間作取捨[20]。

當殘餘子力不多時即進入殘局階段，但有時可能中局就分出勝負，所以若中局無法分出勝負就會進入殘局階段；自古以來，因為殘局子力很少，人類棋手就把殘局分成許多殘局類型，並探討不同類型的殘局孰優孰劣(例如「孤兵擒王」、「高低兵例勝雙士」…等等)，用於引導中局進入對我方有利的殘局類型；而審局函數到了殘局階段往往很難真正判斷盤面優劣，例如「俥炮對單車」的殘局類型通常有俥炮方需使用海底撈月來贏棋，但審局函數通常會認為炮走到將後面是不好的，並且很多殘局盤面的勝負是在二三十步之後才能分出，中局搜尋引擎很難達到這個深度；當棋局進入殘局時，為了彌補象棋 AI 程式在殘局的弱點，並且因為殘局子力很少，可使用窮舉法把所有的盤面的資訊事先計算出來，再儲存到資料庫中，此資料庫即為「象棋殘局庫」(Chinese Chess Endgame Databases)[1][9][10][16][18]；如果有事先計算出相符合的殘局類型的殘局庫，象棋 AI 程式在殘局階段就可以直接查詢象棋殘局庫來找出最佳棋步，以節省象棋 AI 程式的搜尋時間，而且比起開局庫和中局搜尋，殘局庫的資料是完全正確的，所以可以彌補中局搜尋的不準確性以及搜尋深度不足，甚至在進入殘局前中局搜尋就可以引導入有利的殘局盤面。

1.2 研究動機及目的

當棋局由中局階段進入殘局階段時，雙方的攻殺接近尾聲，就表示即將分出勝負了，此時若能以殘局庫輔助搜尋，象棋 AI 程式的棋力必能有所提升。雖然殘局庫的資料是完全正確的，但如果建立的殘局庫太少，對象棋 AI 程式幾乎是沒用的，因為大部份的棋局可能在用到殘局庫前就已經分出勝負，或甚至沒有進入已建立的殘局庫類型；例如如果只建立雙方所有殘餘子力加總起來小於或等於五子的象棋殘局庫時，去掉將帥，雙方各剩一至三顆子力，這些殘局類型在現實棋局中往往在進行到這些盤面之前就分出勝負了。因此，若象棋 AI 程式要靠殘局庫提升棋力就要盡可能地建立夠多的殘局庫，對象棋 AI 程式才有實質上的幫助。

但是殘局庫是使用窮舉所有盤面的方式計算出來的，所以殘局庫的資料量是非常大的，當增加殘局庫中的殘餘子力數時，殘局庫所佔用的空間會非常迅速的增加，例如一隻俥在棋盤上有九十個可能的位置，在殘局庫中增加一隻俥時，就要把俥在這九十個位置的所有盤面的資料計算出來，增加量非常驚人。因此，當增加殘局庫中的殘局類型時容易產生以下兩種問題：

1. 儲存空間不足：多大的殘局庫對象棋 AI 程式才是有幫助的呢？如果象棋 AI 程式能愈早在象棋殘局庫中查詢到盤面的最佳棋步，則象棋 AI 程式的棋力就愈強，但這就需要愈多的殘局庫，所以在有限的儲存空間建立愈多的象棋殘局庫，對象棋 AI 程式愈有幫助，因此造成儲存空間會影響到象棋 AI 程式的棋力，所以需要大量的儲存空間。
2. 記憶體空間不足：資料在記憶體(Memory)中的存取速度遠快於在硬碟(Disk)中的存取速度，所以我們會盡量把資料載入到記憶體中處理，但是由於象棋殘局庫的資訊量很大，往往會大於記憶體空間所能容納的量，而一般產生殘局庫的演算法需要不斷的對資料作隨機存取(Random Access)，因此在產

生大於記憶體空間的象棋殘局庫時，會不斷對硬碟作隨機存取的動作，I/O(Input/Output)的次數也會大大的增加，於是便需要使用特殊的演算法，但也無可避免對硬碟作大量的存取動作；所以大於記憶體空間的象棋殘局庫的產生速度會相對慢很多[13][14]。

本篇論文的目的在於研究如何壓縮象棋殘局庫所佔用的空間，並且探討如何在產生象棋殘局庫盡量減少 I/O 的次數；而壓縮象棋殘局庫的議題可分為兩種：一種是壓縮象棋殘局庫大小但不失完整的盤面資訊，另一種是縮減殘局庫中記錄的盤面資訊但不影響下棋結果，本篇論文將就使用這兩方面提出不同的方法並評估其所帶來的效益。

1.3 論文大綱

本篇論文在第二章介紹前人在象棋殘局庫上的研究，包括盤面資訊(Position Value)、回溯分析(Retrograde Analysis)演算法與相關壓縮(Compression)技術。在第三章說明我們提出的分割象棋殘局庫的方法與其結構特性。在第四章說明我們在記錄象棋殘局庫的盤面資訊時所使用的實作方法，並分析不同方法產生象棋殘局庫所花費的空間成本與時間成本的實驗數據並討論實驗結果。在第五章提出我們的結論與未來展望。

第二章、相關文獻及基礎理論

早期的象棋殘局庫礙於棋規問題[1][17]，不能直接將西洋棋建立殘局庫的方法直接套用到象棋上，因為象棋雙方都有攻擊子力(即車(俥)、馬(傌)、包(炮)和卒(兵))的盤面才有可能犯棋規，所以不能建立雙方都有攻擊子力的殘局庫，其後方浩任等人提出建立包含「亞洲棋規」(Asia Rule)的「長將」盤面的殘局庫[2][3]，但也未能完全解決棋規問題，直到最近幾年方浩任才再提出建立包含「亞洲棋規」的「長捉」盤面的殘局庫，因此現在已能建立出包含完整的「亞洲棋規」的殘局庫[4][5]，雖然仍未解決「中國棋規」問題，但已大大增加了實用性。

本章介紹象棋殘局庫過去的發展狀況與其所使用的回溯分析演算法理論，並介紹其所使用的相關的改良技術，以及如何解決棋規問題。第 2.1 節介紹象棋殘局庫的分類與結構。第 2.2 節介紹象棋殘局庫的壓縮技術。第 2.3 節介紹在殘局庫中所記錄的資訊。第 2.4 節介紹產生殘局庫所使用的回溯分析演算法。第 2.5 節介紹象棋的棋規問題及其相關解法，並介紹在殘局庫中如何記錄棋規資訊。

2.1 殘局庫結構(Structure of Endgame Databases)

象棋殘局通的結構與西洋棋是類似的，我們先定義一個「盤面」是包含「特定的殘餘子力」與「每個子力的所在位置」以及這個盤面的「先走方」，而一個「合法的盤面」必須每個子力的所在位置都是合法位置，而一個合法盤面能經由合法棋步走到的盤面則稱作是它的「子盤面」；象棋殘局通常會將所有有相同殘餘子力數量與種類的盤面歸為同一個類型，象棋殘局庫的結構也是以此為基準，命名原則上是以左邊為紅方殘餘子力，右邊為黑方殘餘子力，並以殘餘子力的英文縮寫為其命名[1]，例如：「單俥對士象全」是指所有紅方殘餘一隻俥，黑方殘餘兩隻士、兩隻象的所有盤面，即為ＫＲＫＡＡＥＥ。

棋子	縮寫
將帥(King)	K
士仕(Assistant)	A
象相(Elephant)	E
車俥(Rook)	R
馬傌(Horse)	H
包炮(Cannon)	C
卒兵(Pawn)	P

表 2-1 象棋各種子力縮寫

象棋殘局庫通常以「先走方」與「殘局類型」做分割[1][9][10]，並將相同先走方與殘局類型的所有盤面資訊存在同一個資料庫裡：

1. 以「先走方」分割象棋殘局庫：每個象棋殘局庫會分成紅先走與黑先走兩個資料庫。在紅先走殘局庫中的盤面的所有合法盤面的子盤面都在黑先走殘局庫中，反之亦然。
2. 以「殘局類型」分割象棋殘局庫：有相同的殘餘子力數量與種類的所有盤面資訊都存在同一個象棋殘局庫中。因此，一個合法盤面經由非吃子步所走到的子盤面仍在同一個殘局庫中(因為殘餘子力仍然相同)，而經由吃子步所走到的子盤面必然在另一個不同的殘局庫中(因為殘餘子力不同)；在此定義同一個殘局庫中的所有盤面能經由吃子步所走到的各種類型的子盤面所在的殘局庫為其「子資料庫」(Supporting Database)，例如 KPK 與 KKE 皆是 KPKE 的子資料庫。象棋殘局庫結構如圖 2-2。

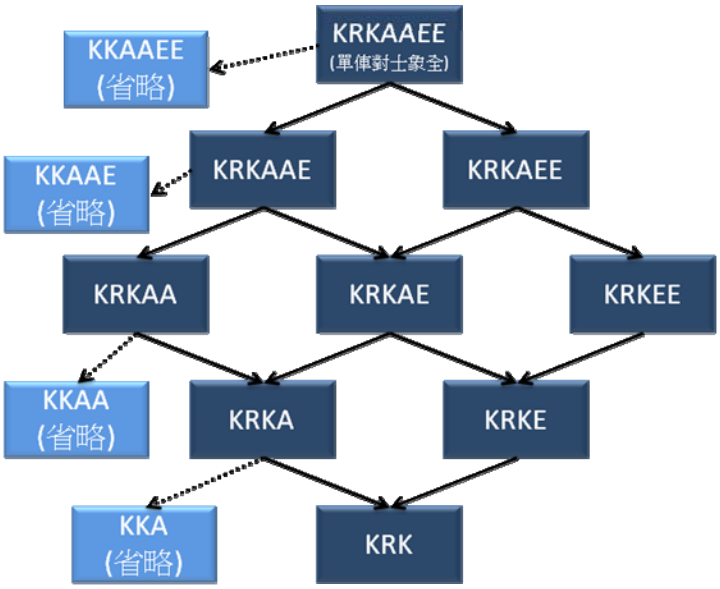

圖 2-2 象棋殘局庫結構(單俥對士象全)

將象棋殘局庫分割出許多子資料庫後，因為吃子後子盤面便轉換到子資料庫，而這轉換是不可逆的，建立象棋殘局庫時便必須先建立所有子資料庫。但數個象棋殘局庫可能會有相同的子資料庫，例如 KRKAE 是 KRKAAE 與 KRKAEE 的子資料庫，所以可以節省儲存相同子資料庫的空間成本與建立相同子資料庫的時間成本。有些子資料庫是互相獨立的，例如 KRKAAE 與 KRKAEE，這些互相獨立的子資料庫不會互相影響其盤面結果，所以可以平行處理。另外，象棋殘局庫在經過分割之後，有些子資料庫會夠小而能夠完全在記憶體中處理，所以象棋殘局庫能夠分割的愈小就愈好。

2.2 壓縮方法(Compression Methods)

窮舉所有盤面時會利用雜湊函數(Hash Function)將每個盤面對應不同的雜湊值(Hash Value)，並利用這個值當做盤面的索引值(Index Value)，而盤面資訊就是存在這個索引位址上，所以在查詢盤面資訊時會把這個索引值當做查詢位址，這個雜湊值是由盤面上所有棋子的位置編碼計算所得到的，所以在象棋殘局庫中就不必紀錄盤面上所有棋子的位置。

因此，盤面索引值的範圍愈小，則所需儲存空間越小。以下介紹四種常用的，能夠不失盤面資訊，而是藉由改變編碼方法以縮小盤面索引範圍的壓縮方法[13]。第 2.2.1 節介紹去除棋子在不合法位置的盤面，第 2.2.2 節介紹去除垂直對稱的盤面，第 2.2.3 節介紹去除同類子互換位置的盤面，第 2.2.4 節介紹去除棋子位置重疊的盤面。

2.2.1 棋子合法位置(Pieces' Legal Addresses)

棋子在不合法的位置的盤面是不合法的，所以只需要記錄子力在合法位置上的盤面，如此可以大幅降低盤面索引範圍。例如將有 9 個合法位置，壓縮率為 9/90；士有 5 個合法位置，壓縮率為 5/90。

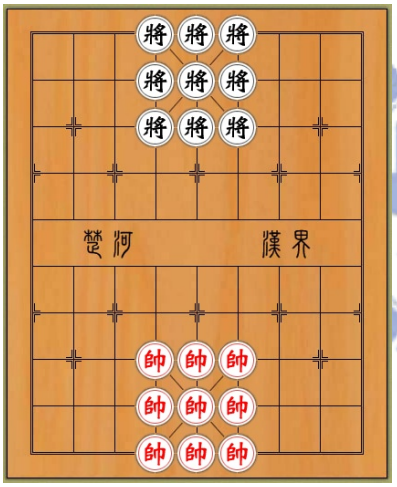

圖 2-3(a) 將帥合法位置

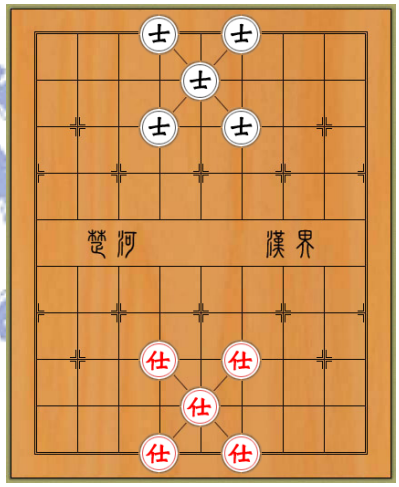

圖 2-3(b) 士仕合法位置

2.2.2 垂直對稱(Vertical Symmetry)

垂直對稱的兩個盤面只需要記錄其中一個。在查詢這兩個盤面時，可藉由左右翻轉其中一個盤面來查詢最佳棋步，因此可以認為這兩個盤面是等價的(如圖 2-4(a))；除了左右完全對稱盤面(如圖 2-4(b))，所有盤面都有與其垂直對稱的另一個盤面，因此這種方法可使幾乎一半的盤面索引用不到，所以這種壓縮方法的壓縮率接近 50%。

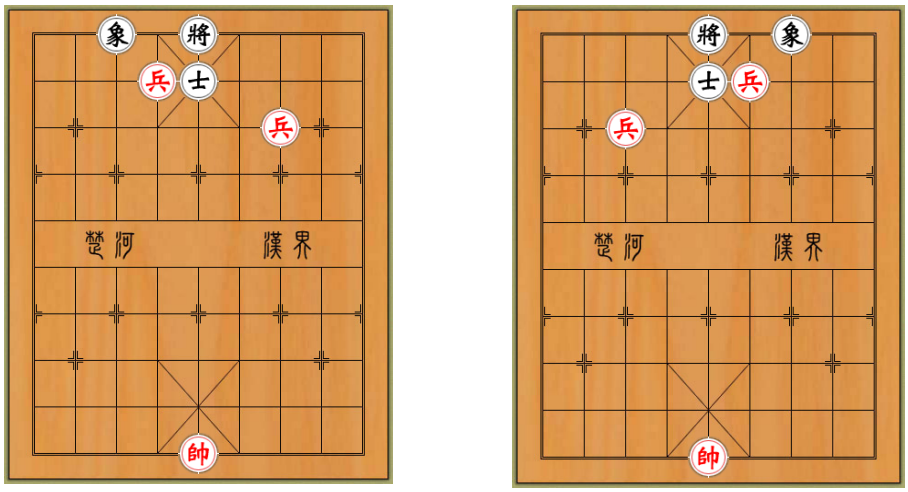

圖 2-4(a) 垂直對稱的兩個盤面(紅先走)例子

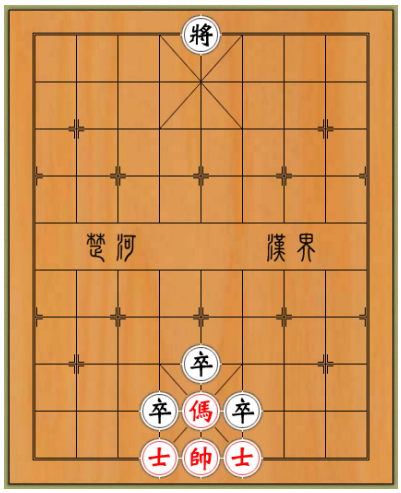

圖 2-4(b) 左右完全對稱的盤面(紅先走)例子

2.2.3 同類子互換(Multiple Piece Symmetry)

若一個盤面有兩個以上同類子時，同類子的位置互換在人類棋手眼中會認為是同一盤面，與垂直對稱盤面同理，這些盤面是等價的，只需要記錄其中一個。因為盤面索引是根據每個棋子的位置來編碼計算的，所以同類子互換位置的兩個盤面會被認為是不同的盤面，用了此方法後，例如兩隻俥的位置互換，兩隻俥的位置數由 90*90 降到 90*89/2，兩隻兵的位置互換，兩隻兵的位置數由 55*55 降到 55*54/2。

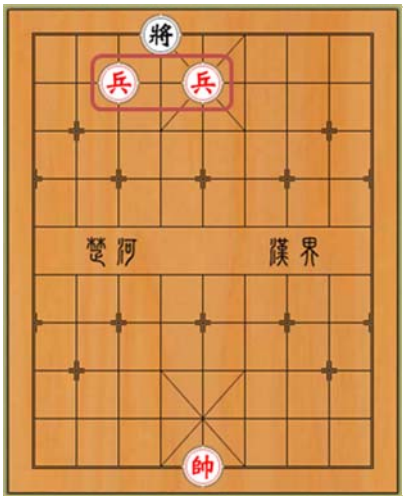

圖 2-5 有同類子(雙兵)的盤面(紅先走)例子

2.2.3 棋子群組(Piece Grouping)

帥和仕與將和士的位置的在九宮內，因此這些棋子的合法位置的重疊率相當高，由於根據每個棋子的位置編碼盤面索引，所以兩個以上棋子在同一個位置的盤面也會占用索引值，但這種盤面是不合法的；藉由把帥和仕與將和士看成是同一個組合，可以降低不合法盤所占用的索引值。例如有單仕的象棋殘局庫，帥和仕的組合位置數由 9*5 降到 8*5，有雙仕的象棋殘局庫，帥和仕的組合位置數由 9*5*5 降到 7*5*4。

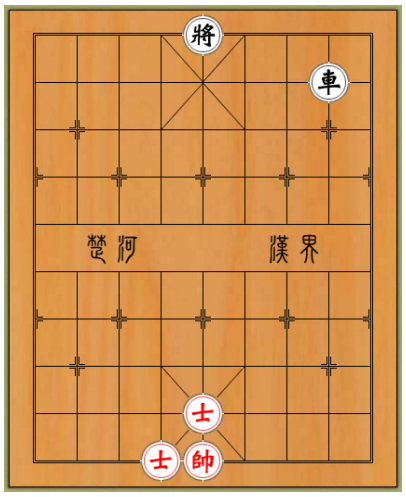

圖 2-6 有雙仕的盤面(紅先走)例子

2.3 盤面資訊(Position Value)

在一般殘局庫中，有三種型態的盤面資訊是最常使用的，不同型態的盤面資訊各有其優缺點，以下介紹三種不同型態的盤面資訊[1]。第 2.3.1 節介紹勝-敗-和盤面資訊，第 2.3.2 節介紹 DTM 盤面資訊，第 2.3.3 節介紹 DTC 盤面資訊。另外，這裡介紹的盤面資訊會忽略象棋的 50 回合規則(50-move-rule)(象棋的雙方在 50 回合裡都無吃子則判為和棋)。

2.3.1 勝-敗-和(Win-Loss-Draw)

象棋的每個盤面最終會是三種結果之一：必勝、必敗或必和。在此定義這三種盤面：

1. 必勝盤面表示先走方最少有一種方法能吃到對手的王、強迫對手走到無任何合法步的盤面或強迫對手違反棋規。
2. 必敗盤面表示先走方的所有合法棋步都會走到對方必勝盤面或無合法棋步的盤面(如圖 2-4(b))。
3. 必和盤面表示先走方能避免對手走到必勝盤面，反之亦然。

最簡單的盤面資訊就是只記錄盤面結果，因此每個盤面只可能有勝/敗/和三種狀態(Win/Loss/Draw States)，所以每個盤面最多只需用 2 bits 的空間，但是這樣的資訊太少，並不足以找出最佳棋步。

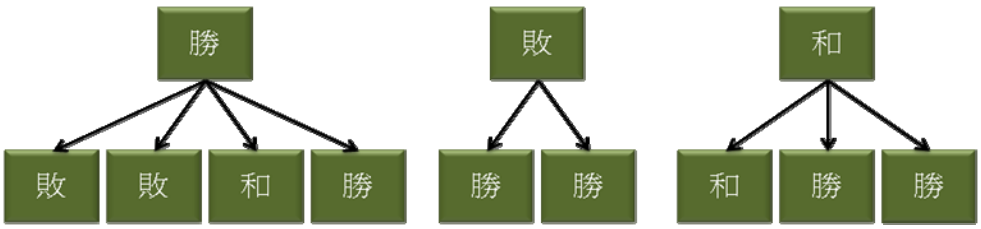

圖 2-7 只有勝敗和的盤面狀態例子

2.3.2 DTM(Distance-To-Mate)

為了找出最佳棋步，就必須增加盤面資訊，通常會使用「DTM」：對於必勝盤面，代表最少需多少棋步能贏；對於必敗盤面，代表最多能用多少棋步抵抗對手走到贏。因為此種紀錄資訊會造成盤面資訊的值剛好比最佳棋步走到的子盤面的值多 1，所以用 DTM 找出盤面的最佳棋步的方法就是搜尋所有子盤面，並找出比此盤面的值少 1 的子盤面。

在此使用奇數棋步表示必勝(至少為 1)，偶數棋步表示必敗(至少為 2)，0 這個特殊值代表和棋。由於增加了盤面資訊，每個盤面所需儲存空間也增加了，為了區別不同的 DTM 值，每個盤面最少需 $\lceil \log_2(\text{最大棋步}) \rceil$bits(「最大棋步」(Maximum Number of Plies)為同一象棋殘局庫中所有盤面的最大 DTM 值)，例如最大棋步為 117 時則每個盤面至少需 7bits($2^7 = 128$)，有時為了方便電腦處理，通常會以 byte 為單位，最大棋步為 255 以下時使用 1byte 來紀錄，超過 255 時會使用 2bytes 來紀錄。

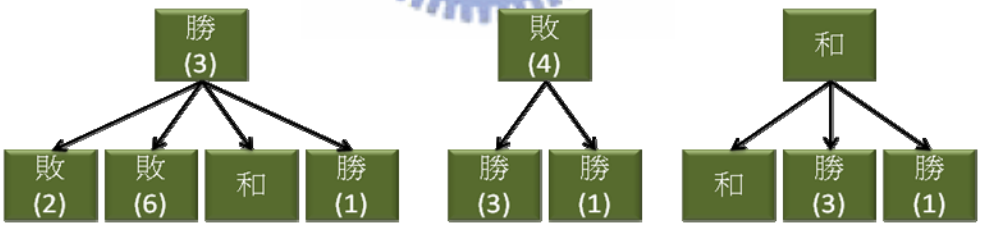

圖 2-8(a) 使用 DTM 紀錄的盤面狀態例子

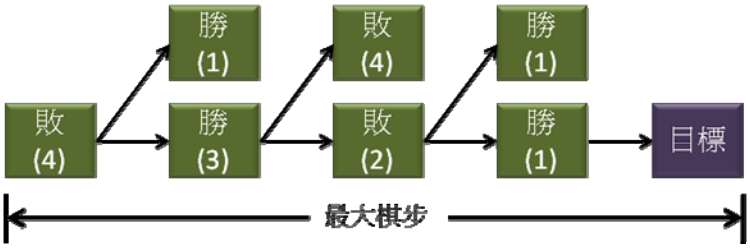

圖 2-8(b) 使用 DTM 紀錄的最佳棋步路徑例子

2.3.3 DTC(Distance-To-Conversion)

但是 DTM 值有時會隨象棋殘局庫子力增多時而增加，因為子力增多時，盤面通常愈複雜，因而使最大棋步增加。所以有時會改用「DTC」來紀錄盤面資訊：對於必勝盤面，代表最少需多少棋步能贏或藉由吃子而贏；對於必敗盤面，代表最多能用多少棋步抵抗對手走到贏或藉由吃子而贏。目的是盡量減少最大棋步，也就能減少使用空間。

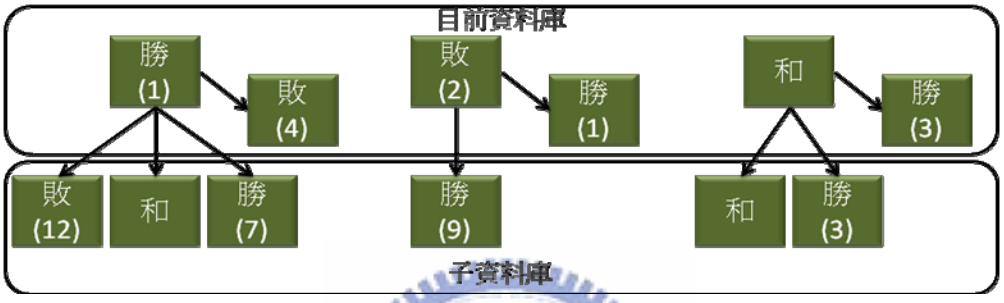

圖 2-9(a) 使用 DTC 紀錄的盤面狀態例子

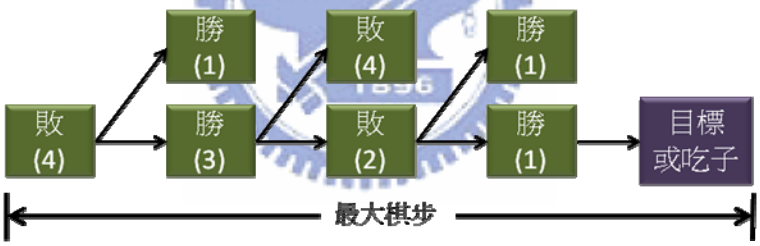

圖 2-9(b) 使用 DTC 紀錄的最佳棋步路徑例子

因為一個盤面在走到目標的路徑上有可能需要吃子，當路徑上有吃子步時使用 DTC 紀錄的值就會變小，以圖 2-10 為例，使用 DTC 時，盤面資訊所需空間由$\lceil \log_2(137) \rceil = 8$bits 降至$\lceil \log_2(55) \rceil = 6$bits；有時為了方便電腦處理，以 byte 為單位處理時，可能從 2bytes 降至 1byte。

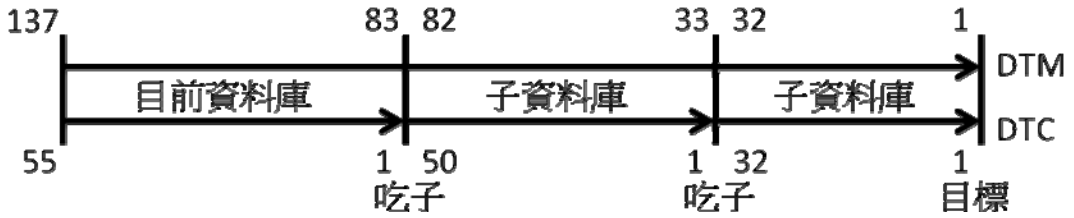

圖 2-10 使用 DTM 與使用 DTC 的差別

只是使用 DTC 有個潛在問題是必勝盤面可能比使用 DTM 更慢贏，以圖 2-11 為例，雖然卒二平三(5 步,DTM=5)比卒二平一(9 步,DTM=9)更快贏，但 DTC 會優先選擇卒二平一(DTC=1)吃掉傌，但不影響盤面最終的結果。另外，象棋的 50 回合規則對 DTC 的影響不大，因為雖然贏的慢但吃子後無吃子回合數會重新計算。

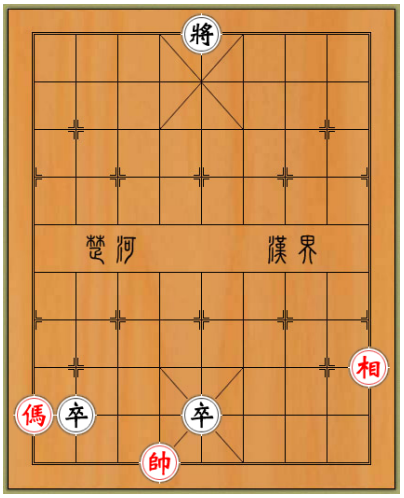

圖 2-11 吃子後更慢贏的盤面(黑先走)例子

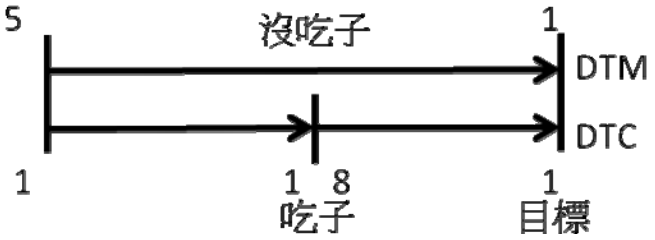

圖 2-12 吃子後更慢贏的距離差異比較圖

2.4 回溯分析(Retrograde Analysis)

一般的回溯分析演算法會分成三個階段[1][9]：

1. 初始化階段(Initialization Phase)：找出所有立刻分出能勝負的盤面，以象棋來說就是能立即吃王的盤面(必勝)、將死(Checkmate)盤面(必敗)或無合法步的(Stalemate)盤面(必敗)，並將其他盤面設成未知(Unknown)狀態。
2. 傳播階段(Propagation Phase)：使用回溯分析(Retrograde Analysis)根據第 2.3 節的定義找出必勝和必敗的盤面。
3. 最終階段(Final Phase)：將剩餘未知的盤面設成必和。

為了建立超過記憶體空間大小而改良的外部記憶體回溯分析演算法(External-Memory Retrograde Analysis)[14]，則會把要存取的盤面的索引值排序後依序存取硬碟空間。

2.5 棋規問題(Problem of Special Rule)

一般棋類遊戲常會遇到一個問題：就是盤面的重覆(Repetition of a Position)，不同的棋類有不同的處理方法；例如圍棋是使用全同禁著，西洋棋則是同一盤面出現 3 次為和棋。象棋則會在同一盤面循環 3 次(即出現 4 次)後在出現第 5 次時根據重覆的盤面之間的棋步是否皆為禁止的著法而出現不同的結果，若一方走出禁止的著法而另一方沒走出禁止的著法，則走出禁止的著法方判敗，沒走出禁止的著法方判勝，若雙方皆走出禁止的著法或雙方皆沒走出禁止的著法則判和，所以象棋殘局庫除了一般的盤面資訊也需要包含棋規資訊。而象棋通常使用亞洲棋規(Asia Rule)，所以會在象棋殘局庫中紀錄受亞洲棋規影響的盤面資訊，而亞洲棋規規定「長將」與「長捉」為違反棋規[17]，如圖 2-13 為紅俥不斷長將的盤面例子。

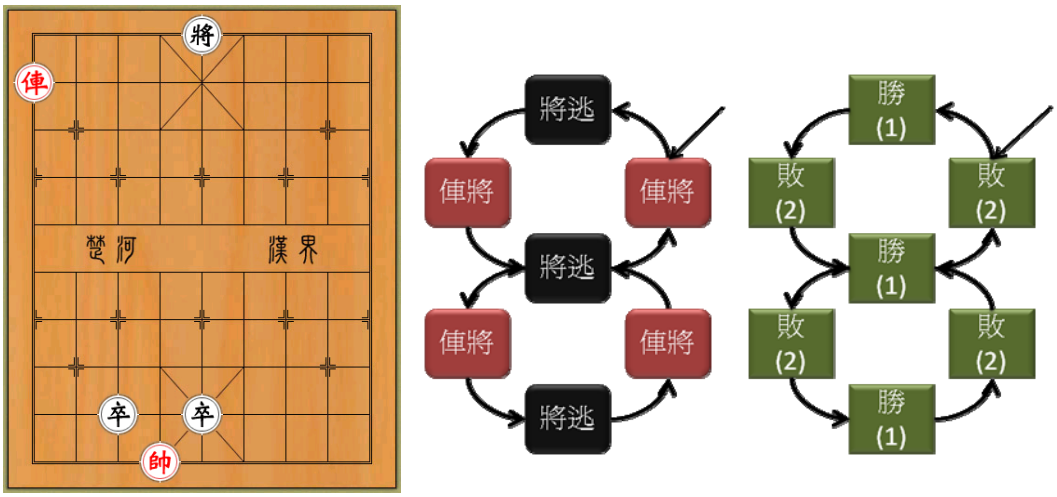

圖 2-13 長將盤面(紅先走)例子

此問題已由方浩任提出一套找出象棋重覆盤面的方法，他的演算法在一般的回溯分析演算法中的傳播階段之後加入[4][5]：

1. 找出長將盤面並用回溯分析更新其他未知盤面。
2. 找出長捉盤面並用回溯分析更新其他未知盤面。
3. 重覆 1 和 2 直到無任何重覆盤面被找到。

他使用 Distance-To-Check/Chase 紀錄盤面資訊：對於必勝盤面，代表最少需多少棋步走到重覆盤面；對於必敗盤面，代表最多能用多少棋步抵抗對手走到重覆盤面。

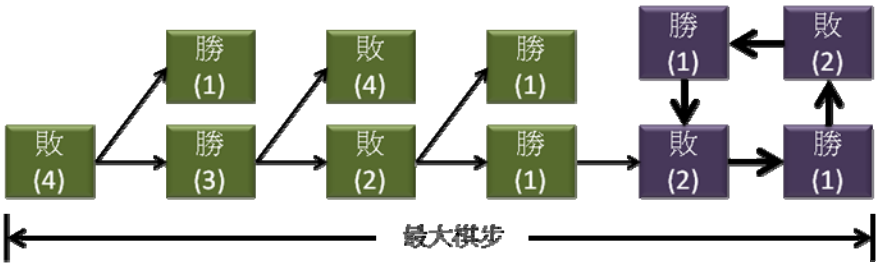

圖 2-14 使用 Distance-To-Check/Chase 紀錄的最佳棋步路徑例子

根據圖 2-15 的下棋策略，必須區別出所記錄的盤面資訊是哪一種距離(Distance)，所以在盤面資訊中加入層級(Level)以示區別；第 0 層代表最後會走到吃王的盤面，層級 1 以上代表最後會走到長將或長捉的盤面，所以第 0 層代表 Distance-To-Conversion，第 1 層代表 Distance-To-Check 或 Chase，第 2 層代表 Distance-To-Check 或 Chase，第 3 層代表 Distance-To-Check 或 Chase，……以此類推，而。因為他的演算法會先找長將再找長捉，例如在第 1 層時有找到長將盤面，第 1 層就代表 Distance-To-Check，若第 1 層沒找到長將盤面而找到長捉盤面第 1 層就代表 Distance-To-Chase。而每個盤面狀態改由(層級，距離)(Level，Distance)表示，在搜尋最佳棋步時就選擇走向同一層級中距離減 1 的子盤面，當盤面重覆至快因犯棋規而敗時時，就選擇走向較低的最大層級中距離最大的子盤面；而這就是 Distance-To-Conversion/Check/Chase。

圖 2-15 象棋下棋策略優先順序

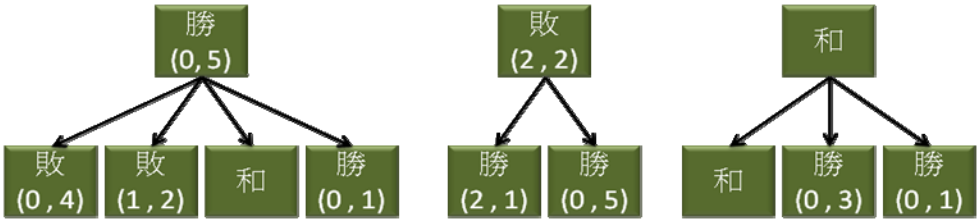

圖 2-16 使用 Distance-To-Conversion/Check/Chase 紀錄的盤面狀態例子

圖 2-17 層級與距離關係圖

第三章、分割殘局庫

本章說明如何利用兵卒的走法進一步分割象棋殘局庫，並為有兵卒的象棋殘局庫間的關係新增定義。第 3.1 節說明兵卒的走法以及用兵卒的位置來細分殘局類型。第 3.2 節說明以兵卒的位置分割象棋殘局庫時，有兵卒的象棋殘局庫會有甚麼結構，以及其結構特性將有何種效益。第 3.3 節說明以兵卒的位置分割象棋殘局庫時，盤面資訊應該做的相應的改變。

3.1 兵卒走法(The Movement of the Pawn)

象棋的兵卒走法為只能前進一格不能後退，並且在兵卒過河後，有能夠左右橫向移動一格的能力；若把兵卒在同一列上的所有盤面視為同一殘局類型會發現，當兵卒橫向移動時子盤面仍然是同一殘局類型，但當兵卒前進時子盤面便轉換到另一殘局類型，而且這轉換是不可逆的；這個特性跟吃子後子盤面轉換到子資料庫一樣，根據這個特性，可以以兵卒的位置將象棋殘局庫再做一次分割。

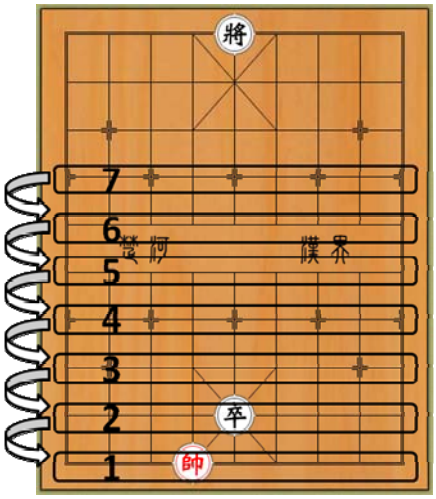

圖 3-1 卒在不同列的位置關係圖

3.2 分割殘局庫(Split Endgame Databases)

當象棋殘局庫有至少一隻兵卒時，便可以以兵卒的所在位置的來分割殘局庫；但若兵卒的所在位置為已過河後的位置(從對方底線依序算起第 1 列、第 2 列、第 3 列、第 4 列和第 5 列)，因為兵卒可橫向移動，最多只能以兵卒的所在列數作分割；可是若兵卒的所在位置為未過河後的位置(第 6 列和第 7 列)，因為兵卒只能前進而不能進行橫向移動，所以不但能以兵卒的所在列數作分割，還能以兵卒的所在行數作分割，也就是說，若兵卒在未過河位置，可以把象棋殘局庫分割成兵卒分別在10個位置(兵卒第6列和第7列有10個合法位置) 的情形。但為了簡化說明，本篇論文只討論以兵卒列數作分割的情形，並且在實作上也是以兵卒列數作分割。

為了說明，象棋殘局庫的命名改由 Pn 表示兵卒在第 n 列，例如 KP1P2K 限制所有的盤面一隻兵的位置在第 1 列，另一隻兵的位置在第 2 列。在此為與「子資料庫」(Supporting Database)作區別，在此另外定義同一個類型殘局庫中的所有盤面能經由兵卒前進(但不吃子)所走到的各種類型子盤面所在的的殘局庫為其「進資料庫」(Advancing Database)，例如 KP1K 是 KP2K 的進資料庫。

如此分割象棋殘局庫後，如圖 3-2(a)與圖 3-2(b)，和子資料庫類似，建立象棋殘局庫時也必須先建立所有進資料庫。在建立進資料庫時，也因為有些進資料庫之間是互相獨立的，例如 KP5P7K 與 KP6P6K，這些進資料庫是可以平行處理的；另外原本 KPKA 需等到 KPK 建立完後才可開始建立，KP1KA 則不必等到 KP1K~KP7K 全部建立完才能開始建立，只要 KP1K 建立完畢，KP1KA 即可開始建立。而在分割象棋殘局庫後，每個進資料庫的大小都比原來的未分割的資料庫小，因此有些進資料庫就能完全載入到記憶體中處理。

圖 3-2(a) 象棋殘局庫結構(KPK)

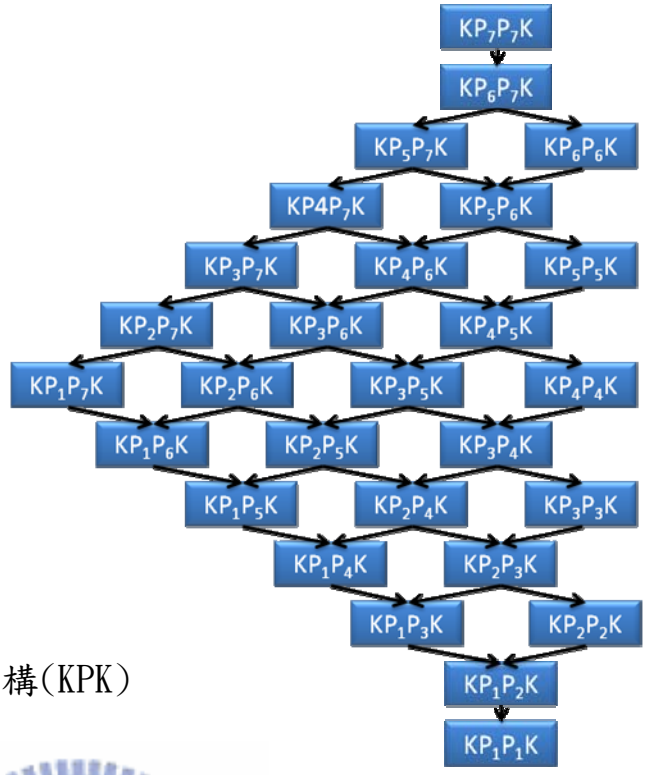

圖 3-2(b) 象棋殘局庫結構(KPPK)

3.3 盤面資訊的問題(Problem of Position Value)

前面提過，使用 DTC(Distance-To-Conversion)時會造成必勝盤面可能贏的更慢的問題，但象棋的 50 回合規則影響不大，原因是吃子後不吃子回合數會重算；而轉換到進資料庫的棋步是兵卒前進但並不吃子，所以贏的太慢可能會超過 50 回合，雖然在產生象棋殘局庫時忽略 50 回合規則，但我們還是希望它對於分割象棋殘局庫的影響不要太大，因而減少殘局庫的實用性；如果在轉換到進資料庫時使用 DTC，則 50 回合規則對象棋殘局庫的影響可能會太大，因為在一般的情況下，兵卒保持在比較高的位置的會比較有利，機動性也比較好。

為了避免分割象棋殘局庫後，因為使用 DTC 而被 50 回合規則影響殘局庫的實用性，我們建議改使用 DTM 來紀錄盤面資訊，但為了避免使用 DTM 導致殘局庫變大，如果子盤面在子資料庫時，我們仍然使用 DTC 來減少最大棋步，也就是說我們利用兵卒位置分割象棋殘局庫後混合使用 DTM 與 DTC，但 DTM 只限於當子盤面在進資料庫時，例如當兵卒前進而又同時可以吃掉一顆子時，我們還是優先使用 DTC 來記錄盤面資訊，因此盤面資訊的值會與未分割前的殘局庫的 DTC 值一模一樣，所以搜尋最佳棋步的方法也跟 DTC 一樣選擇走向距離減 1 的子盤面，這樣就可以避免優先選擇兵卒前進一步而贏的太慢，畢竟對於能贏的盤面沒有人會希望因為贏的太慢而被判和。

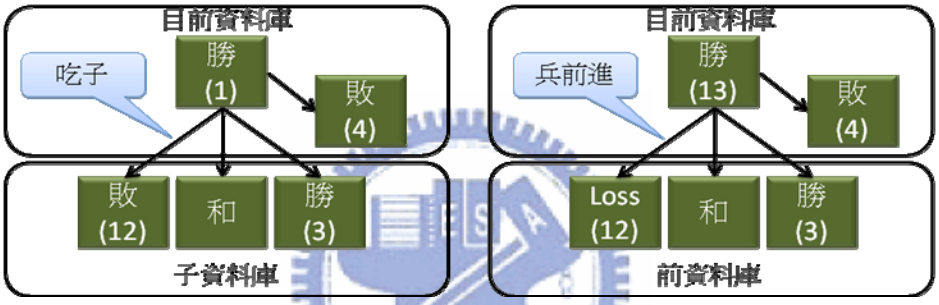

圖 3-3 DTC 與 DTM 的差別

第四章、實作與實驗結果

本章說明我們在實作象棋殘局庫時，使用映射(Mapping)的方法來儲存盤面資訊以減少盤面資訊的範圍。第 4.1 節說明可以使用映射的原因。第 4.2 節說明盤面資訊的映射。第 4.3 節說明本篇論文的實驗環境，第 4.4 節說明實驗結果，第 4.5 節分析以兵卒分割殘局庫的空間成本比較，第 4.6 節分析以兵卒分割殘局庫的時間成本比較。

4.1 不存在的盤面(Nonexistent Position)

在 KRKAAE 中，因為黑方沒有攻擊子力，所以黑方除了用王見王的方式吃掉紅方的帥以外，沒有其他的手段可以吃掉對方的王，而有王見王的盤面是 1 步就到目標的盤面，因此黑先走的盤面不存在 3 步以上能贏的盤面；相對的，紅先走的盤面也不存在 2 步以上會輸的盤面。另一方面，由於紅方有一隻俥，紅方有很多攻擊的手段，在這個象棋殘局庫中，紅先走的盤面中有最大棋步是 17 步的盤面，而黑先走的盤面中有最大棋步是 18 步的盤面。如表 4-1。

前面說過會把象棋殘局庫分割成紅先走和黑先走兩個資料庫，但是在 KRKAAE 中，紅先走的盤面資訊可能值是 0(Draw)、1、3、5、7、9、11、13、15 和 17，不存在值為 2、4、6、8、10、12、14 和 16 的盤面；而黑先走的盤面資訊可能值是 0(Draw)、1、2、4、6、8、10、12、14、16、18，不存在值為 3、5、7、9、11、13、15、17 的盤面。如果直接在資料庫中記錄盤面的距離(Distance)，紅先走殘局庫的每個盤面需要 $\lceil \log_2(17) \rceil = 5$bits 空間，黑先走殘局庫的每個盤面需要 $\lceil \log_2(18) \rceil = 5$bits 空間。但是紅先走的盤面只有 10 個可能值，而黑先走的盤面只有 11 個可能值，理論上兩者每個盤面各最多只需使用 $\lceil \log_2(10) \rceil = 4$bits 空間與 $\lceil \log_2(11) \rceil = 4$bits 空間。

因此，雖然在象棋殘局庫中直接記錄盤面的棋步距離會比較方便處理，但是卻為了一些不存在的盤面，增加了盤面資訊的範圍。

棋步距離	紅先走盤面數	黑先走盤面數
0(Draw)	12067	19921
1	96876	40626
2		11109
3	40401	
4		23349
5	28323	
6		42060
7	8895	
8		22482
9	4527	
10		14865
11	2250	
12		9981
13	1707	
14		8343
15	597	
16		2709
17	66	
18		264
總計	195709*5bits	195709*5bits

表 4-1 KRKAAE 的各棋步距離值的盤面數

4.2 盤面資訊映射(Position Value Mapping)

相對於直接記錄盤面的棋步距離，為了減少盤面資訊的範圍，改成間接記錄盤面的距離：在象棋殘局庫中記錄盤面資訊時，使用一個映射表(Mapping Table)將較大的棋步距離映射到較小的值，並且在象棋殘局庫中紀錄盤面資訊的映射值(Mapping Value)；在查詢最佳棋步時，才再經由一次反查映射表將盤面資訊的映射值轉換回真正的棋步距離。

這種映射法是利用最少的 bit 數來表示盤面資訊，可以分為一對一映射法和多對一映射法，接下來在第 4.2.1 節介紹一對一映射法，在第 4.2.2 節介紹多對一映射法，在第 4.2.3 節介紹多對一映射法的解壓縮。

4.2.1 一對一映射法(1-To-1 Mapping)

此法將盤面資訊的棋步距離值以一對一映射到最小 bit 數所能表示的範圍，如表 4-2(右)的映射表(Mapping Table)將所有盤面資訊轉換成較小的值，在紅先走的映射表中，將 0(Draw)、1、3、5、7、9、11、13、15 和 17 映射到 0~9，在黑先走的映射表中，將 0(Draw)、1、2、4、6、8、10、12、14、16、18 映射到 0~10。因此 KRKAAE 的每個盤面壓縮到只需 4bits 空間，而且不失完整的盤面資訊。

映射值	紅先走盤面數	黑先走盤面數
0(Draw)	12067	19921
1	96876	40626
2	40401	11109
3	28323	23349
4	8895	42060
5	4527	22482
6	2250	14865
7	1707	9981
8	597	8343
9	66	2709
10		264
總計	195709*4bits	195709*4bits

紅先走	黑先走
Draw<=>0	Draw<=>0
1<=>1	1<=>1
2<=>3	2<=>2
3<=>5	3<=>4
4<=>7	4<=>6
5<=>9	5<=>8
6<=>11	6<=>10
7<=>13	7<=>12
8<=>15	8<=>14
9<=>17	9<=>16
	10<=>18

表 4-2 KRKAAE 的各映射值的盤面數與映射表(一對一)

當查詢到盤面的映射值時只要再反查一次映射表就能得回盤面的真實距離，例如我們在 KRKAAE 中查到一個盤面的映射值為 10，再反查一次映射表就知道是 18 步輸的盤面。

4.2.2 多對一映射法(M-To-1 Mapping)

此法和一對一映射法類似，也是將盤面資訊的棋步距離值映射到最小 bit 數所能表示的範圍，但此法將盤面總和較小的棋步距離值區段改成以多對一映射，如表 4-3(右)紅先走映射表將 13、15、17 對映到 7，黑先走映射表 14、16、18 對映到 8。因此 KRKAAE 的紅先走的每個盤面能進一步壓縮到只需 3bits 空間。

映射值	紅先走盤面數	黑先走盤面數
Draw	12067	19921
1	96876	40626
2	40401	11109
3	28323	23349
4	8895	42060
5	9147	22482
6	2250	14865
7	2370	9981
8		11316
總計	195709*3bits	195709*4bits

紅先走	黑先走
Draw<=>0	Draw=>0
1<=>1	1<=>1
2<=>3	2<=>2
3<=>5	3<=>4
4<=>7	4<=>6
5<=>9	5<=>8
6<=>11	6<=>10
7<=>13, 15, 17	7<=>12
	8<=>14,16,18

表 4-3 KRKAAE 的各映射值的盤面數與映射表(多對一)

但是此法會失去完整的盤面資訊，因為反查映射表時有可能會反查到多個值，無法直接由映射表得知盤面原本的真實距離值，於是需要額外花時間去做解壓縮，但是並非每個盤面都需要做解壓縮，因為我們把會反查到多個值的盤面限制在少數的盤面，大部分情況下都還是可以直接反查到真實距離值，這個部分可以做調整；最極致的情況下，多對一映射可以把殘局庫壓縮到只剩 0、1 和 2 共三種距離值，但這就退化到 Win-Loss-Draw 型的盤面資訊。

4.2.3 多對 1 映射的解壓縮(Decompress of M-To-1 Mapping)

原本記錄完整盤面資訊時，找最佳棋步只需展開一層合法步，為了還原多對一映射的壓縮，就需要多對一映射的解壓縮，而多對一映射的解壓縮就是多展開數層合法步，直到找到不同映射值的盤面，如圖 4-4(b)，當找到 12 步敗的盤面時，將它一路往上更新，就能還原盤面原來的棋步距離值，因此就找出最佳棋步。

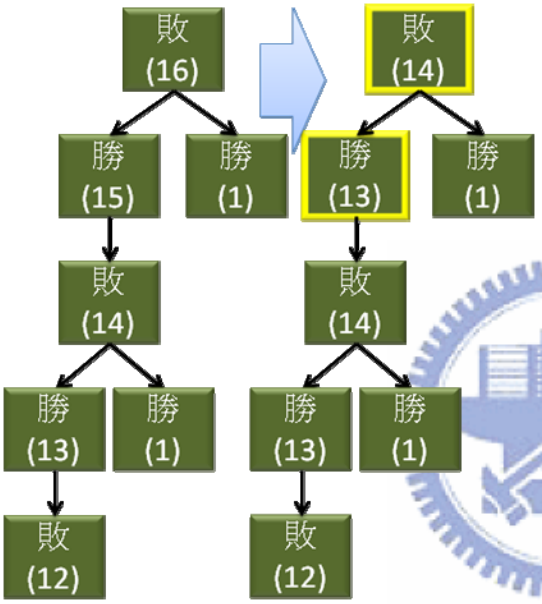

圖 4-4(a)多對一映射法的壓縮例子

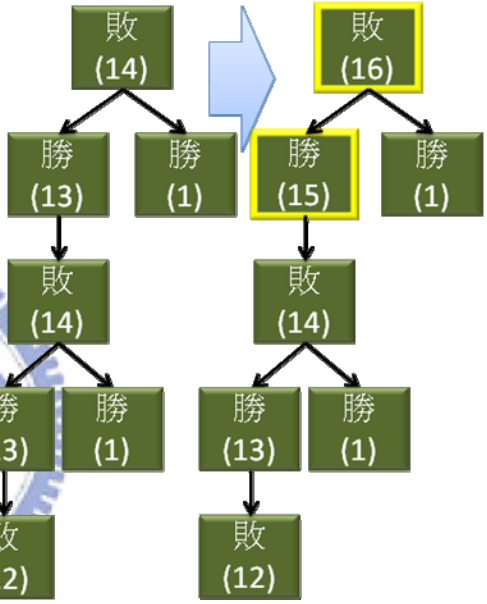

圖 4-4(b)多對一映射法的解壓縮例子

4.3 實驗環境

本篇論文的實驗機器使用 AMD Athlon(tm) 64 X2 Dual Core Processor 4800+(2.5GHz)、2.0GB RAM 和 Seagate Barracuda 7200.10 的硬碟，作業系統使用 Windows Server 2003 Standard Edition Service Pack 2。

4.4 實驗結果

本篇論文實驗以兵卒位置分割殘局庫所使用的回溯分析演算法在找出包含棋規盤面是依照方浩任的的演算法[4][5]再略加修改，並且在建立分割後的殘局庫時，為了得到更客觀的數據，每個互相獨立的進資料庫(Advancing Databases)並未進行平行計算；在建立殘局庫時每個盤面使用 1byte 空間，當 1byte 空間不夠用時，動態使用 2bytes 空間，因此在產生殘局庫前會以 byte 為單位對殘局庫大小進行評估，如果殘局庫太大就使用 External-Memory 回溯分析演算法[14]，否則就用一般的回溯分析演算法；在建立完殘局庫後，使用一對一映射法以 bit 為單位對殘局庫進行壓縮。

本論文的實驗建立了 40 個包含 1 隻兵(卒)的殘局庫，30 個包含 2 隻兵(卒)的殘局庫，6 個包含 3 隻兵(卒)的殘局庫，總共 76 個殘局庫。

表 4-5 即為本論文的實驗數據，以兵卒數量(由少至多)及殘局庫大小(由小至大)為順序，為未以兵卒位置分割的殘局庫與分割後的殘局庫的建立時間與使用空間之比較；而分割後殘局庫的數據都是每個進資料庫的總和，例如 KRKHP 的建立時間是 KRKHP1 、KRKHP2、... 和 KRKHP7 的建立時間的總和，使用空間也是一樣，層級數為未分割殘局庫之最大層級。

以下空間比是指已分割的殘局庫的使用空間除以未分割的殘局庫的使用空間，時間比則是已分割的殘局庫的建立時間除以未分割的殘局庫的建立時間。

殘局庫	層級數	空間(bit)（未分割）	空間(bit)（已分割）	時間(sec)（未分割）	時間(sec)（已分割）
kcckp	1	91104750	88123140	470	376
krckp	0	160380000	136687500	319	319
krpkc	0	160380000	147258000	361	369
krkcp	2	200475000	176782500	655	442
krpkh	1	200475000	174231000	445	432
khckp	1	200475000	183343500	447	361
khpkc	1	200475000	193914000	614	466
kcpkc	0	200475000	167670000	408	363
krkhp	3	240570000	208129500	906	611
khpkh	4	280665000	267543000	1467	1057
kcpkh	6	280665000	199746000	2256	1178
krpkr	18	320760000	295609500	3804	2945
kcckap	1	416319750	398910015	9791	1975
kcckep	1	634392000	613630080	16102	3061
kahpkc	1	841995000	814438800	9722	2272
krpkah	2	841995000	841995000	5701	2354
krpkac	1	841995000	786882600	5774	2535
kcpkah	1	841995000	730239300	21117	2059
kcpkac	1	841995000	841995000	25977	2227
kacpkc	4	1010394000	872613000	27294	3269
krkahp	3	1010394000	944565300	9372	3171
krpkec	1	1026432000	1026432000	8006	2935
kcpkec	0	1026432000	951782400	21610	2557
kahpkh	3	1178793000	1108371600	14463	4390
khpkac	3	1178793000	1068568200	30706	5132
kehpkc	1	1283040000	1241049600	15298	3422
krpkeh	1	1283040000	1178064000	6723	3159
karpkr	15	1347192000	1305857700	54926	13202
kacpkh	6	1347192000	1233905400	50128	9455
krpkar	18	1347192000	1311981300	116080	17904
khpkah	8	1347192000	1236967200	66337	9053
kcpkeh	2	1411344000	1103414400	48612	4078
kecpkc	2	1539648000	1201392000	31459	3520
krkehp	3	1539648000	1474329600	16463	5674

kehpkh	3	1796256000	1497657600	22559	6728
khpkeh	4	1796256000	1721606400	76568	9161
khpkec	3	1796256000	1565308800	53541	6297
kerpkr	15	2052864000	2052864000	92306	23619
kecpkh	10	2309472000	1847577600	201910	18838
krpker	24	2309472000	2106518400	220340	34746
krkpp	2	59940000	48174750	324	191
kckpp	1	71928000	50463000	359	209
khkpp	7	95904000	73722150	1383	656
kcppka	0	201398400	173536020	313	319
krppke	0	230169600	213036480	920	464
karkpp	2	251748000	209171970	1410	887
krkapp	3	276922800	210634830	2029	933
kahkpp	3	302097600	251526870	2915	1388
kackpp	2	302097600	248703210	2045	911
kckapp	1	302097600	217421820	8201	1096
khppke	0	306892800	306892800	818	439
kcppke	0	383616000	311558400	969	520
khkapp	8	402796800	306350100	28657	3358
karpkp	1	411642000	359642430	903	761
krpkap	0	411642000	355423950	794	718
krkepp	3	421977600	318271680	7660	1447
keckpp	2	460339200	292429440	13764	1568
kckepp	1	460339200	336934080	18238	1533
kacpkp	1	514552500	454915440	1249	853
kehkpp	3	537062400	391858560	15503	2159
khkepp	9	613785600	488229120	55333	5627
kahpkp	1	617463000	492116310	1251	792
khpkap	2	617463000	500008950	6635	872
kcpkap	0	617463000	437293080	4656	832
kerpkp	0	627264000	543360960	2044	1117
krpkep	0	627264000	538176960	2624	1133
kehpkp	1	784080000	734676480	3849	1182
kecpkp	0	784080000	674904960	3589	1224
khpkep	0	940896000	762359040	5992	1299

kcpkep	0	940896000	640094400	9122	1310
kappkp	0	153846000	125901216	529	368
kppkap	6	215384400	131530581	1815	422
keppkp	0	234432000	187734528	797	568
kppkep	0	281318400	193688640	2562	624
krpkpp	3	3956040000	2799700200	120278	11953
kcpkpp	0	4615380000	3642626700	123206	18616
Total		61767108000	53341025040	1743778	284061

表 4-5 殘局庫的建立時間與使用空間的實驗數據

這 76 個殘局庫的總空間比為 86.36%，總時間比為 16.29%，亦即分割後的殘局庫節省了約 7 分之 1 的空間，產生時間則約為原本的 6 分之 1。

圖 4-6 與圖 4-7 為 76 個殘局庫的個別空間比與時間比，順序由左至右與表 4-5 由上至下相同。根據圖 4-6，我們發現當殘局庫的兵卒數量愈多時，分割後的殘局庫愈省空間；根據圖 4-7，我們發現當殘局庫的大小愈大時，分割後的殘局庫愈省時間。以下將就空間與時間這兩方面做各別分析。

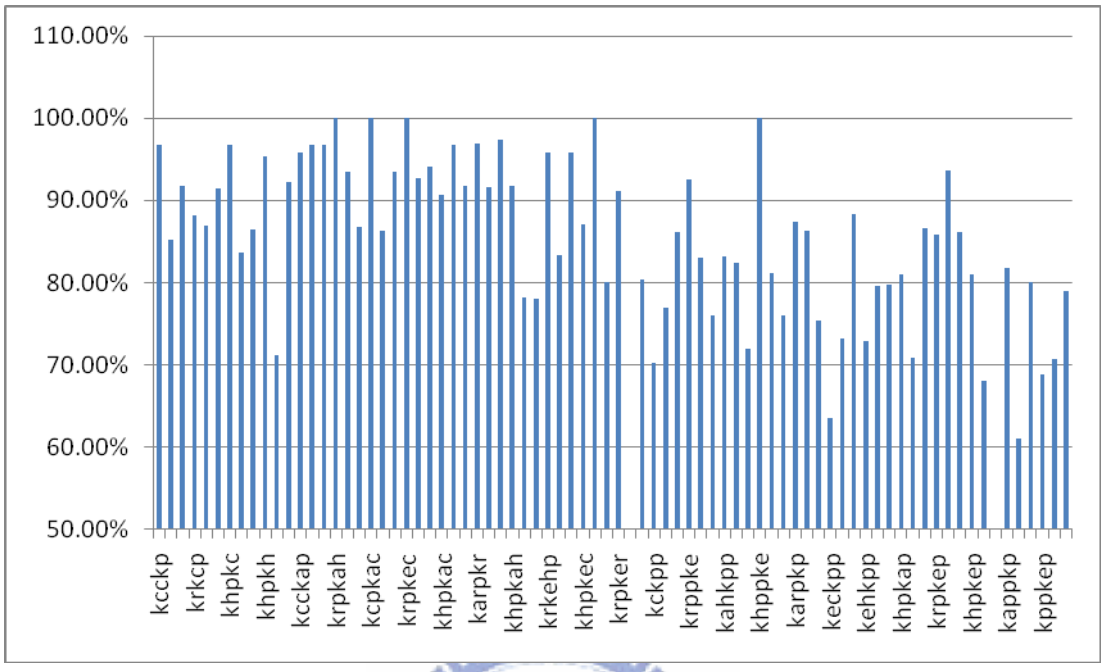

圖 4-6 76 個殘局庫的個別空間比

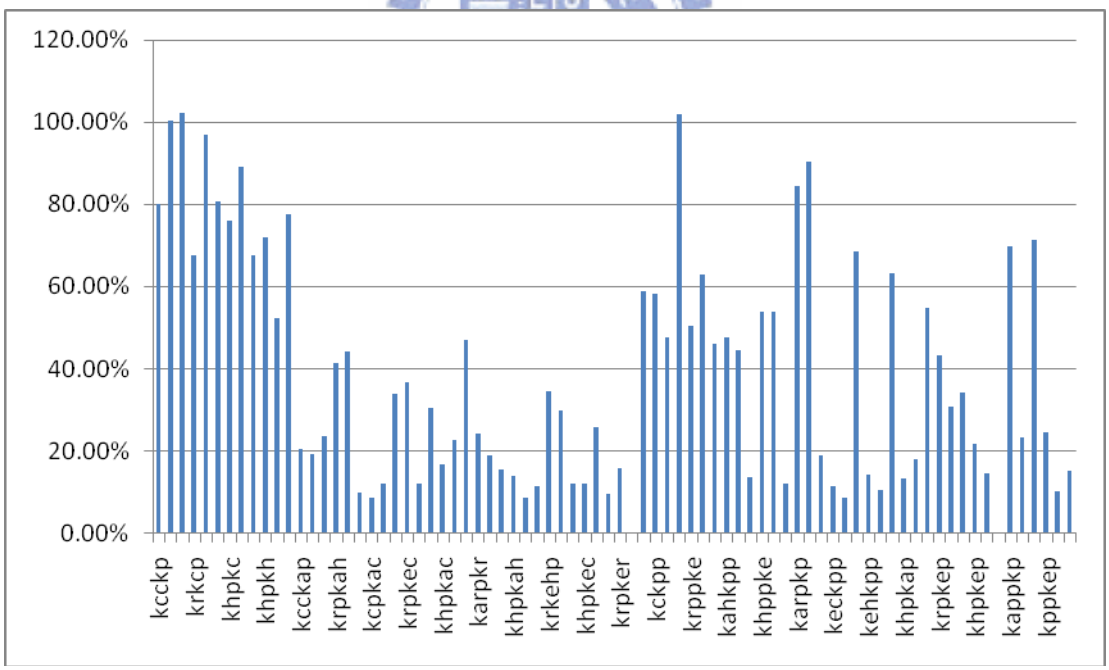

圖 4-7 76 個殘局庫的個別時間比

4.5 空間成本分析

根據圖 4-8(深色為紅先走殘局庫，淺色為黑先走殘局庫)，我們發現當兵卒在第 1、2 列時所使用的 bit 數通常比在第 6、7 列時所使用的 bit 數少，而第 6、7 列時所使用的 bit 數通常比在第 3、4、5 列時所使用的 bit 數少，我們在其他的殘局庫中也發現到類似的情形；這是因為第 1、2 列的兵卒機動性較差，且容易死亡，爭勝負的情況就少，因此最大步數就相對較少；第 6、7 列的兵卒則是因為只能前進不能左右移動且離九宮太遠，因此對王的威脅不大，但還有過河爭勝負的機會，所以最大步數比 1、2 列的兵卒稍大；當兵卒在第 3、4、5 列時，機動性最大，又接近九宮，對王的威脅也大，所以這種殘局庫的最大步數相對比其他幾列的都大。

當兵卒在不同列時普遍會令最大步數產生不同，因此很少殘局庫在分割後的使用空間跟分割前一樣，如圖 4-8，原本未分割的 KHKPP 的每個盤面皆須使用 8bits 空間，但分割後有些盤面只需 2bits 空間，而分割後的 KHKPP 的總合使用空間為原來的 77%。

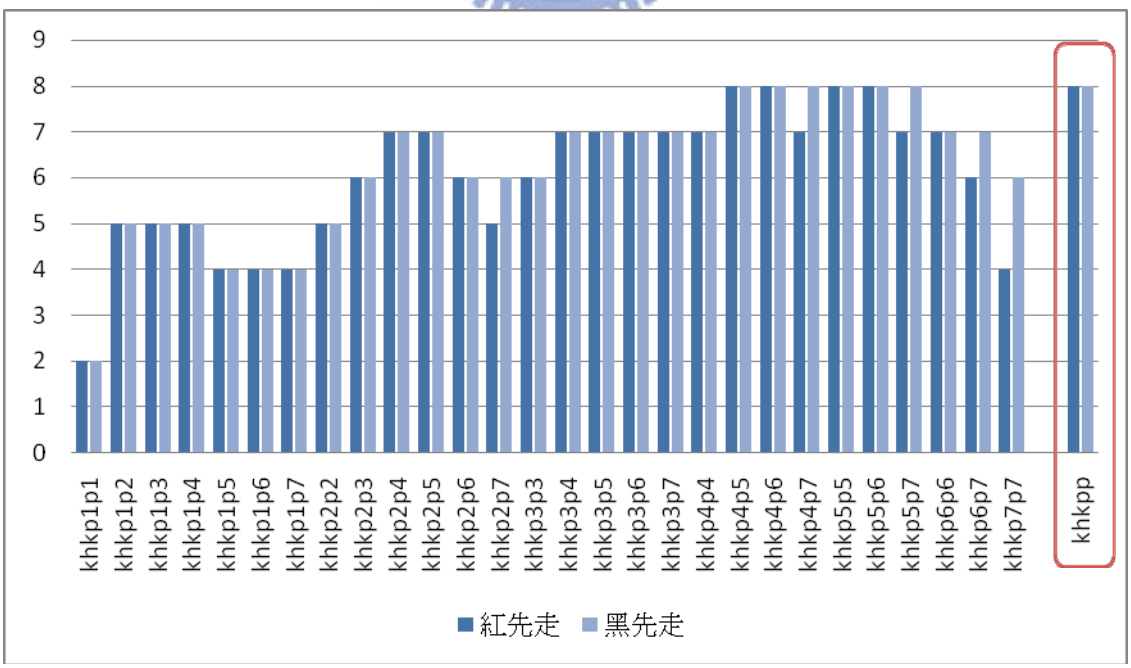

圖 4-8 KHKPP 被分割後的每個殘局庫使用的 bit 數

4.6 時間成本分析

根據圖 4-9，我們發現層數較高的盤面大多是集中在兵卒在 3、4、5 列的殘局庫，理由同第 5.3 節的兵卒機動性；方浩任的演算法為了找犯棋規的重覆盤面時，會尋訪同一個殘局庫中的每個仍是未知狀態的盤面，並對其展開子盤面或父盤面，但有些計算量是可以避免的，例如在 KHP1P1 中，當在第 3 層發現已無犯棋歸盤面時，便不會繼續找有無第 4 層犯棋規的盤面，但是未分割 KHKPP 時仍會去尋訪這些盤面，而分割後的 KHKPP 後的總合建立時間為原來的 47%。

另外，有些殘局庫未分割時無法全部載入記憶體中處理，在分割之後有些可以全部載入記憶體，大幅減少 I/O 的次數。

因此，時間成本方面，節省了計算量與 I/O 花費的時間，而且若互相獨立的進資料庫平行計算將會更省時間。

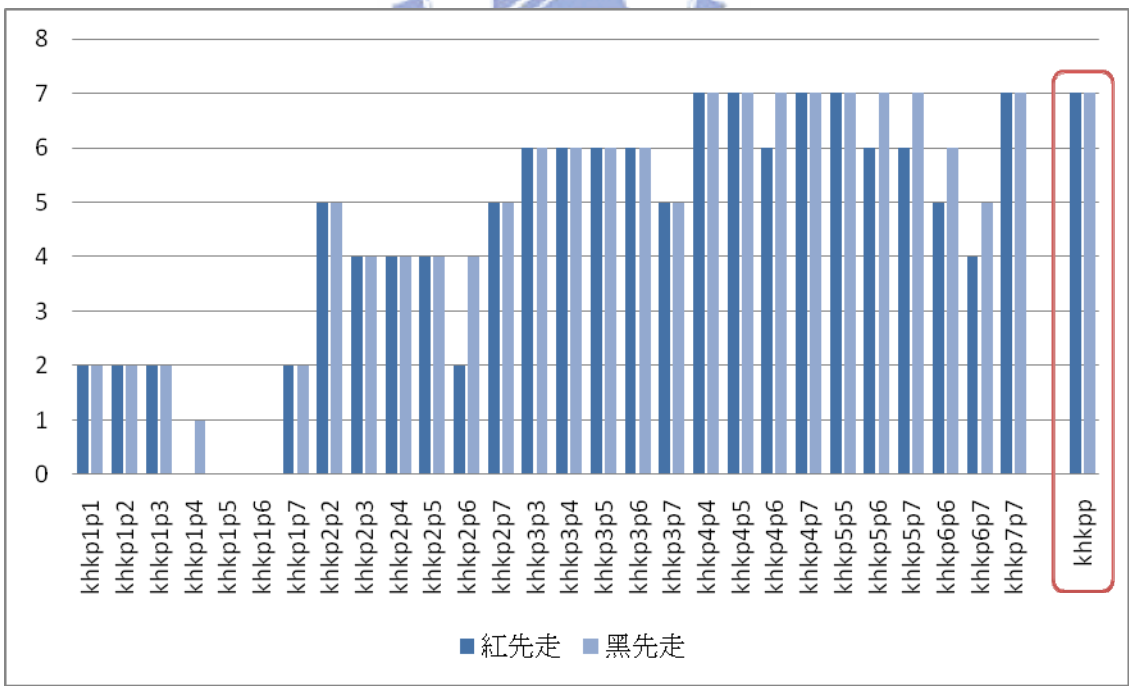

圖 4-9 KHKPP 被分割後的每個殘局庫的最大層級

第五章、結論與未來展望

本篇論文的主要貢獻是利用象棋兵卒走法的特性進一步分割象棋殘局庫，在有限的儲存空間內建立更多的象棋殘局庫，並且降低建立時間的花費，並且以映射法更進一步壓縮象棋殘局庫。另外，在產生其他棋類遊戲的殘局庫時，若有類似象棋的兵卒前進這種不可逆的走法時，也可以用來做殘局庫的分割。

對於違反棋規的盤面，盤面資訊仍有許多不完整，在未來的發展上，應包含更精確的盤面資訊，以及計算此資訊的演算法：

1. 在循環盤面中，最佳棋步的走法，對於勝方應選擇最小循環路徑令循環棋步愈少愈好，對於敗方應選擇最大循環路徑令循環棋步愈多愈好。
2. 重覆盤面的敗方在脫離循環盤面時的最佳走法，一般人類棋手為了可以多抵抗幾步棋，會選擇在盤面循環三次後脫離循環盤面，而最小的循環盤面是四步循環，所以因犯棋規而輸的盤面至少可以從 2 個以上的盤面可以脫離循環，從不同的盤面脫離循環可抵抗的步數也不同，這時應選擇可抵抗最久的盤面脫離。

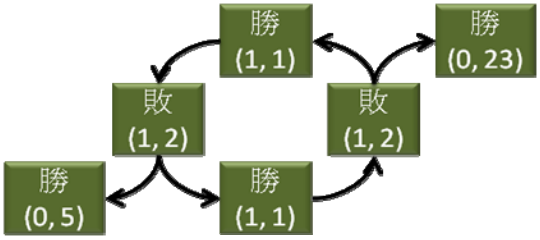

圖 6-1 四步循環的盤面

以象棋殘局庫作為研究背景，我們期望能夠藉由產生大量的象棋殘局庫來改善象棋 AI 程式在殘局階段的棋力。我們也希望在其他棋類遊戲上能推廣運用回溯分析演算法，並對人工智慧的研究上有相當大的幫助。

參考文獻

[1] Haw-ren Fang, Tsan-sheng Hsu, and Shun-chin Hsu (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.

[2] Haw-ren Fang, Tsan-sheng Hsu, and Shun-chin Hsu (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. Müller). Springer-Verlag, New York, N.Y.

[3] Haw-ren Fang (2004). Rule-Tolerant Verification Algorithms for Completeness of Chinese-Chess Endgame Databases. Computers and Games 2004: 129-144

[4] Haw-ren Fang (2005). The Nature of Retrograde Analysis for Chinese Chess – Part 1. ICGA J 28 (2): 91-105 SEP 2005.

[5] Haw-ren Fang (2005). The Nature of Retrograde Analysis for Chinese Chess – Part 2. ICGA J 28 (3): 140-152 SEP 2005.

[6] E. A. Heinz (1999). Knowledgeable encoding and querying of endgame databases. ICCA Journal, 22(2):81–97, 1999.

[7] Tsan-sheng Hsu and Ping-Yi Liu (2002). Verification of endgame databases ICGA J 25 (3): 132-144 SEP 2002.

[8] P. Karrer. KQQKQP and KQPKQP? (2000). ICCA Journal, 23(2):75–84, 2000.

[9] Thompson, K. (1986). Retrograde Analysis of Certain Endgames. ICCA Journal, Vol. 9, No. 3. 131-139.

[10] Thompson, K. (1996). 6-Piece Endgames. ICCA Journal, Vol. 19, No. 4, pp. 215–226.

[11] T. R. Lincke and A. Marzetta (1999). Large endgame databases with limited memory space. ICCA Journal, 22(4):131–138, 1999.

[12] Jih-tung Pai (2007). Chinese Chess Endgame Databases Query System, http://lpforth.forthfreak.net/endgame.html.

[13] Ren Wu and Donald F. Beal (2002). A Memory Efficient Retrograde Algorithm and Its Application To Chinese Chess Endgames More Games of No Chance MSRI Publications Volume 42 , 2002.

[14] Ping-hsun Wu, Ping-Yi Liu, Tsan-sheng Hsu (2004). An External-Memory Retrograde Analysis Algorithm. Computers and Games 2004: 145-160

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

[16] 方浩任 (1999). 象棋殘局回溯分析演算法之實作與評估, 八十八年全國計算機會議論文集, 淡水, pp. A367-372

[17] 亞洲象棋聯合會裁判組 (2003). 象棋比賽規例, 亞洲象棋聯合會.

[18] 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.

[19] 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.

[20] 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.

[21] 許舜欽, 曹國明 (1991). 電腦象棋開局知識庫系統之研製, 台灣大學工程學刊, 第五十三期, 頁75—86

[22] 象棋百科全书 (2004). 计算机博弈 - 象棋百科全书, http://www.elephantbase.net/computer.htm.