國立臺灣師範大學
資訊工程研究所碩士論文

指導教授：　林順喜　博士

台灣直棋的勝負問題之研究

The Study of the Taiwan Morris games

研究生：　　李明臻　　撰

中華民國　一百　年　一　月

摘 要

台灣直棋的勝負問題之研究

李明臻

電腦棋類遊戲在人工智慧領域中是很重要的，各種棋類遊戲研究層出不窮。直棋(Morris)遊戲屬於雙人遊戲的一種，自從 Ralph Gasser 教授在 1996 年提出破解 Nine Men's Morris 的結果之後，有關 Morris games 更高一層或其它版本的研究，相關文獻就十分少見。

台灣規則的直棋遊戲是 Twelve Men's Morris 在台灣的變體。在本論文中，我們使用 CPU 規格為 Intel Xeon E5520 2.27GHz(雙處理器)，記憶體總量為 36 GByte 的機器，證明了台灣規則的直棋遊戲在開始雙方各拿六子及九子的情況，其結果都為和棋。

我們除了找到台灣規則的直棋遊戲在開始雙方各拿六顆棋子及九顆棋子的勝負結果是和棋，更從破解 Nine Men's Morris 的方法中，在放子階段提出跟原本作法不同的破解方法。在研究台灣規則的直棋遊戲的過程中，找到了將資料庫分割得更細的方法以及加速旋轉對稱運算的方法，並且將其運用在資料庫技術及回溯分析演算法上。

關鍵字：電腦直棋、回溯分析演算法、人工智慧

ABSTRACT

The Study of the Taiwan Morris games

by

Li, Ming-Zhen

Computer chess games are very important in the field of artificial intelligence. There are very few research results on Morris in higher dimensions or in other variations since Professor Ralph Gasser solved Nine Men's Morris in 1996.

The board of Taiwan Morris game is same as the board of Twelve Men's Morris. In this thesis, a personal computer equipped with AMD Athlon64 X2 4000+ 2.1GHz CPU and 36 GBytes RAM is utilized to conduct our experiments. Thus, it gets the results of Taiwan Morris game that each player starts with six or nine pieces are a first-player draw.

In addition, we find some skills for improving the performance of our programs. We used retrograde analysis algorithm and databases in the opening stage of Taiwan Morris games. We also describe some methods employed to accelerate the calculation, such as rotation and symmetry permutation.

誌　謝

感謝指導教授林順喜博士，指導本論文之研究內容及寫作方向，並且教導許多演算法與人工智慧領域相關的知識。

還要感謝在實驗室裡的其它成員給予的幫助，感謝潘典台學長、魏仲良學長、黃士傑學長、黃立德學長、莊臺寶學長、趙義雄學長、劉雲青學長、白聖群學長、葉俊廷學長和黃信翰學長，感謝詹傑淳同學、賴昱臣同學、謝政孝同學、陳俊佑同學、李啟峰同學和陳冠明同學，感謝勞永祥學弟、陳志宏學弟、蔡宗賢學弟、唐心皓學妹、林立學弟、施宣丞學弟和李萬和學弟。

特別感謝林順喜教授，以及潘典台學長、黃士傑學長跟勞永祥學弟在實作程式時給予我良好的建議及方向。

最後感謝我的家人包容我的一切失敗，盡心盡力地支援我的一切需要，讓我無所顧忌的投入研究之中，直到我順利完成學業，僅將此論文獻給我敬愛的父母。

目 錄

摘 要 ..... **i**

ABSTRACT ..... **ii**

誌 謝 ..... **iii**

附表目錄 ..... **vi**

附圖目錄 ..... **vii**

第一章、 緒論 **1**

第一節、 簡介 *1*

第二節、 研究動機 *2*

第三節、 論文組織 *3*

第二章、 相關研究探討 **4**

第一節、 台灣直棋 *4*

第二節、 直棋(Morris)相關研究成果 *8*

第三節、 回溯分析演算法 *12*

第四節、 殘局庫 *15*

第三章、 如何破解台灣直棋 **18**

第一節、 回溯分析演算法的改良 *18*

第二節、 殘局資料庫的改良 *22*

第三節、 放子階段的作法 *26*

第四節、 Zobrist hashing 演算法 *29*

第四章、 其它加速方法 **31**

第一節、 旋轉及對稱 *31*

第二節、 回溯分析演算法運用旋轉及對稱 *35*

第三節、 加速相似運算 *37*

第五章、 實驗成果 **42**

第一節、 實驗設計 *42*

第二節、　六子台灣直棋 42

第三節、　九子台灣直棋 46

第六章、　結論與未來研究方向 49

第一節、　結論 49

第二節、　未來研究方向 49

參考著作 50

附表目錄

表一　旋轉及對稱數學表示式 33

表二　有相似計算跟沒相似計算的比較 40

表三　六子台灣直棋後手勝利的其中一條路徑 43

表四　九子台灣直棋和棋的其中一條部分路徑 46

附圖目錄

圖 1-1 Twelve Men's Morris 盤面。 1

圖 1-2 「直」的圖示。 2

圖 2-1 Twelve Men's Morris 盤面。 4

圖 2-2 「直」的圖示，當黑方產生此情況時，可以拿掉盤面上任一白方棋子。 5

圖 2-3 「橫」的圖示，黑方玩家將棋子移到圓圈的位置可以吃掉中間的棋子。 6

圖 2-4 「擔」的圖示，白方玩家將棋子移到圓圈的位置 6

圖 2-5 直棋遊戲最短的循環路徑 7

圖 2-6 儲存盤面勝負結果示意圖 10

圖 2-7 放子階段搜尋樹示意圖 12

圖 2-8 輸盤面回溯的示意圖 13

圖 2-9 贏盤面回溯示意圖 14

圖 2-10 Ralph Gasser 的資料庫分類 16

圖 2-11 A 跟 B 的盤面勝負結果其實是相同 17

圖 3-1 吃子步跟走子步示意圖 19

圖 3-2 程式的流程圖 20

圖 3-3 回溯程序時判定輸贏示意圖 21

圖 3-4 於讀取資料庫時設定盤面勝負示意圖 22

圖 3-5 台灣直棋殘局庫部分關係圖 23

圖 3-6 將交叉點塗上黑白兩色 24

圖 3-7 分割方法示意圖 25

圖 3-8 放子階段回溯分析演算法示意圖 27

圖 3-9 放子階段的資料庫部分關係圖 28

圖 3-10 同個 hash table 在程式運用示意圖 29

圖 4-1 相似盤面的例子 31

圖 4-2 16 個相似盤面 32

圖 4-3 相似盤面數學表示法示意圖 33

圖 4-4 計算分支度錯誤原因示意圖 36

圖 4-5 相似盤面於回溯程序的問題示意圖 37

圖 4-6 去掉中間正方形不會改變相似運算的例子 38

圖 4-7 去掉中間正方形影響到相似運算 39

圖 5-1 每人六個棋子的台灣直棋遊戲初始選擇 43

圖 5-2 九子台灣直棋遊戲初始選擇 46

第一章、 緒論

第一節、 簡介

直棋（Morris），是起源於西元前一千四百年的古埃及，相傳是羅馬經由貿易將直棋遊戲流傳於各國。直棋遊戲為一個雙人遊戲，通常棋盤為同心的數個正方形，並用直線或斜線將不同的正方形相連結，如圖 1-1。

直棋(Morris)遊戲廣泛流傳於世界各地，因而發展出各種不同規則的變形遊戲，流傳於華人地區的是 Twelve Men's Morris 的變體。在各地有不同名稱，例如：閩台稱為直棋、放直棋、行直棋，閩南語俗諺『仙棋乞丐直』，意思是下棋是有錢有閒人的娛樂，沒錢的窮苦民眾只能以簡單的直棋自娛。北京稱為連兒棋；四川稱為三棋；湘西土家族稱為打三棋，其他地區還有花窗棋、刪棋、三子棋等稱呼。

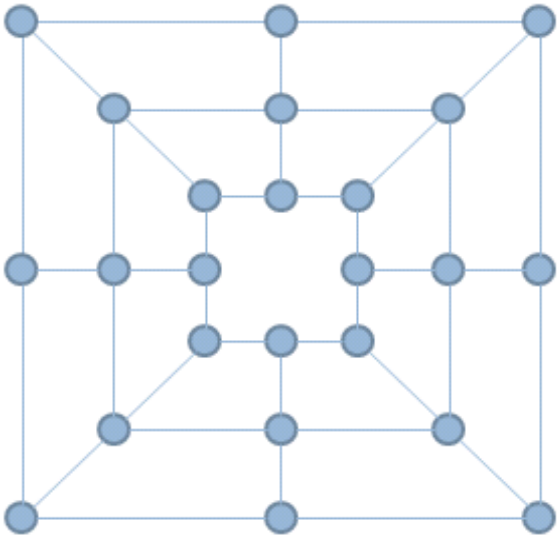


圖1-1　Twelve Men's Morris 盤面。

各地直棋遊戲變體都依循的共通遊戲規則為開始將手中的棋子以雙方輪流的順序放入棋盤上。當手中的棋子都已經放完後，才可以移動棋子。因此直棋遊戲至少擁有兩個不同階段。如果造成三個己方棋子連成一線，俗稱此情況為「直」，就可以拿掉棋盤上對方的一顆棋子，並且拿掉的棋子不能再放回棋盤上，本篇論文將拿掉棋盤上對方棋子的行為稱為「吃子」。

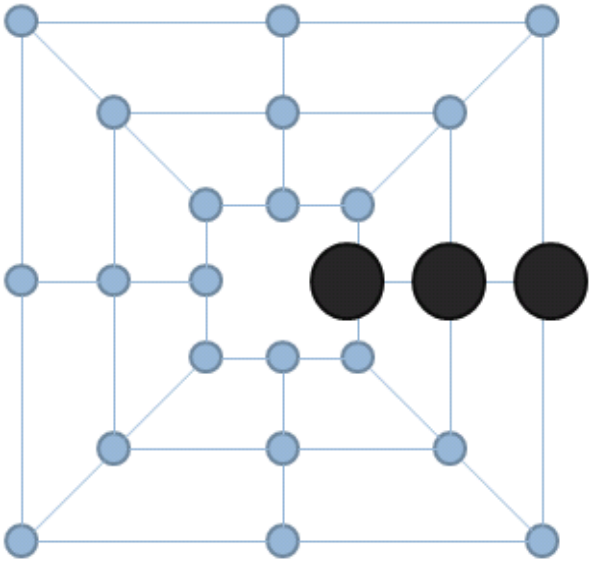


圖1-2 「直」的圖示。

第二節、 研究動機

Ralph Gasser [1][2] 利用回溯分析法及建立殘局庫的方式，證明了 Nine Men's Morris 的結果為和棋，而台灣直棋流行於台灣地區已經很久，卻一直都鮮有人對其研究。因此我們參考 Ralph Gasser 的作法加以改良為適合台灣直棋的演算法，進而證明台灣直棋的結果。

由於台灣直棋的狀態空間複雜度是 $O(3^{24})\approx O(2^{39})$，如何運用現階段的電腦解決台灣直棋的勝負問題，就變成一種有挑戰性的問題。在解決此類問題時，需要

使用到輔助記憶體來幫助我們解決，而我們在探索及嘗試各種演算法增進破解台灣直棋程式的速度時，所發現的方法，也期望能運用在其它相似的棋類遊戲破解之上，從而在電腦博奕領域中貢獻綿薄之力。

第三節、 論文組織

本論文共分為六章。第一章為緒論，簡述前言、研究動機及論文架構。第二章介紹什麼是台灣直棋及先前跟直棋相關的研究，並說明其基礎理論及如何應用。第三章介紹一些破解台灣直棋所使用到的演算法及其設計理念。第四章介紹嘗試加速破解台灣直棋程式的方法。第五章為我們的研究成果。第六章為結論及未來研究方向。

第二章、 相關研究探討

第一節、 台灣直棋

各種不同地區的直棋(Morris)遊戲的規則都存在著些許的不同，在此篇論文主要研究的問題是直棋的台灣規則，於此篇論文稱之為台灣直棋。台灣直棋是 Twelve Men's Morris 的變體之一，依照 Twelve Men's Morris 的規則雙方各有 12 顆棋子，其棋盤是由 24 個點組成的三同心的正方形，直斜線交叉，棋盤上有穿越正方形的線有八條，沿正方形邊而成的線有十二條。

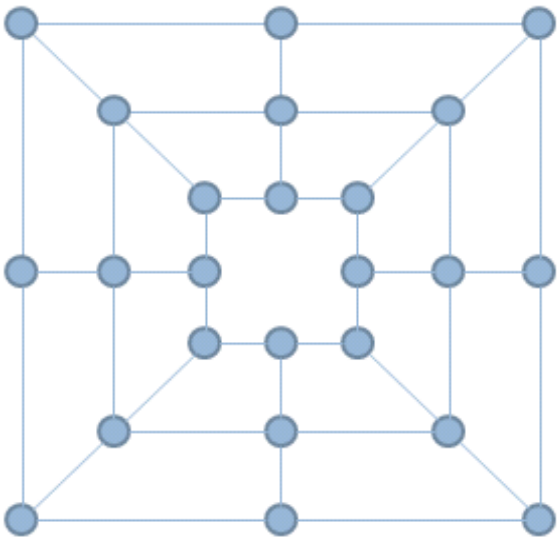


圖2-1 Twelve Men's Morris 盤面。

Twelve Men's Morris 遊戲規則為開始雙方手中各擁有 12 顆棋子，輪流將棋子放置於棋盤之上，當雙方手中已經無棋子時，就開始輪流移動棋盤上的棋子。Twelve Men's Morris 在遊戲過程中只有一種「吃子」情況，即為「直」。當在放置棋子的階段發生「直」時，限制不能吃掉三顆棋子連成一線中的任何一顆棋子；

當在移動棋子的階段發生「直」時，則無此限制。遊戲進行至某一方無棋子可移動或是盤面上的棋子少於兩顆，則該方判定為輸。

台灣直棋遊戲規則特別之處在於遊戲進行中發生吃子的情況，除了「直」之外還有「擔」和「槓」，並且於遊戲過程中，同時滿足多種可以吃子的情況時，只能選擇一種吃子情況來吃子。

「直」: 當己方主動有三顆棋子連成一線，吃掉對方任何一枚棋子。如圖 2-2。

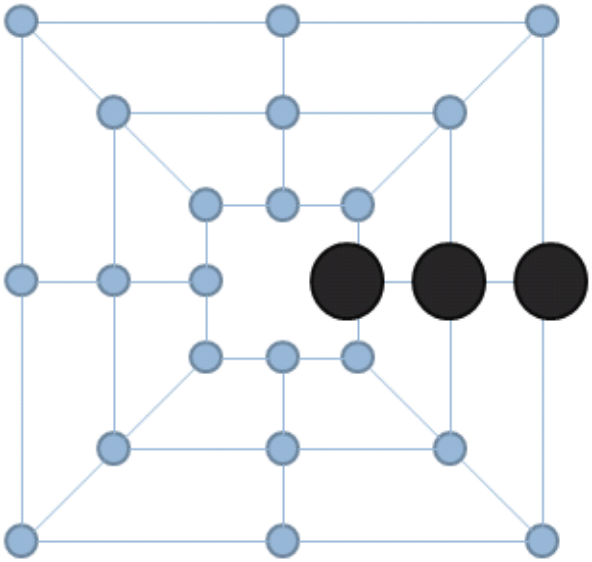


圖2-2　「直」的圖示，當黑方產生此情況時，可以拿掉盤面上任一白方棋子。

「槓」：當己方主動在穿越矩形的線上有兩枚棋子夾住一枚敵棋，吃掉對方該枚棋子。「槓」只能限定在八條穿越正方形的放射線上，而不能發生在圍成正方形的線段上。如圖 2-3。

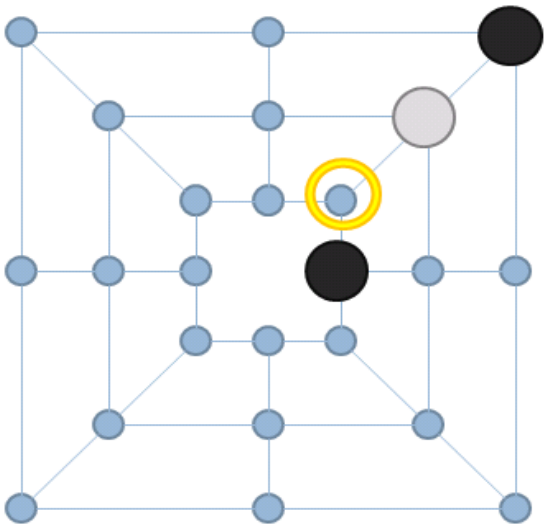


圖2-3　「槓」的圖示，黑方玩家將棋子移到圓圈的位置可以吃掉中間的棋子。

「擔」：當己方主動在沿著正方形的線段上有一枚棋子至兩枚敵棋的中間，吃掉敵方該兩枚棋子。「擔」只能限定在圍成正方形的線段上。如圖 2-4。

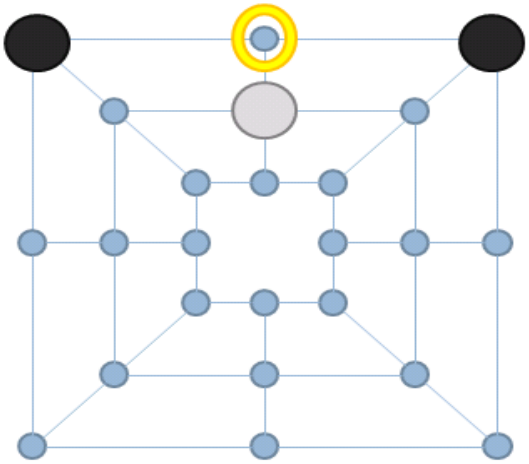


圖2-4　「擔」的圖示，白方玩家將棋子移到圓圈的位置
可以吃掉左右兩邊的黑棋。

台灣直棋遊戲一開始的階段於此篇論文稱為放子階段，放子階段雙方輪流放置手上的棋子於棋盤之上。在放子階段時，將棋子放置於棋盤上線段的交叉點上，但不可放置在已經有棋子的點上，已放置的棋子在此階段是不可移動的。因

此在此階段等同於為接下來的行子階段先佈局，此階段也可以產生三種「吃子」的情況，而且在此階段不會產生結束盤面。

當手中無棋子時，開始移動盤面上的棋子，於此篇論文稱為行子階段，於行子階段時不會再增加盤面上雙方的棋子數，雙方開始利用移動盤面上的棋子來產生「吃子」的情況，直到某方棋子數量少於三個或是某方沒有可以移動的棋子時，就判定為輸棋，此時就結束遊戲。移動棋子方式限定只能將棋子沿著線段往臨近的交叉點上移動，當兩個交叉點間沒有線段連接的狀況，則棋子不能由兩點中一點往另一點移動。

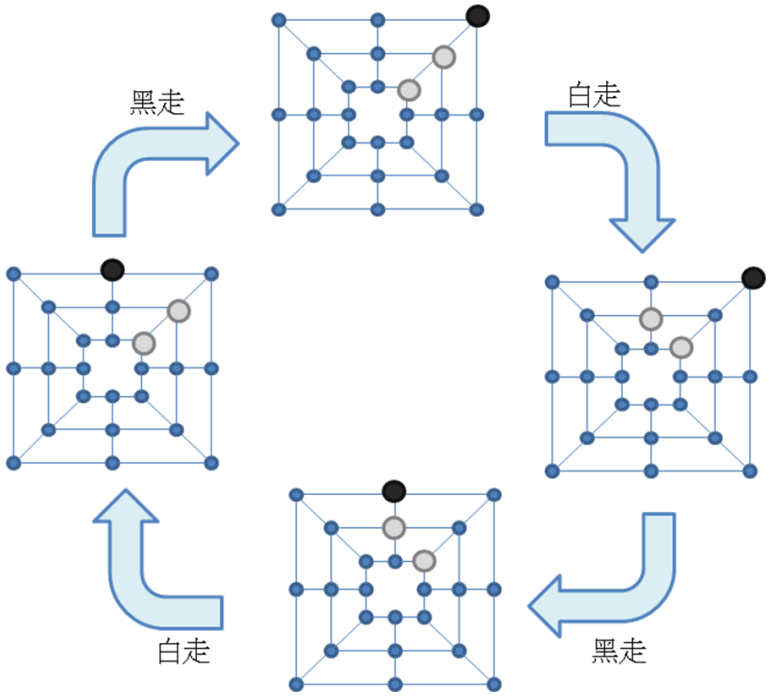


圖2-5　直棋遊戲最短的循環路徑

因為雙方輪流移動棋子，所以有機會發生走過的盤面又再一次出現，造成循環的盤面，如圖 2-5 所示。在雙方都不改變行子方式的情形下有可能產生和棋的結果。

自從 Ralph Gasser 在 1996 年提出破解 Nine Men's Morris 的結果之後，一直都沒有人對於台灣直棋研究其勝負結果，本論文參考 Nine Men's Morris 的作法，並且加以改良，得到六子及九子的台灣直棋結果為和棋。

第二節、　直棋(Morris)相關研究成果

民國七十一年，許舜欽教授於國立台灣大學工程學刊上發表了"直棋的電腦解法及其實現"論文[7]，其中所指的「直棋」為台灣本土所流行的台灣直棋。於論文中描述利用電腦來下台灣直棋所採用的資料結構，以及演算法、思考策略。其中強調一個強而有力的估值函數。

估值函數是對盤面上的棋子分佈加以估值。當棋子將被置於盤面上某一點時，盤面上原有各棋子都受其影響而發生作用，依其遠近距離之不同，這些作用可以分成三類：

(1) 直接作用：與置放點同一直線上的棋子發生直接作用。

(2) 間接作用：與直接作用點同一直線上的棋子發間接作用。

(3) 微弱作用：盤面上的其它各位置上的棋子，對於目前放置的棋子，僅有微弱作用。

對於各個作用的支線上也有各種不同的情況出現。將這些不同情況加以整理，得出 15 個具體的等級：

1. 提取對方兩子。
2. 解除對方提取兩子的威脅。
3. 提取對方一子。

4. 解除對方提一子的威脅。
5. 強烈威脅提子(對方無法防守)。
6. 威脅對方兩子。
7. 威脅提子。
8. 阻止對方發展。
9. 使我方具有發展性。
10. 稍有利於我方。
11. 平分秋色。
12. 稍不利於我方。
13. 對方有發展性。
14. 受對方提一子的威脅。
15. 受對方提兩子的威脅。

把直接作用支線與間接作用支線依其不同情況，加總統計其利弊得失，藉此可找出較最佳著手。

實作程式的觀察力約略可算是和一般人不相上下了，但是由於全盤周詳的考慮使程式顯得比一般人還細心，因此程式的得勝率高達八成。但是其程式尚未能產生最佳著手。

西元 1996 年，Ralph Gasser 教授發表了"Solving Nine Men's Morris"論文[2]，於此論文中，運用 retrograde analysis、endgame databases 跟 alpha-beta search 的方法解決 Nine Men's Morris 的問題。求解 Nine Men's Morris 過程中依照 Morris 的遊戲特性分成兩階段求解，在放子階段使用 alpha-beta search 演算法，在行子階段使用 retrograde analysis 演算法及 endgame databases 的技術。

在放子階段不會出現循環盤面並且使用樹搜尋的深度固定，但因為電腦硬體的限制，Ralph Gasser 針對 Nine Men's Morris 問題提出了減少程式記憶體使用量的方法。

Nine Men's Morris 遊戲的放子階段共有十八回合，即雙方手中各九顆棋子並且輪流放置棋子到盤面上，去除不可能產生的盤面，將所有在行子階段經由回溯分析演算法求得的結果讀入記憶體裡還需要 9 GBytes 的記憶體空間。

首先改善資料庫在記憶體儲存方式以縮小記憶體使用空間，於資料庫裡儲存的資訊包括盤面勝負值、至結束盤面所需最少回合數等，但在放子階段並不需要計算至結束盤面所需最少回合數，因此在將資料庫讀進記憶體時，不需要讀最少回合數。

盤面勝負結果有「贏」、「輸」跟「和」三種，因此一個盤面的「勝」、「負」跟「和」都會用到 2 Bits 的空間儲存。每一個不同盤面都有其勝負結果，因為盤面數量還是太過龐大，以致於還是無法將資料庫完全存入記憶體內。為了解決記憶體的問題，更進一步使用 1 Bit 來表示盤面勝負值，將八個盤面勝負結果使用 1 Byte 儲存，如此可以將原需要 9 GBytes 的記憶體空間壓縮成 1 GBytes 的記憶體空間。

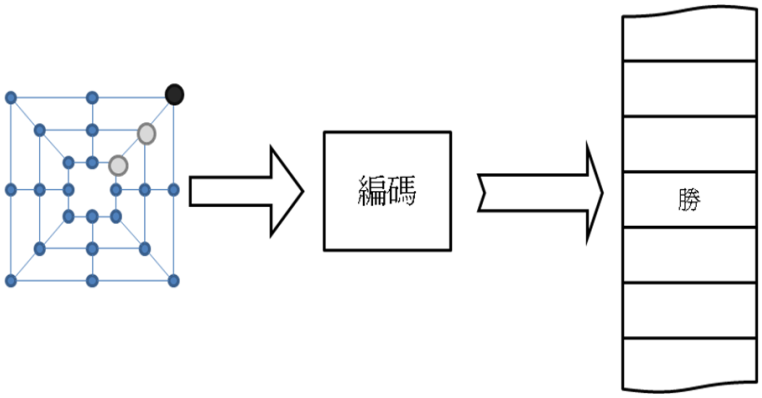


圖2-6　儲存盤面勝負結果示意圖

1 Bit 的空間只能表示兩種狀態，為配合資料庫盤面勝負結果以 1 Bit 儲存的方式寫入記憶體，特別將搜尋演算法進行改良。搜尋的目的是想要搜尋在一開始的盤面先手是必勝、必輸還是和棋，原本只要搜尋一次的演算法，為了配合減少記憶體的使用，而改變為搜尋兩次，第一次搜尋先手是否必勝，將盤面的勝負結果儲存為先手勝或是先手不會勝兩種狀態，第二次搜尋先手是否必敗，將盤面的勝負結果儲存為先手必敗或是先手不會必敗兩種狀態，經過兩次搜尋之後，就可以確認先手是必勝、必敗或是和棋。

觀察玩家實際玩 Nine Men's Morris 的情形，發現到放子階段結束時的盤面大多是盤面上還存在的白子跟黑子數量都為九子或八子的盤面，在求解放子階段時，能將需要讀取的行子階段資料庫縮減至八子對九子、九子對八子、八子對八子的資料庫，這樣一來將需要的記憶體空間縮減至 115 MBytes，而對於不存在於八子對九子、九子對八子或八子對八子資料庫的盤面利用估值函數的方式給與一個值。

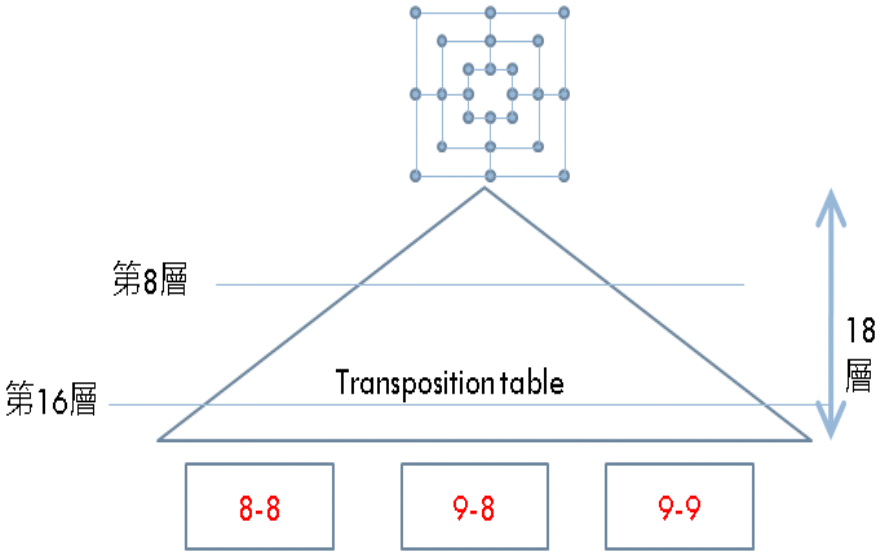


圖2-7　放子階段搜尋樹示意圖

為了減少讀取資料庫的次數，特別在搜尋樹第十六層的地方加上 Transposition table，因為為所有層數都製作 Transposition table 會過於龐大，所以只在第十六層，並且在搜尋樹的前八層製作一個開局庫，先計算出所有可能出現的盤面和到達盤面的路徑，如此可以減少電腦計算的時間，以加速求解 Nine Men's Morris 問題的速度。

第三節、　回溯分析演算法

回溯分析演算法是利用已知的結果反過來推出前一個情況結果的方法，從最後結果開始反過來尋找前一個情況的結果，最理想情況可以一路反過來推到一開始的狀態。回溯分析演算法最常被用來解決棋類遊戲中盤面循環的問題，而直棋遊戲在行子階段就存在盤面循環的問題，即於遊戲過程中存在回到已經走過盤面的問題。

計算 Nine Men' s Morris 問題行子階段的所有盤面結果，依照回溯分析演算法的程序：

- 初始化
- 輸盤面回溯
- 贏盤面回溯

初始化即產生所有盤面，並且賦予每一個盤面一個值表示「贏」、「輸」、「和」或「未知」。跟據 Nine Men's Morris 的遊戲規則決定初始盤面的值，例如某方棋子已經無空位可以移動就可以將其設為已輸盤面。

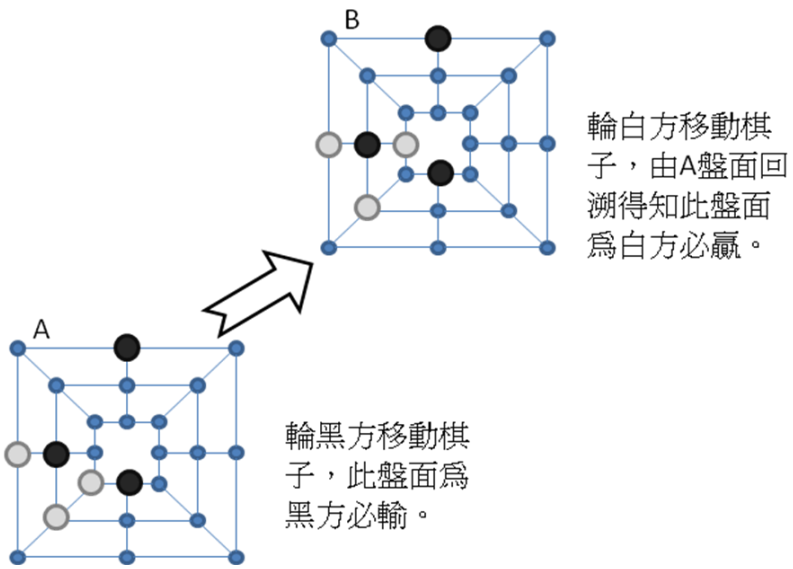


圖2-8　輸盤面回溯的示意圖

第二步搜尋輸的盤面，由盤面勝負值為輸的盤面可以回溯至上一個盤面並設定上一個盤面勝負值為贏。較直覺的想法就是在上一個盤面時，知道有一種走法可以走到對方必輸的盤面，那麼只要選擇該走法，己方必定會贏。

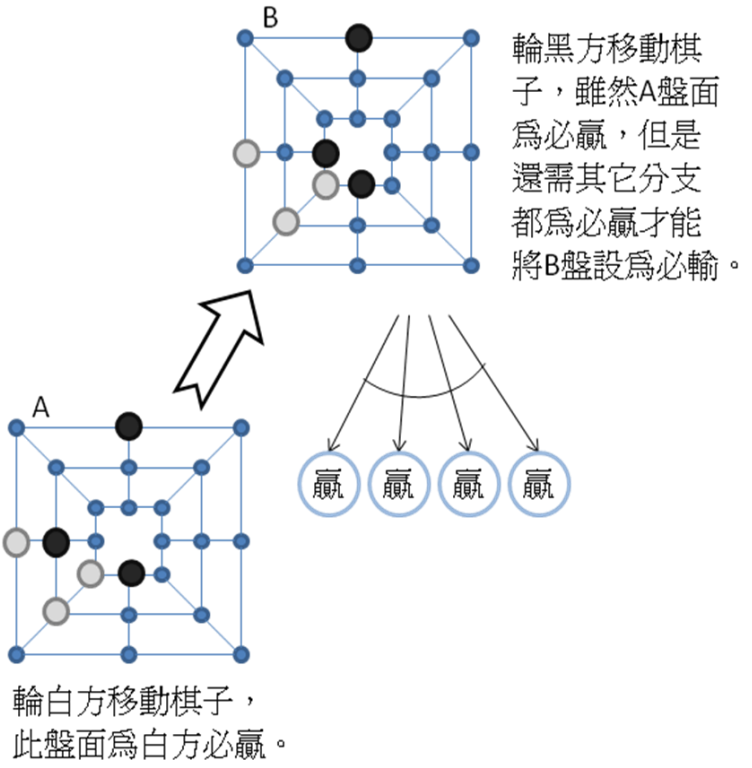


圖2-9　贏盤面回溯示意圖

第三步搜尋贏的盤面，現實玩家都會避免走到讓對方會贏的盤面，因此在贏盤面回溯至上一個盤面的時候，程式必需要判斷上一個盤面的其它還可選擇的走步是否都是已知結果為贏的盤面。

一直不斷循環第二步和第三步直到最後所有盤面的勝負結果都不變才停止程序，最後儲存結果時，將未知跟和都存為和棋。

本論文將一個盤面可以產生的所有結果為非贏的盤面數量稱之為「分支度」，Ralph Gasser 在贏盤面回溯程序時，為每一個贏盤面回溯至的盤面額外增加一個記憶體空間用來儲存該盤面的分支度。

每一個贏盤面在回溯到前一個盤面時，都會讀取前一個盤面的分支度，並將其分支度減一，當分支度降至為零時，表示前一個盤面的所有可以產生的盤面勝負結果都為贏，就可以將前一個盤面的勝負結果設為輸。

Ralph Gasser 使用一個 buffer 來儲存已知結果為「贏」跟已知結果為「輸」的盤面，初始化時將判定輸贏的盤面存入 buffer。當初始化結束之後，從 buffer 裡取出一個盤面，若盤面勝負結果為「輸」，則進行輸盤面回溯程序，並且將原本結果「未知」改變為「贏」的盤面在存入 buffer；若盤面勝負結果為「贏」，則進行贏盤面回溯程序，並且將原本結果「未知」改變為「輸」的盤面在存入 buffer。直到存在 buffer 裡面的盤面都被取出來計算後就停止程序，當 buffer 裡的盤面數量變為零時，就表示完成贏盤面回溯及輸盤面回溯後沒有改變任何盤面勝負結果。

第四節、　殘局庫

Nine Men's Morris 行子階段的所有盤面數量非常的龐大，如果要一次處理所有的盤面幾乎是不可能的事，因此如何將盤面分開來處理就變的非常重要了，在"Applying Retrograde Analysis to Nine Men's Morris"論文中，將殘局庫以盤面上雙方子力數量分割成許多小資料庫。

在行子階段時，移動棋子可能產生可以吃子的情況，本論文中稱之為「吃子步」；而移動棋子不能產生可以吃子的情況，則稱之為「走子步」。「吃子步」是不會造成盤面循環的情形，因為被吃的棋子不能再回到棋盤上，而「走子步」因

為盤面雙方棋子都不會減少，很有機會產生盤面循環的情形。依照「吃子步」跟「走子步」的特性以雙方棋子數量為分割依據將資料庫分割成較小的資料庫。

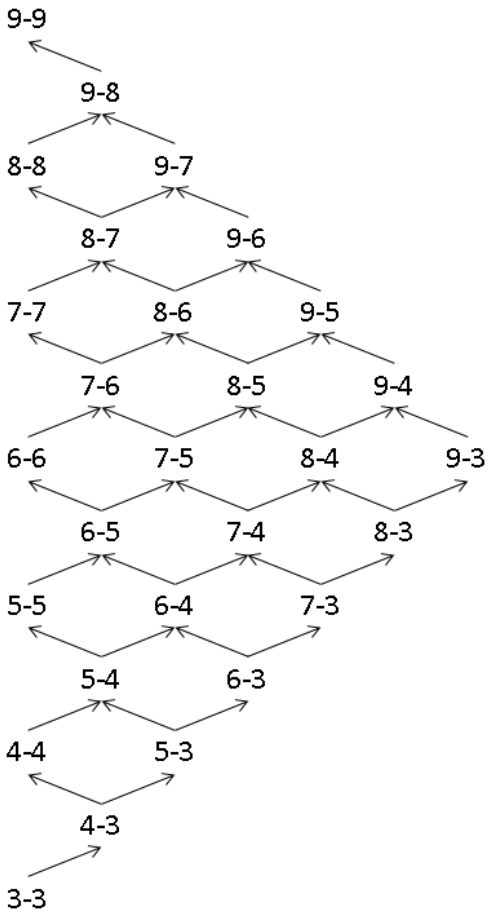


圖2-10　Ralph Gasser 的資料庫分類

不同資料庫之間以「吃子步」連結，五子對四子的盤面只可能會到達四子對四子或是五子對三子兩種盤面，因此資料庫的相互關係連結如圖 2-10。又因為利用回溯分析演算法來求盤面勝負結果，所以是從三子對三子的盤面開始慢慢一步一步往雙方更多棋子的盤面回溯。

相同棋子的盤面還可以分為輪到黑方移動棋子和輪到白方移動棋子兩種，雙方子力分佈跟攻守互換，得到的答案也會是相同。例如黑方現在擁有四子，白方

擁有三子的盤面輪到黑方移動棋子的結果是黑方贏，棋子的位置不變，將黑色棋子變白色，白色棋子變黑色，輪到黑方變成輪到白方移動棋子，則最後結果仍是白方贏。資料庫裡存的是盤面的棋子跟現在是輪到哪一方移動棋子，因為黑白雙方互換也可以得到相同的結果，所以四子對三子的資料庫可以當作三子對四子使用。

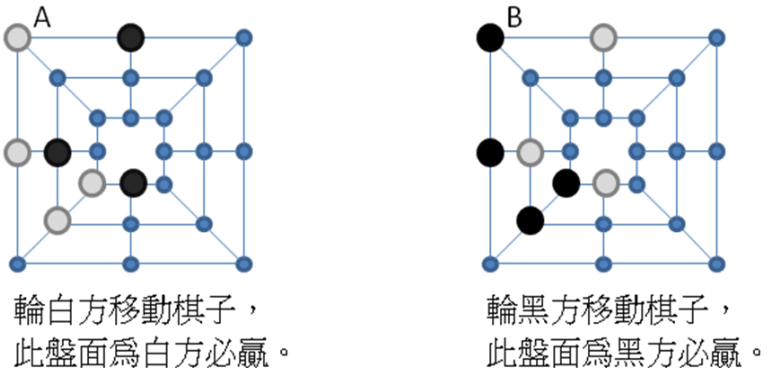


圖2-11 A 跟 B 的盤面勝負結果其實是相同

Ralph Gasser 證明了 Nine Men's Morris 是先手必和的遊戲，在放子階段運用了 alpha-beta search 演算法；在行子階段運用了回溯分析演算法跟殘局庫。在此基礎上，我們為了得知台灣直棋遊戲是先手必勝、先手必敗還是和棋的問題，特別針對台灣直棋的特性對回溯分析演算法及殘局庫進行改良。

第三章、 如何破解台灣直棋

台灣直棋是 Twelve Men's Morris 的變體的一種，流行於台灣地區的鄉土遊戲，雙方各有 12 顆棋子，其棋盤是由 24 個點組成的三同心的正方形，直斜線交叉，棋盤上有穿越矩形的線有八條，沿矩形的線有十二條。有三種方式可以吃子，「直」之外還有「擔」和「槓」，於兩階段都可以適用三種吃子方式，當滿足兩種以上吃子情況時，只能選一種方式吃子。

以下列出台灣直棋的規則：

- 台灣直棋為 2 人玩之遊戲棋，雙方各執 12 個棋子。
- 開始時每人每次下一個棋子於棋盤上，若三個棋子成一直線則可拿掉對方任何一顆棋子。
- 若己方主動造成「擔」的情形，則可拿掉左右對方的棋子。
- 若己方主動造成「槓」的情形，則可拿掉中間對方的棋子。
- 待 12 個棋子下完，則可移動己方任何一個棋子，使成上述情況直到對方只剩 2 個棋子時為勝方。

本論文中特別為台灣直棋將"Solving Nine Men's Morris"所提到的演算法加以修改成適合台灣直棋，並且加入提昇運算速度的方法。

第一節、 回溯分析演算法的改良

回溯分析演算法主要分成兩個部分，即初始化部分跟回溯部分，在初始化時需要產生所有盤面，並給予一個值表示盤面勝負。決定盤面勝負的方式依照規則來決定，有兩個規則可以在初始化就決定勝負結果，其一為輪到走子的那方無棋子可以移動時，判定該方為輸，其二為當輪到走子的那方於盤面上的棋子數少於三顆時，判定該方為輸。

當產生的盤面存在吃子步時就需要讀取吃子步產生盤面的資料庫，為了準確的判斷此盤面的勝負結果，需要將所有吃子步生成的盤面都看一次，如果吃子步生成的盤面有一個的勝負結果為輸，則可以將產生的盤面勝負結果設定為贏。但是如果每一個吃子步生成的盤面勝負結果均為輸，也不保證產生的盤面勝負結果為輸，再者如果吃子步生成的盤面有一個為和棋，只保證產生的盤面不會為輸，但是不保證產生的盤面為和棋。

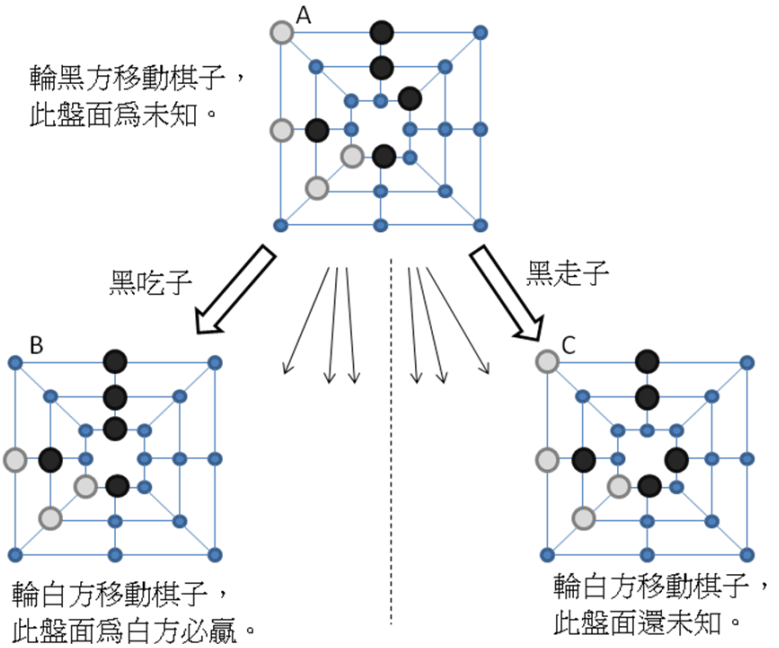


圖3-1　吃子步跟走子步示意圖

在回溯程序時，利用「分支度」來表示一個盤面還可以產生多少勝負結果不為贏的盤面，將計算分支度的動作提前到初始化階段時進行，如此可以一併計算吃子步的結果。

讀取資料庫可以配合資料庫儲存的方式而改變讀取的方式，我們特別將讀取資料庫的程式從初始化程序中脫離開來。在初始化程序跟回溯程序中間加入讀取

資料庫的程序，如此做的原因在於實作程式時需要讀取很多資料庫，為了節省記憶體的使用量，特別將讀取資料庫的行為獨立出來。

當讀取資料庫裡的盤面時，每一次只讀一個資料庫進記憶體，在資料庫裡每讀取一個盤面勝負結果為「輸」，就利用輸盤面回溯的方式，將可能產生此盤面的上一個盤面勝負結果設為「贏」；反之讀取到「贏」盤面時，就需要判斷上一個盤面還有沒有吃子步可以產生其它盤面。如此一次只需要讀取一個資料庫，當資料庫裡的盤面都搜尋過一次後，這個資料庫就再也不會被讀取到了，換而言之其佔用的空間可以釋放出來。

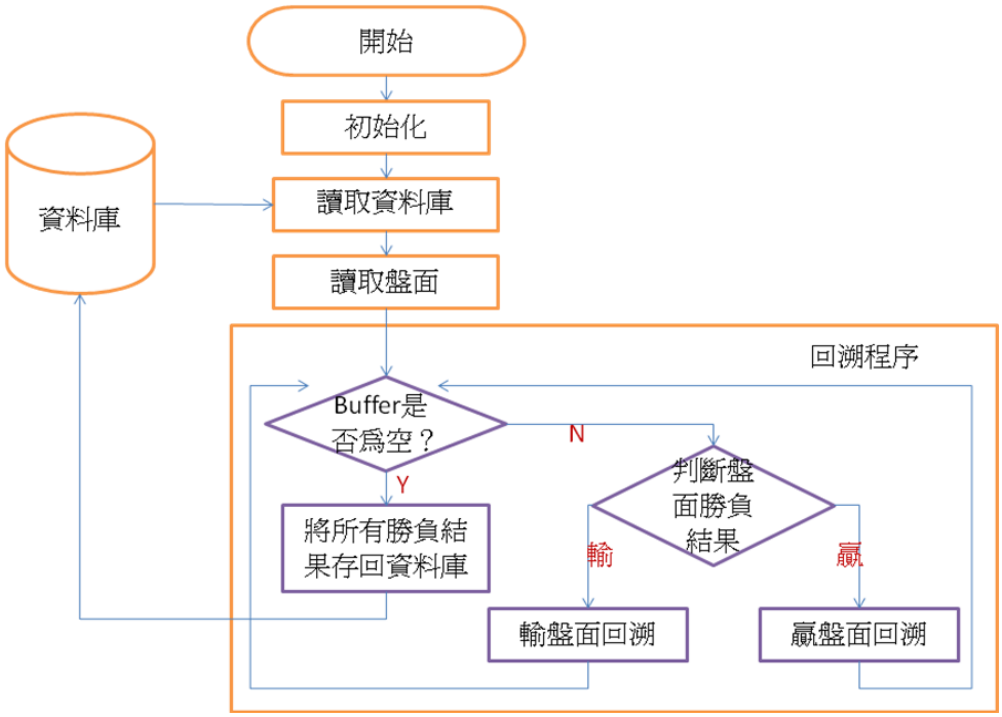


圖3-2　程式的流程圖

盤面勝負結果有「贏」、「輸」、「和」跟「未知」四種，在回溯程序中「贏」跟「輸」是確定不會改變的結果，因為盤面勝負結果為「贏」需要可能生成的盤

面群中有一個盤面的勝負結果為「輸」；而盤面勝負結果為「輸」的盤面需要任何一個生成盤面的勝負結果都為「贏」。

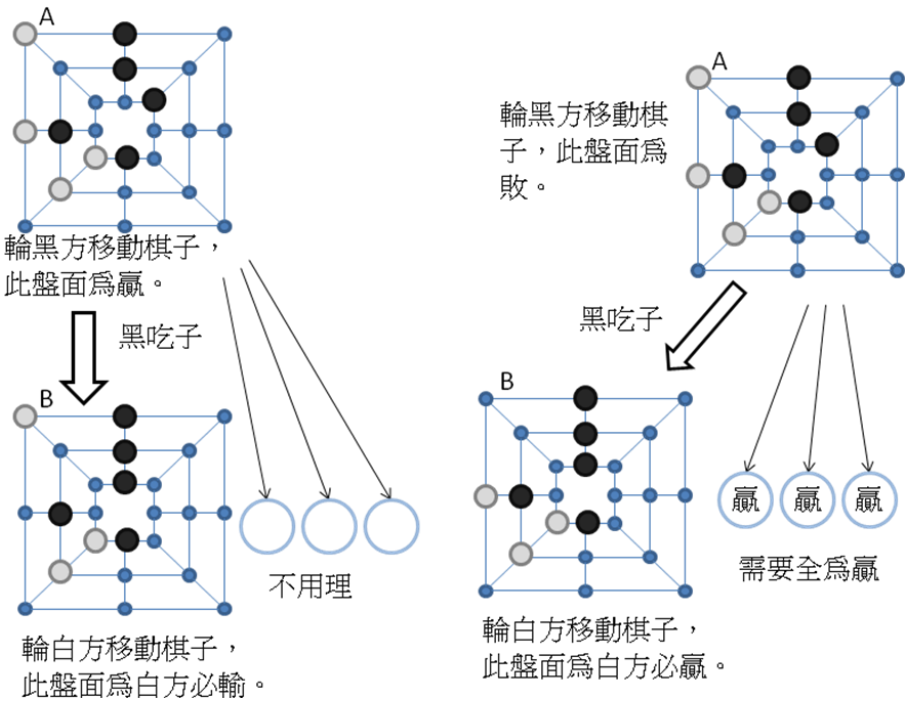


圖3-3　回溯程序時判定輸贏示意圖

但是「和」跟「未知」有可能會變成「贏」或是「輸」，所以程式的重點就在「和」跟「未知」的處理上，在最後結束回溯程序的時候，會將「和」跟「未知」同樣視為和棋，因為「未知」表示與勝負完全不相關。

配合資料庫讀取的方式，我們特別將「未知」跟「和」的意義作一個區分，「未知」定義為吃子後的所有盤面勝負結果都為「贏」;「和」定義為吃子後的所有盤面勝負結果中至少有一個盤面的勝負結果為「和」。在開始階段將不為「贏」或「輸」的盤面其勝負值設為「和」，在讀取資料庫時，利用贏盤面回溯的方式將原本設定為「和」的盤面修改成「未知」，如此區分在結束讀取資料庫的程序

之後，「輸」盤面回溯會修改「和」及「未知」的盤面，而「贏」盤面回溯只會修改「未知」的盤面，以期能減少存取記憶體的次數。

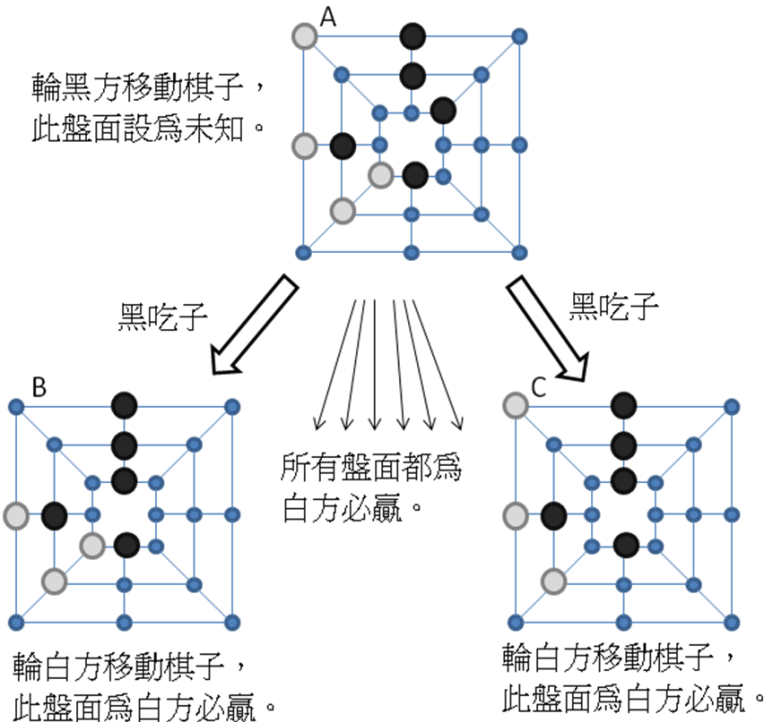


圖3-4　於讀取資料庫時設定盤面勝負示意圖

因為在回溯程序的「贏」盤面回溯時，會需要知道此盤面經由吃子步是否能到達勝負值為「和」的盤面。如果發現回溯的盤面其勝負值為「和」，則不用修改任何資訊，直接略過此盤面。

第二節、　殘局資料庫的改良

因為台灣直棋有「直」、「擔」跟「槓」三種吃子的情況，其中「擔」是當己方主動在沿著正方形的線段上，己方一枚棋子主動出現在兩枚對方棋子的中間位

置，進而吃掉敵方該兩枚棋子。因此台灣直棋有一次吃兩子的可能性，需要對殘局庫相互之間的連結關係作調整，以配合台灣直棋遊戲的特色。

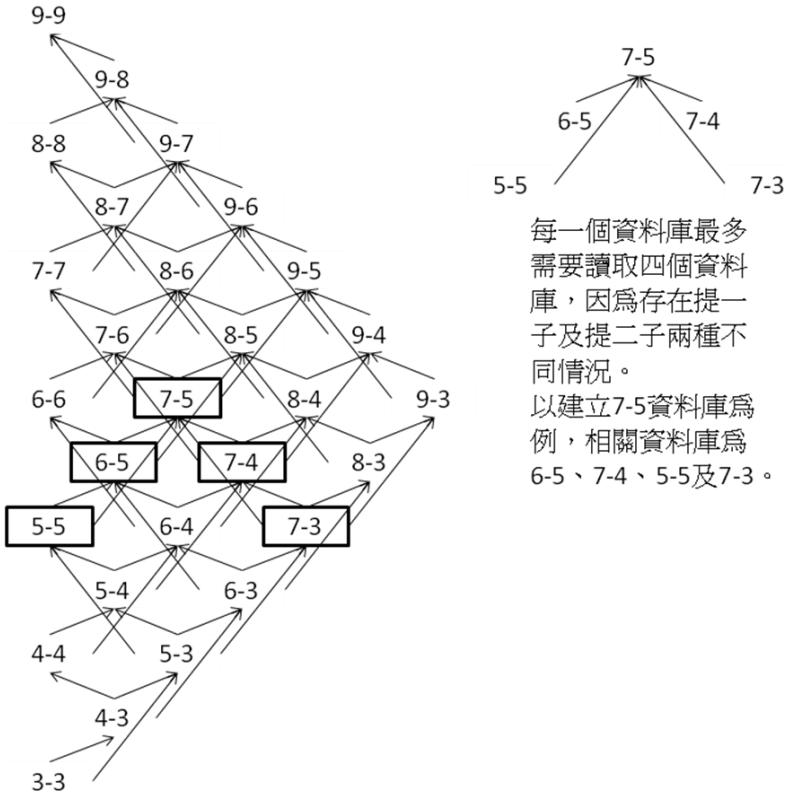


圖3-5　台灣直棋殘局庫部分關係圖

台灣直棋殘局庫之間的連結也是使用「吃子步」作為連結，跟 Nine Men's Morris 不同處在於台灣直棋的吃子情況有「直」、「槓」和「擔」三種，因此要生成其中一個殘局庫時，需要讀取的殘局庫比 Nine Men's Morris 多了一倍。

因為在台灣直棋遊戲規則中，雙方在開始的時候都擁有十二顆棋子，所以盤面需要考慮到十二子對十二子的情形，雖然十二子對十二子是剛好把盤面所有空點填滿，雙方都無棋子可移動，因而不用計算，但是對於十二子對十一子的情形，還是需要採用回溯分析演算來計算盤面的結果，考慮到記憶體空間的限制，我們打算再對殘局庫進行一次分割。

觀察直棋遊戲過程中的盤面，發現到一個現象，在行子階段出現循環盤面時，在同樣的盤面是絕對輪到同一方玩家移動棋子。因此發現到直棋遊戲具有如 8-puzzle 一樣擁有兩種盤面及其產生盤面的集合彼此互斥，進而將此特性運用在殘局庫，使得殘局庫能以棋子在盤面上位置分佈的不同來分割。

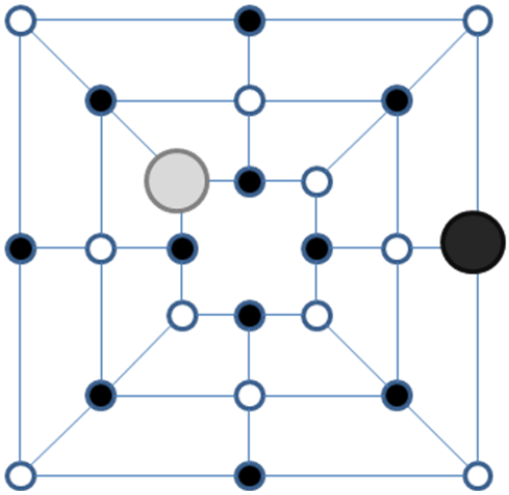


圖3-6　將交叉點塗上黑白兩色

如上圖，將交叉點塗上黑白兩種顏色作為區分，假設現在輪到黑方可以移動棋子，此時黑方只有三個白色交叉點可以選擇，當黑方選擇移動黑色棋子到上方白點後，空著的黑色交叉點就有 12 個，空著的白色交叉點就有 10 個。然後輪到白方可以移動棋子，當白方選擇移動白色棋子之後，空著的黑色交叉點就剩 11 個，空著的白色交叉點就增加為 11 個。

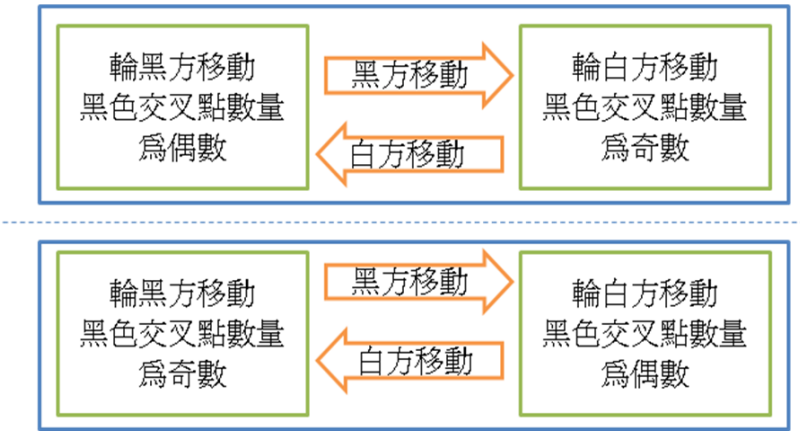


圖3-7　分割方法示意圖

如此進行下去，就可以發現當輪到黑方移動棋子時，在盤面上，空著的黑色交叉點總數為奇數；輪到白方移動棋子時，在盤面上，空著的黑色交叉點總數為偶數。

如何運用程式來實現這個想法，我們將盤面 24 個交叉點以 0 到 23 編號，將單數跟偶數點彼此交錯，讓移動棋子只能從單數交叉點移動到偶數交叉點，要計算單數交叉點的數量只需要運用加法運算跟 and 運算，就可以得到單數交叉點的總數量是奇數還是偶數。

將資料庫更細的分類之後，將原本一個資料庫存的資料分開成兩個資料庫儲存。其單一資料庫需要儲存的盤面數為原本的一半，但是總共資料庫數量也變為原本的兩倍。造成要生成一個新的殘局庫需要讀取八個已生成的殘局庫，即原本需要讀取的四個資料庫經過此分割後變為八個資料庫。

第三節、　放子階段的作法

放子階段是指將手中的棋子放置在盤面上的空白交叉點上，其搜尋深度固定加上不會出現循環盤面影響樹搜尋演算法的效率[6]，因此放子階段使用樹搜尋的方式求得第一步最佳放置棋子的位置，似乎是一件非常容易的事情。

樹搜尋的速度取決於一個節點的分支數跟搜尋深度，雖然在放子階段的搜尋深度是固定的，但節點分支數的大小卻不是固定的，因為吃子步數量最大可以是非吃子步兩倍，存在吃子步的盤面搜尋時間是不存在吃子步盤面搜尋時間的兩～三倍。我們一開始考慮採取 MIN-MAX search 演算法[4]進行裁剪，在 Six Men' Morris 的放子階段採用 MIN-MAX search 演算法實驗之後，發現到使用樹搜尋的方式並不足夠有效率的得到解答。

有鑑於此，我們嘗試在放子階段也使用回溯分析演算法尋找第一步最佳放置棋子的位置，首先對應放子階段的特性改良原本的回溯分析演算法，回溯分析演算法的主要觀念是從結束往開始回溯，在放子階段的結束即是行子階段的開始，因此結束盤面勝負結果就是行子階段盤面的勝負結果。放子階段時，還要考慮到的是現在雙方手中還有幾顆棋子還未放置於棋盤上，因為不同盤面表示的不止棋子在盤面上的分佈情形。

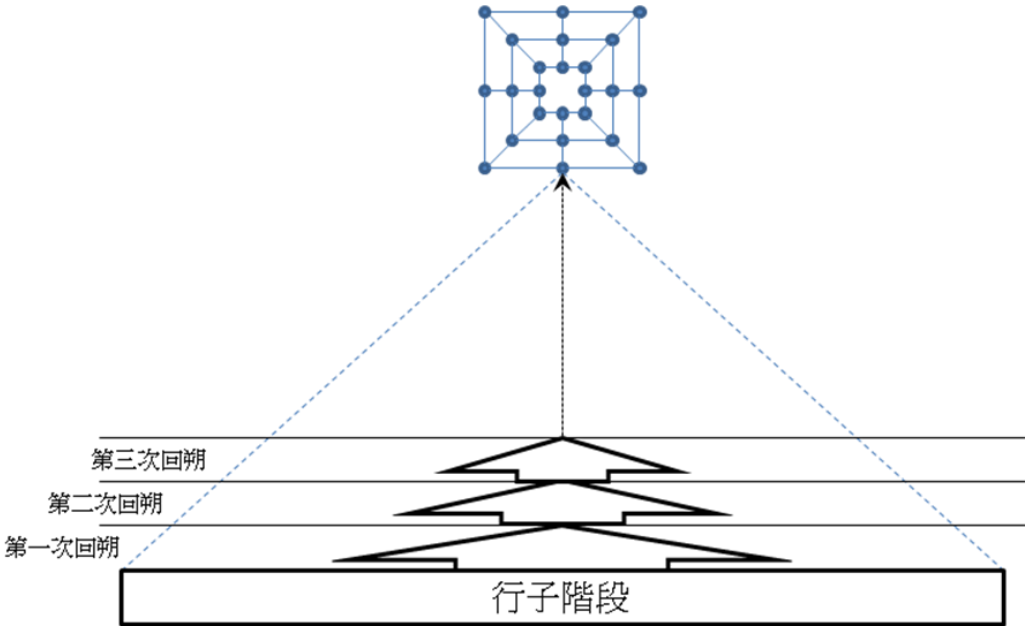


圖3-8　放子階段回溯分析演算法示意圖

配合放子階段特色改良的回溯分析演算法跟行子階段的差別，在於沒有初始化跟回溯的階段，只留下讀取資料庫部分。不需要初始化的原因在於放子階段不會出現結束盤面，也就是說在放子階段中每一個盤面都一定存在下一個盤面，反而言之就是放子階段的盤面都可以由其下一個盤面往回產生，因此不需要初始化程序。因為不會發生循環的情形，所以不會需要重覆檢查盤面勝負值是否有改變，只要將資料庫的盤面讀完，剩下的就是所有盤面的結果了。

儲存資料的方式是採用資料庫的想法，因為盤面數量太多而記憶體無法儲存所有盤面，所以需要將資料庫作一個分割，分割資料庫的方式是依據回合數量跟盤面雙方棋子數分配，不管有沒有吃子行為發生都會跳到另外一個資料庫。

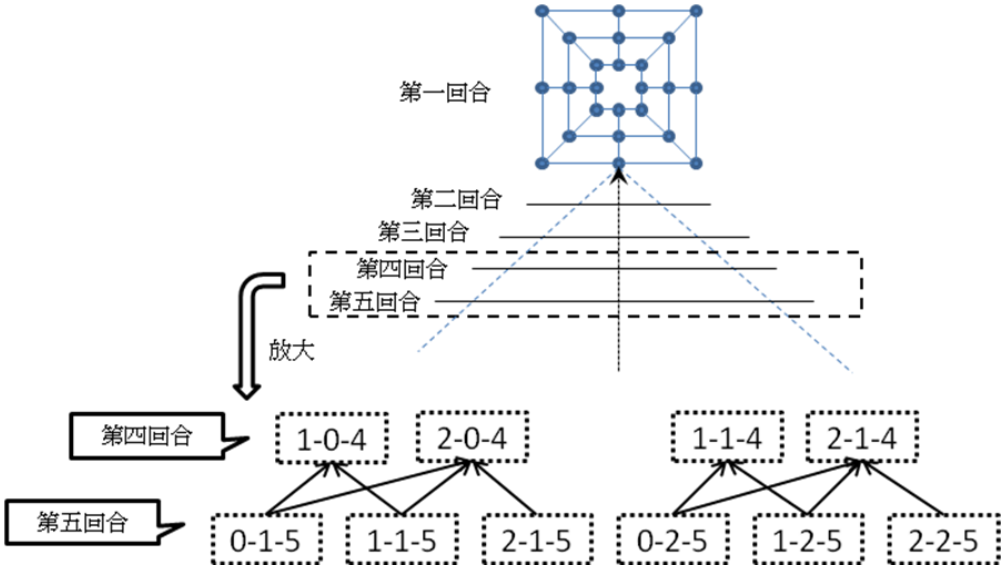


圖3-9　放子階段的資料庫部分關係圖

殘局資料庫以盤面上雙方棋子數及存在的回合數為命名，例如 1-0-4 表示第 4 回合盤面上黑方玩家棋子數為 1 顆；白方玩家棋子數為 0 顆。

放子階段存取硬碟的次數遠比行子階段還要多很多，由上圖可以發現要製作第 4 回合盤面黑方一個棋子白方一個棋子(1-1-4)的所有盤面勝負資料庫時，需要讀取兩個資料庫，然而在製作下 2-1-4 的所有盤面勝負資料庫時，也會重覆讀取相同的資料庫。為了減少從硬碟抓資料到記憶體的次數，將製作資料庫順序依使用時間先後次序排序，並且已經讀進記憶體的資料庫一直保留到確定沒有其它資料庫還需要讀取其內容的時候，再將其存在的空間留給其它尚未讀取到的資料庫使用。如此在理想狀態下可以使得將硬碟資料讀進記憶體的次數減少為原本的三分之一。

第四節、 Zobrist hashing 演算法

為了配合各種大小不同的資料庫，在 hash table 的設計時考慮最大需要的空間跟碰撞機率，我們採用 Zobrist hashing 演算法，期望達到同一個 hash table 能夠計算所有的資料庫。Zobrist hashing 的作法是將每一個特徵都給予一個隨機值當作那個特徵轉換的 hash key 值，在計算盤面 hash key 值的時候，先取得盤面裡存在哪些特徵，再將每一個特徵轉換成的 hash key 值運用 XOR 運算後產生的就是盤面的 hash key 值。

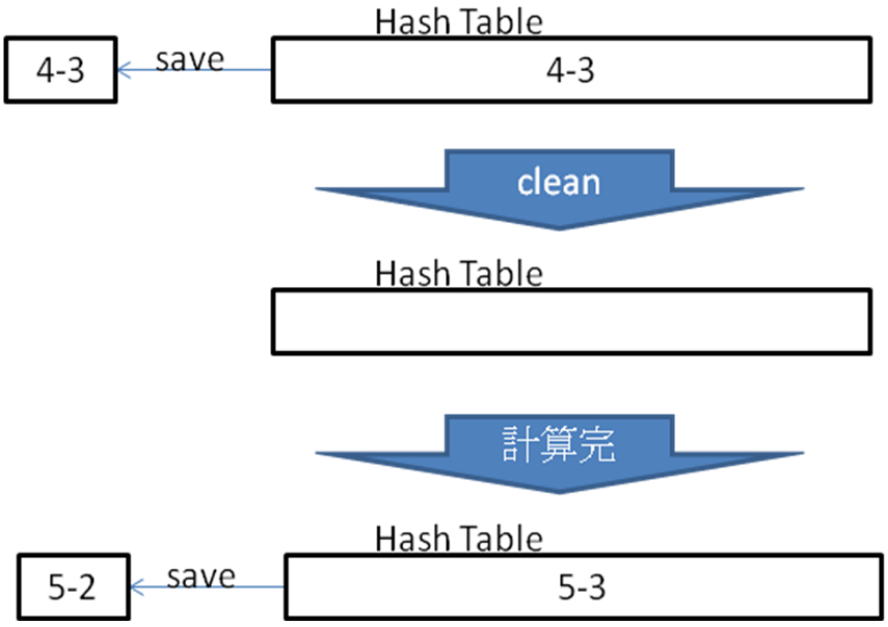


圖3-10　同個 hash table 在程式運用示意圖

Hash table 每次計算完一個資料庫都清空一次，讓下一個資料庫計算時，使用的 hash table 的空間不會被上一個資料庫的資料佔用。例如計算完(4-3)的資料庫並存入硬碟之後，就將 hash table 裡面的所有資料清除，再進行下一個資料庫的計算。

程式所取的特徵為盤面上棋子的所在位置以及棋子屬於哪一方來轉換成 Zobrist hash key。Hash table 演算法的好處在於對於整體數量相當龐大的資料，而在計算時只需使用到一部分資料。用在棋類遊戲時，能有效的減少儲存需要的空間，如暗棋[5]。Zobrist hash 演算是利用 rand 的方式，希望達到將資料以一對一的方式對應到 hash-table 裡，理想狀況是每筆資料在 table 裡的位置都不相同，並且資料在 hash table 裡的分佈是均勻的。Zobrist hash 的好處是只要擁有足夠大的 hash table，相同的特徵鍵值就可以適用各種不同大小的資料庫。

第四章、 其它加速方法

第一節、 旋轉及對稱

台灣直棋遊戲盤面由三個同心正方形組成，雙方棋子也沒有編號，因此有許多盤面經過旋轉及對稱運算之後，其實是相同盤面。對於相同的盤面可以不需要再重新計算結果一次，運用旋轉及對稱運算可以大幅度縮小我們計算的盤面數量，同時能減少記憶體的使用跟計算的時間。

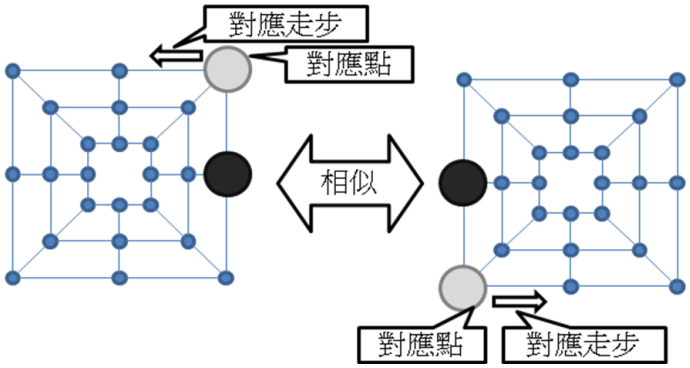


圖4-1 相似盤面的例子

「相似」方式除了旋轉 (可轉 90 度、180 度、270 度)跟左右對稱(以斜線跟正方形的中線為中心) 之外，還有最裡面的正方型跟最外面的正方型的對稱，因此可以說存在 16 個盤面是相似的，而且第一步只需要考慮四個位置放置棋子。而相似盤面的點和點是採取 1 對 1 對應，因此可以放置棋子的點也是相同的，能產生的下一個盤面也是相似的盤面，在本論文中定義「相似」的意義為對應點產生對應的走步因而產生相同的結果，換句話說相似盤面所得到的結果必定相同。

回溯分析演算法中，對於盤面勝負結果為「贏」跟「輸」的盤面都會做一次回溯的運算，如果沒有利用旋轉及對稱運算，則會在相似盤面上重覆相同的動作只為了得到其實早就已經完成的結果。

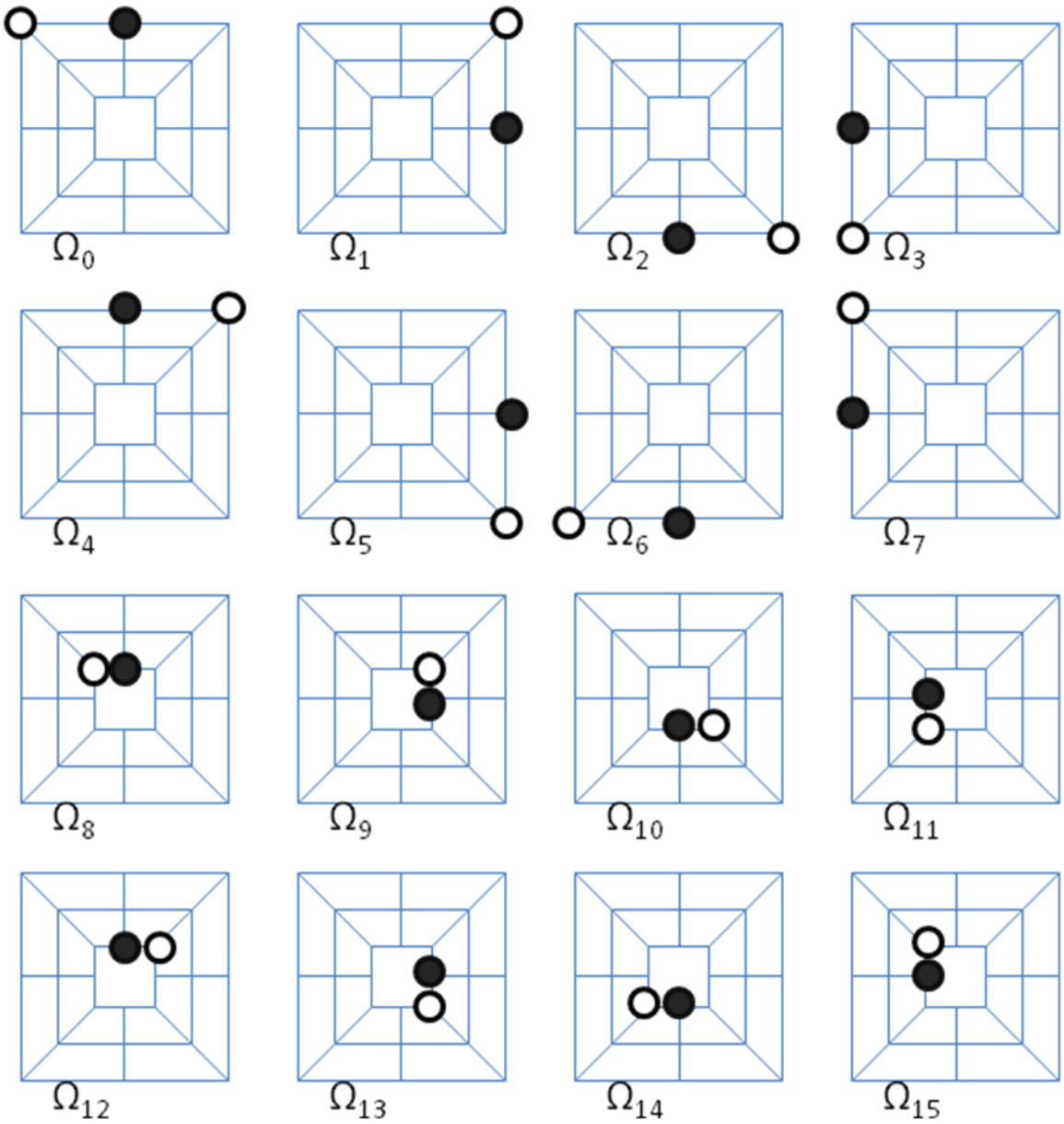
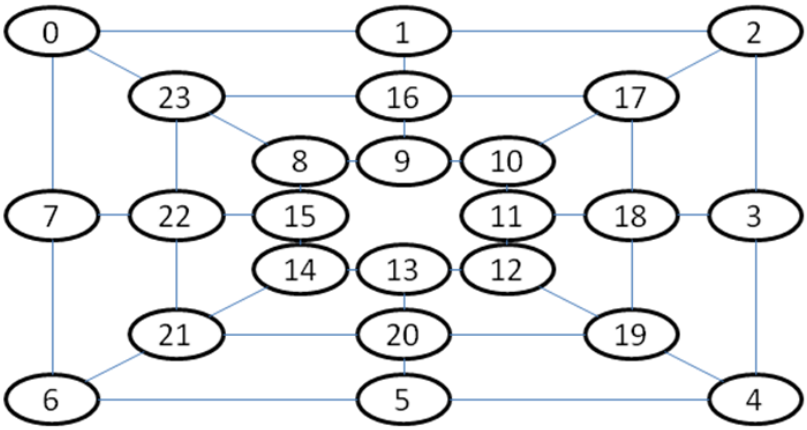


圖4-2　16 個相似盤面

我們用數學符號 Ω_0～Ω_{15} 表示 16 種旋轉及對稱運算，並將棋盤上的每一個交叉點編號，使用數學式子表示旋轉及對稱運算，數學式子可以幫助我們明白地看

出 Ω_0～Ω_{15} 中各存在著什麼樣的盤面會經過旋轉及對稱運算之後，會回到運算前的盤面，如此不止可以計算各個資料庫最大需要計算的盤面數量，更可以利用此特性加速旋轉及對稱運算。



Ω_1 = (0 2 4 6)(1 3 5 7)(8 10 12 14)(9 11 13 15)(16 18 20 22)(17 19 21 23)
(0 2 4 6) 表示 0 到 2、2到4、4到6、6到0的循環。

圖4-3　相似盤面數學表示法示意圖

如此編號的方式是考慮上一章所提到存在兩種盤面及產生子盤面的集合彼此互斥，以及後面會提到的加速旋轉及對稱運算方法。以此種編號方式將每一個旋轉及對稱運算轉換數學表示式，如下表。

表一　　旋轉及對稱數學表示式

旋轉及對稱方式	文字表示	數學表示式
Ω_0	無旋轉及對稱	(0)(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(13)(14)(15)(16)(17)(18)(20)(19)(22)(21)(23)
Ω_1	旋轉 90 度	(0 2 4 6)(1 3 5 7)(8 10 12 14)(9 11 13 15)(16 18 20 22)(17 19 21 23)
Ω_2	旋轉 180 度	(0 4)(1 5)(2 6)(3 7)(8 12)(9 13)(10 14)(11 15)(16 20)(17 21)(18 22)(19 23)
Ω_3	旋轉 270 度	(0 6 4 2)(1 7 5 3)(8 14 12 10)(9 15 13 11)(16 22 20 18)(17 23 21 19)

Ω_4	左右對稱	(0 2)(1)(3 7)(4 6)(5)(8 10)(9)(11 15)(12 14)(13)(16)(17 23)(18 22)(19 21)(20)
Ω_5	左右對稱+旋轉 90 度	(0 4)(1 3)(2)(5 7)(6)(8 12)(9 11)(10)(13 15)(14)(16 18)(17)(19 23)(20 22)(21)
Ω_6	左右對稱+旋轉 180 度	(0 6)(1 5)(2 4)(3)(7)(8 14)(9 13)(10 12)(11)(15)(16 20)(17 19)(18)(21 23)(22)
Ω_7	左右對稱+旋轉 270 度	(0)(1 7)(2 6)(3 5)(4)(8)(9 15)(10 14)(11 13)(12)(16 22)(17 21)(18 20)(19)(23)
Ω_8	裡外對稱	(0 8)(1 9)(2 10)(3 11)(4 12)(5 13)(6 14)(7 15)(16)(17)(18)(19)(20)(21)(22)(23)
Ω_9	裡外對稱+旋轉 90 度	(0 10 4 14)(1 11 5 15)(2 12 6 8)(3 13 7 9)(16 18 20 22)(17 19 21 23)
Ω_{10}	裡外對稱+旋轉 180 度	(0 12)(1 13)(2 14)(3 15)(4 8)(5 9)(6 10)(7 11)(16 20)(17 21)(18 22)(19 23)
Ω_{11}	裡外對稱+旋轉 270 度	(0 14 4 10)(1 15 5 11)(2 8 6 12)(3 9 7 13)(16 22 20 18)(17 23 21 19)
Ω_{12}	裡外對稱+左右對稱	(0 10)(1 9)(2 8)(3 15)(4 14)(5 13)(6 12)(7 11)(16)(17 23)(18 22)(19 21)(20)
Ω_{13}	裡外對稱+左右對稱+旋轉 90 度	(0 12)(1 11)(2 10)(3 9)(4 8)(5 15)(6 14)(7 13)(16 18)(17)(19 23)(20 22)(21)
Ω_{14}	裡外對稱+左右對稱+旋轉 180 度	(0 14)(1 13)(2 12)(3 11)(4 10)(5 9)(6 8)(7 15)(16 20)(17 19)(18)(21 23)(22)
Ω_{15}	裡外對稱+左右對稱+旋轉 270 度	(0 8)(1 15)(2 14)(3 13)(4 12)(5 11)(6 10)(7 9)(16 22)(17 21)(18 20)(19)(23)

數學表示式如何便於我們找出經旋轉及對稱運算後還是同樣盤面的盤面？以 Ω_{15} 為例，數學表示式為(0 8)(1 15)(2 14)(3 13)(4 12)(5 11)(6 10)(7 9)(16 22)(17 21)(18 20)(19)(23)，將括號內的位置編號放上相同的顏色的棋子，即是經過 Ω_{15} 運算還是回到自己本身的盤面。

如果將相似盤面視為相同盤面，想要計算其所有盤面數量，需要先找出每一種旋轉及對稱運算會轉回原本盤面的盤面，並計算其數量，將所有數量相加之後除以旋轉及對稱運算的種類數量：

G：表示相似運算的集合，$G = \{ \Omega_0 , \Omega_1 , \cdots , \Omega_{14} , \Omega_{15} \}$

|G|：表示相似運算的數量。

Ψ(Ω)：表示 Ω 運算之後回到原本盤面的盤面數量。

計算將相似盤面視為同一種盤面的所有盤面數量公式：

$$\frac{1}{|G|}\sum_{\Omega_i \in G} \Psi(\Omega_i)$$

第二節、 回溯分析演算法運用旋轉及對稱

如何判斷相似盤面又在何時判斷相似盤面可以省去不必要的時間呢？回溯分析演算法是由已知勝負結果的盤面，往能夠產生此盤面的盤面回溯結果，因此存在一個動作是計算能夠產生當下盤面的所有盤面，我們稱產生當下盤面的盤面為前一個盤面。在計算分支度的時候，是計算當下盤面能產生的所有盤面的勝負結果不為「贏」的數量。在初始化階段是生成資料庫需要的所有盤面並且給予一個暫時的盤面勝負結果，此階段一定需要計算生成的盤面是不是已生成盤面的相似盤面，若是相似盤面則當成不存在的盤面。

在計算分支度的時候，可能產生兩個彼此相似的盤面，如果不記錄產生過的盤面或不考慮相似盤面，就會因為相似盤面的關係而造成分支度計算錯誤。舉個

例子來說明，B 盤面為 A 盤面的下一個盤面，意指某方在 A 盤面上移動了該方棋子生成了 B 盤面。B 盤面的相似盤面 C 也為 A 盤面的下一個盤面，如此在計算 A 盤面的分支度時，會將 B 盤面跟 C 盤面當作兩個不同盤面，但是在初始化階段，B 盤面跟 C 盤面只會有一個盤面視為存在，假設 C 盤面是不存在的盤面，對於 A 盤面來說它會認為它擁有 B 盤面跟 C 盤面，而程式看不到也不會對 C 盤面執行回溯程序，進而造成 A 盤面以為它還有一個 C 盤面未知，到程式執行結後，A 盤面的勝負結果就會變成「和」。

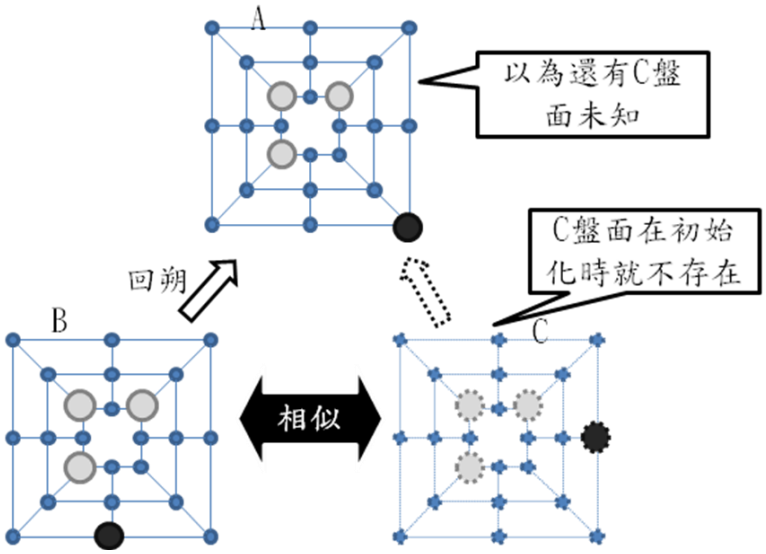


圖4-4　計算分支度錯誤原因示意圖

在計算分支度的地方加入了比較相似盤面後，在執行回溯程序時就出現同樣是相似盤面的問題。舉例說明，B 盤面為 A 盤面的上一個盤面，意指某方在 B 盤面上移動了該方棋子生成了 A 盤面，B 盤面的相似盤面 C 也為 A 盤面的上一個盤面，而程式會因為 C 盤面是不存在的盤面，轉而找 C 盤面的相似盤面 B。如此在執行 A 盤面回溯程序時，如果不對相似盤面作比較並將所有相似當作相同盤面

處理，會造成 A 盤面對 B 盤面執行兩次同樣的回溯程序，B 盤面的分支度就會被減兩次。但是在計算 B 盤面分支度時，只將 A 盤面視為一個盤面，如此一來分支度失去原本用來表示還有多少盤面勝負結果不為「贏」的作用，進而 B 盤面勝負結果的計算就會出錯。

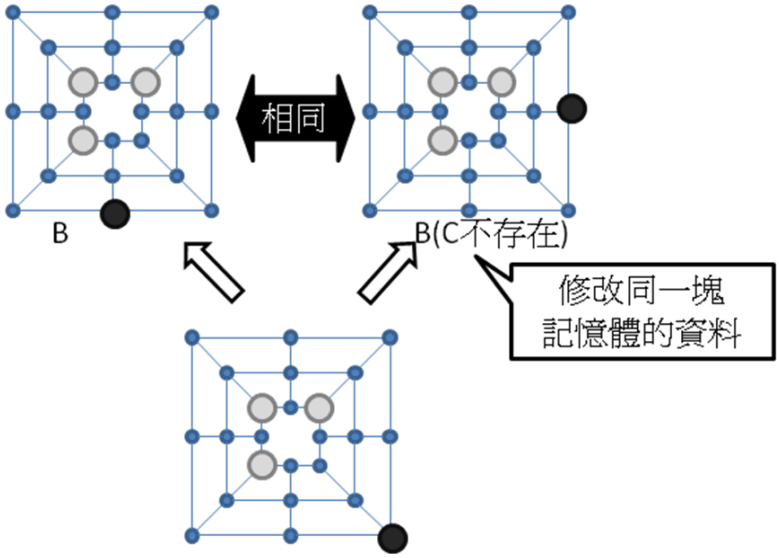


圖4-5　相似盤面於回溯程序的問題示意圖

解決辦法是在回溯程序裡加入相似盤面的比較並且將所有的相似盤面看作一個盤面處理或不使用旋轉及對稱運算。

第三節、 加速相似運算

在初始化、計算分支度跟回溯程序裡加入了比較盤面是否為已存在盤面的相似盤面，如果單純將盤面用十六種旋轉及對稱運算再比較有沒有出現相似盤面，那麼在整個程式執行的時間上不僅沒有減少反而會增加計算時間，因為一個經過雙方移動一、兩棋子就會輸的盤面，在初始化階段會先經過十六種旋轉及對稱運算，再來是在贏盤面回溯程序時至少經過十六種旋轉及對稱運算，最後計算分支

度的時候又會至少再經過十六種旋轉及對稱運算。如果沒有辦法加速該運算，反而會造成程式執行時間的增加。

如何加速旋轉及對稱運算？主要的想法在程式開始時建立一個旋轉及對稱的表格，將相似盤面統一成一個基礎盤面，只要記錄相似盤面需要運用哪一種相似運算到達基礎盤面，如此就不需要做十六次的盤面重排的動作，然而總共存在有 282429536481(＝3^{24})個盤面，一個盤面的資訊用 1 Byte 來儲存也需要 282 GBytes 的空間，於是開始思考能不能以小部分的盤面就可以知道其相似盤面。觀察旋轉及對稱的方式發現到對大部分的相似盤面來說，將位於中間的正方形拿掉，對 16 種相似運算沒有影響。

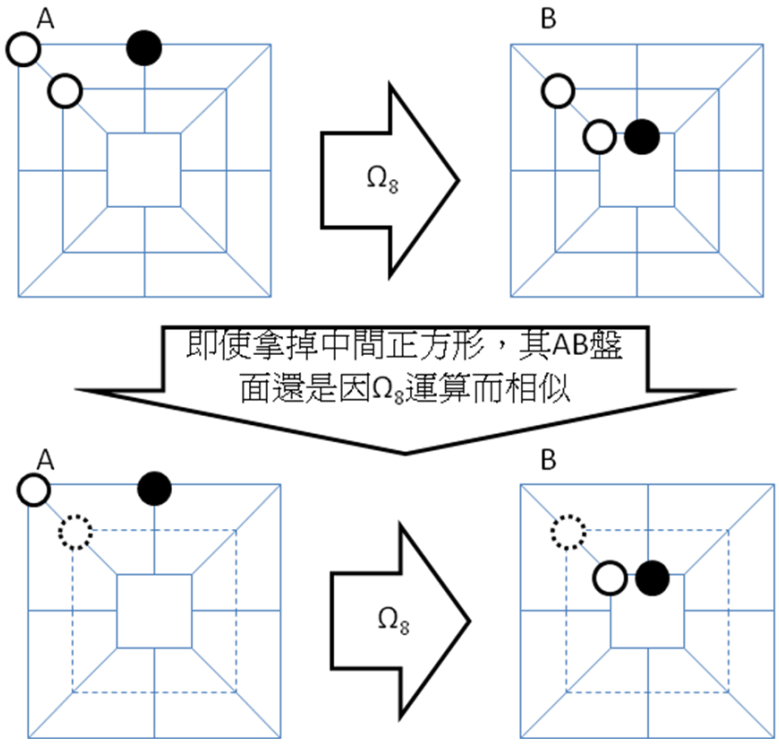


圖4-6　去掉中間正方形不會改變相似運算的例子

因此可以只需要看最外圈和最內圈的正方形上棋子分布決定需要使用哪一種相似運算，這樣一來就能夠將 282 GBytes 的記憶體空間需求減少為 43 Mbytes 的記憶體空間需求。

但是對小部分盤面來說，以最外圈和最內圈的正方形來判斷就會影響到其相似運算種類，特別觀察此類小部分盤面會發現到一個特點，此類盤面若只看最外圈和最內圈的正方形，看到的其實是相同盤面，而若以整體來看，則是看到不一樣的盤面。也就是說只看最外圈和最內圈的正方形時，此類盤面具有經過運算是會回到原本盤面的特性，因此特別為那些在看最外圈和內圈正方形時，經過旋轉運算會回到自己的盤面，作一個分類記錄。

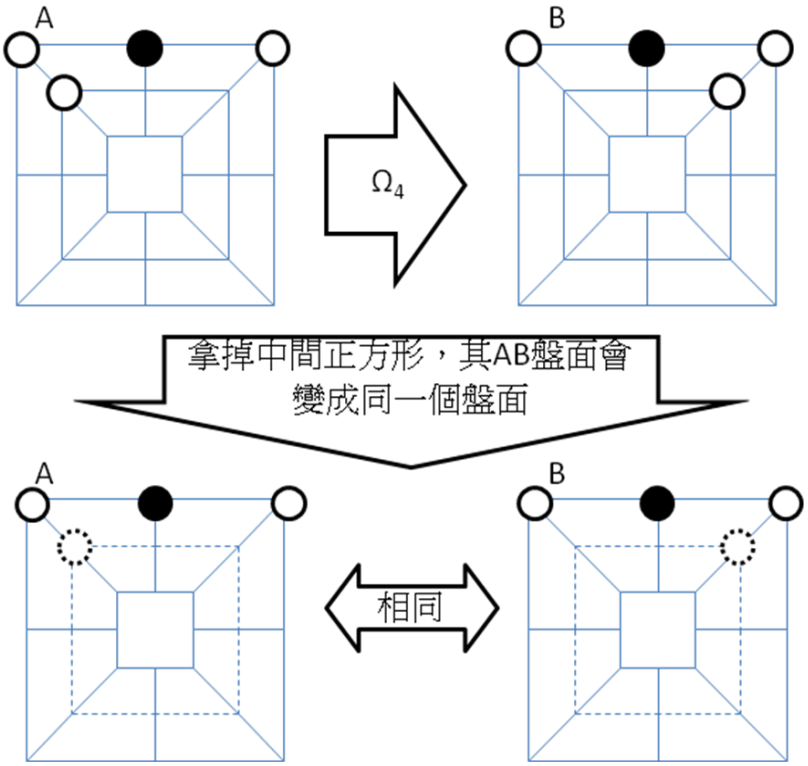


圖4-7　掉中間正方形影響到相似運算

表二　有相似計算跟沒相似計算的比較

黑子數	白子數	相似盤面為相同的所有盤面數量	相似盤面為不同的所有盤面數量	縮小比例
3	3	169626	2691920	0.063013017
4	3	760398	12113640	0.062772049
5	3	2580390	41186376	0.062651543
6	3	6875320	109830336	0.062599463
7	3	14725688	235350720	0.062569122
8	3	25763836	411863760	0.062554268
9	3	37209228	594914320	0.062545524
4	4	3225597	51482970	0.06265367
5	4	10310496	164745504	0.062584385
6	4	25765792	411863760	0.062559017
7	4	51516520	823727520	0.062540729
8	4	83705798	1338557220	0.062534344
9	4	111597612	1784742960	0.062528675
5	5	30914424	494236512	0.062549859
6	5	72116072	1153218528	0.062534611
7	5	133910896	2141691552	0.062525762
8	5	200852988	3212537328	0.062521604
9	5	245479248	3926434512	0.062519634
6	6	156229360	2498640144	0.062525754
7	6	267793288	4283383104	0.062519107
8	6	368205516	5889651768	0.062517366
9	6	409106740	6544057520	0.062515762
7	7	420793096	6731030592	0.062515404
8	7	525919400	8413788240	0.06250685
9	7	525919400	8413788240	0.06250685

分類的目的是為了清楚有哪幾種相似運算會轉回原本的盤面，在進行相似盤面比較時，只需要額外確認分類存在的相似運算是不是會產生相似盤面。改進之後的相似盤面比較，在大部分盤面的情況下，只需要重排盤面一次，而需要重排

16 次的盤面只有 19638 個盤面。將各個資料庫經過旋轉及對稱運算之後，縮小的比例製成表二。

經由旋轉及對稱運算的幫助使得需要用到的記憶體空間幾乎減少為原本的十六分之一，大部分的盤面只看最內圈正方形和最外圈正方形就足夠決定旋轉及對稱的方式，因此使用旋轉及對稱運算能提昇程式執行的速度。

第五章、 實驗成果

第一節、 實驗設計

台灣直棋的規定是一開始每人手中各擁有 12 個棋子，為了測試運算結果的正確性，我們稍微修改了此規則。將台灣直棋修改成三種版本的遊戲，分別為一開始每人六顆棋子、九顆棋子及十二顆棋子。對於放子階段產生的資料庫，因為玩家手上的棋子數不同，造成三種版本放子階段佔用的回合數也不同，因此在放子階段三種版本之間是不能共用相同資料庫，但對於行子階段的資料庫，卻因為三種版本都是放完手上的棋子才開始行子階段，所以能夠使用相同資料庫。

因此可以在計算台灣直棋遊戲行子階段的殘局資料庫途中，從已完成的資料庫裡面拿出每人六個棋子的台灣直棋會用到的部分，另外開始計算每人六個棋子台灣直棋的勝負問題。並且以每人拿六顆棋子台灣直棋的結果驗證資料庫的正確性；以每人拿九顆棋子台灣直棋的結果來確定是此作法是否能夠讓電腦計算十二顆棋子台灣直棋。

第二節、 六子台灣直棋

我們使用 CPU 規格為 Intel Xeon E5520 2.27GHz(雙處理器)，記憶體總量為 36 GByte 的機器，在求解每人各拿六顆棋子時，因為盤面數量不多，只花了接近 24 小時就完成了兩個階段資料庫的建立。

台灣直棋遊戲在開始盤面時因為旋轉及對稱的關係只有四個不同的位置可以選，而第二步只有十四個位置可以選。

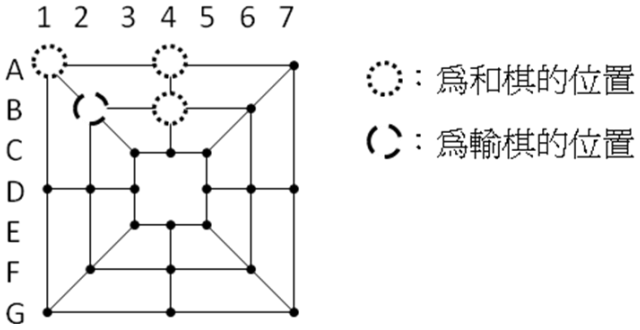
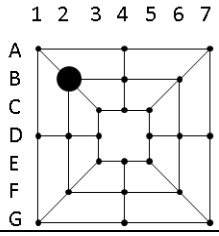
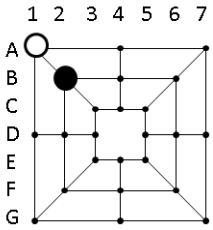
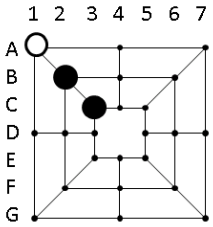
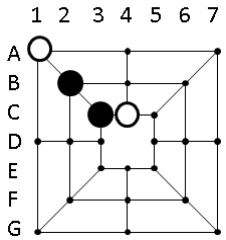
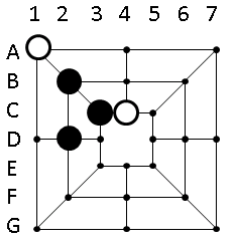
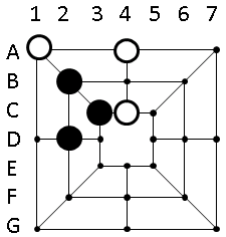
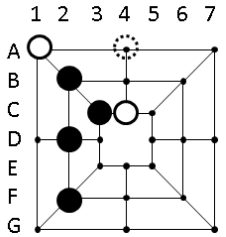
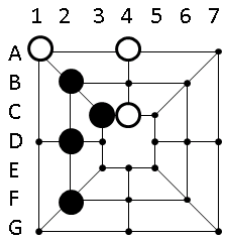


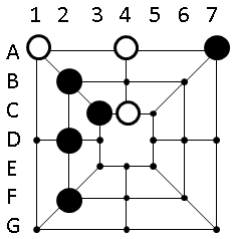
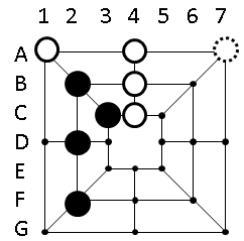
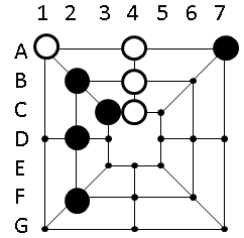
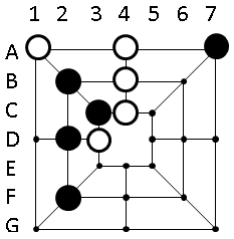
圖5-1　每人六個棋子的台灣直棋遊戲初始選擇

以座標表示棋子位置，行用英文字母表示；列用阿拉伯數字表示，如果黑方玩家選擇 B2 點放置黑色棋子，白方玩家就會得到勝利，表三為其中一條路徑展示。

表三　　六子台灣直棋後手勝利的其中一條路徑

回合	盤面	敘述
1		黑方挑選 B2 放置棋子，以下展示白方勝利的其中一條路徑。
2		白方放置棋子於 A1 會贏，放置棋子於 A4 會和棋，其它位置則會輸。

3		白方已威脅黑方於 B2 上的棋子，因此在第三回合，黑方選擇 C3 的點解除威脅。
4		如果白方玩家將棋子放置在 A4 想要作「直」的情況則結果為和棋，如果放置於其它位置的話則會輸，C4 位置反而是唯一能造成白方勝利的位置。
5		黑方能作「直」的有 B4、B5、D2、D3、E3、F2 六個點，其中 B4、B5、E3、F2 的位置在下一回合都會被白方提掉，因此保守的選擇是 D2、D3，在此我們選擇 D2。
6		在第六回合時如果白方玩家不將棋子放置於 A4 的位置上，而去抵擋黑方玩家作「直」的情況，則白方玩家反而會「輸」。
7		此回合黑方產生「直」然後吃掉 A4 位置上的白子。
8		如果白方玩家將棋子放置於 A4 以外的位置上就會「輸」。

9		上一回合白方強烈威脅提子，黑方已知至少會被吃掉一個棋子。在此展示黑方將棋子放置於 A7 位置。
10		在第十回合時白方將棋子放置 B4 拿掉 A7 逼黑方玩家再將棋子放置於 A7。
11		黑色玩家不將棋子放置於 A7 擋住白方玩家形成「直」的情況，則在進入行子階段後，白方玩家會先勝利。
12		在此時白方玩家將棋子放置於 D3 位置上，造成黑方玩家在下一次輪到白方玩家移動棋子時勢必會被拿掉至少一個棋子

進入行子階段之後，白方已經準備好要造成「槓」的情況拿掉 D2 的棋子，如果黑方想要阻止「槓」的情況，只剩下將 D2 的棋子移到 D1 的作法。但是如此作的話，白方玩家就可以移動 D3 至 D2 造成「擔」的情況，而拿掉 B2 及 F2 的黑方棋子。因此黑方玩家在下回合必會失去棋子。在第十二回合就可以看出黑方玩家已經沒有勝利的機會，白方玩家擁有的吃子手段不止一種而黑方玩家卻沒有能造成吃子的情況。

第三節、 九子台灣直棋

我們使用 CPU 規格為 Intel Xeon E5520 2.27GHz(雙處理器)，記憶體總量為 36 GByte 的機器。在求解每人各拿九顆棋子總共花費約一個月的時間在建殘局資料庫，其中最大的殘局資料庫為八子對八子的資料庫，約有五億八千多個盤面。台灣直棋遊戲在開始盤面時因為旋轉及對稱的關係只有四個不同的位置可以選，而第二步只有十四個位置可以選。

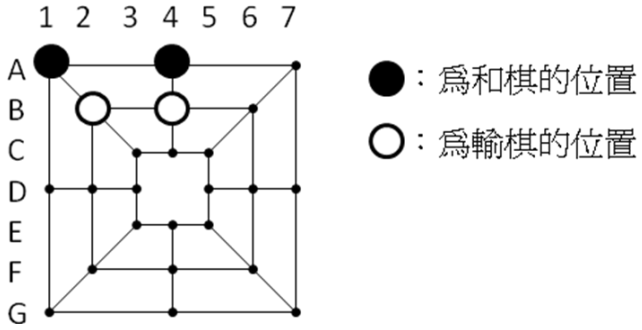


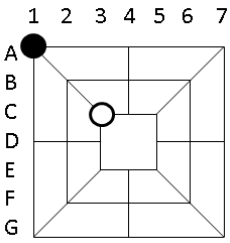
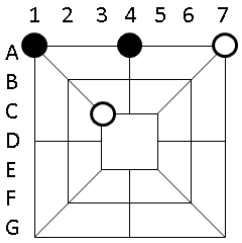
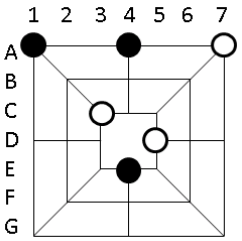
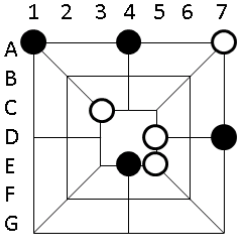
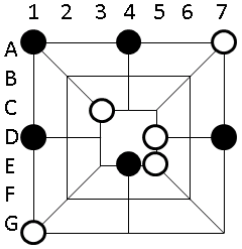
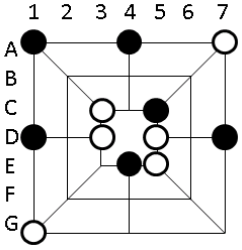
圖5-2　九子台灣直棋遊戲初始選擇

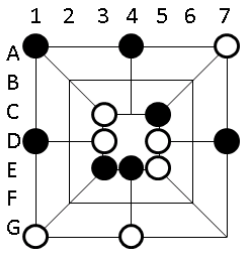
以放子階段第一回合的資料庫來看，可以得知先手有兩個位置會造成輸棋，即是 B1 跟 B4 位置，而且不存在先手必勝的走法。因此從資料庫來看，我們可得知九子台灣直棋的結果為和棋。

放子階段總共產生了 614 個殘局資料庫，行子階段總共產生了 110 個殘局資料庫。表四為先手選擇 A1 放置第一顆棋子和棋的路徑。

表四　　九子台灣直棋和棋的其中一條部分路徑

回合	盤面	敘述

1. A1 2. C3		黑色玩家將棋子放置於 A1。 白色玩家只有一種對應放置位位置為 C3。
3. A4 4. A7		黑方玩家有 A4、D7、G7、C4、C5、D5、B6 跟 F6 的位置可以和棋。若選擇 A4，則白方玩家只一個對應的放置位置為 A7。
5. E4 6. D5		黑方玩家擁有 G7、G5、C5 跟 E4 的位置可以和棋，我們在此選擇 E4。 白方玩家擁有 D7、G4、C4、D5、E5、D3、B6、D6、F6 跟 F2 的位置可以和棋，我們選擇較功擊性較高的 D5 位置。
7. D7 8. E5		黑方玩家擁有 D7、G7 跟 F4 的位置可以和棋，我們在此選擇防禦性較高的 D7。 白方玩家擁有 G7、G4、C4、E5、D3、B6 跟 D6 的位置可以和棋，我們選擇較功擊性較高的 E5 位置。
9. D1 10. G1		黑方玩家只有將棋子放置 D1 才會和棋。 白方玩家卻有很多選擇可以和棋。在此選擇 G1 阻擋黑方吃子。
11. C5 12. D3		黑方玩家此時只剩 C5 才會和棋。 白方玩家卻有很多選擇可以和棋。在此選擇 G1 阻擋黑方吃子。

13. E3 14. G4		黑方玩家只有將棋放置於 E3 位置可以和棋。 白方玩家擁有 G4、B6、D6 及 F6 位置可以和棋。

到十四回合已經可以看出黑白雙方玩家棋子彼此牽制，因為「擔」跟「槓」的關系，限制了棋子的移動，讓黑白雙方很難作出「直」的情況。一旦進入了行子階段則會變成彼此都不想讓對方吃子也無法產生吃子情況，進而變為和棋。

第六章、 結論與未來研究方向

第一節、 結論

本論文提出每人拿六個棋子以及每人拿九個棋子的台灣直棋都為和棋。由於台灣直棋的狀態空間複雜度是 $O(3^{24})\approx O(2^{39})$，我們使用 CPU 規格為 Intel Xeon E5520 2.27GHz(雙處理器)，記憶體總量為 36 GByte 的電腦破解台灣直棋的勝負問題。其中運用了回溯分析演算及殘局庫的技術，針對台灣直棋的特性，提出節省程式運算的時間及記憶體空間的作法。而我們在探索及嘗試各種演算法增進破解台灣直棋程式的速度時，所發現加速旋轉及對稱的方法，或許也能運用在其它相似的棋類遊戲破解之上，從而在電腦博奕領域中貢獻綿薄之力。

第二節、 未來研究方向

未來希望運用本論文的作法破解十二子台灣直棋。十二子台灣直棋的最大殘局資料庫為八子對八子的資料庫，利用九子台灣直棋之行子階段的資料庫可以省去計算的時間。相信本論文的作法是可以應用於十二子台灣直棋遊戲上。經由六子及九子台灣直棋的結果為和棋，預估十二子台灣直棋的結果應該也為和棋。

直棋遊戲流傳於世界各地還有許多變體，每種變體的規則不完全相同，在研究初期希望能找出一套破解所有直棋遊戲變體的演算法，但是因為規則不同的關係，使得我們很難找出能夠套用在所有直棋遊戲變體上的演算法。因此期望能繼續研究破解直棋遊戲的其它變體。

參考著作

[1] Ralph Gasser, "Applying Retrograde Analysis to Nine Men's Morris," Heuristic Programming in Artificial Intelligence; The Second Computer Olympiad, D.N.L. Levy and D.F. Beal (ed.), Ellis Horwood, London, 1991, 161-173.

[2] Ralph Gasser, "Solving Nine Men's Morris," R.J. Nowakowski (Ed.), Games of No Chance. MSRI Publications, Vol. 29, Cambridge University Press, Cambridge, MA, 1996, pp. 101-113.

[3] 白聖群，"八層三角殺棋的勝負問題之研究"，國立臺灣師範大學資訊工程研究所碩士論文，2009。

[4] 黃文樟，"電腦象棋深象中局程式的設計與實作"，國立臺灣師範大學資訊工程研究所碩士論文，2006。

[5] 謝曜安，"電腦暗棋之設計及實作"，國立臺灣師範大學資訊工程研究所碩士論文，2008。

[6] 吳光哲，"電腦象棋搜尋圖歷史交互作用問題之研究"，台灣大學資訊工學研究所碩士論文，2005。

[7] 許舜欽，"直棋的電腦解法及其實現"，國立台灣大學工程學刊，第三十二期，第 1-10 頁，1982。