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

電腦暗棋程式 Observer 的設計與實作

The Design and Implementation of the Chinese Dark Chess Program Observer

研究生：徐大開 撰

中華民國 一百零三 年 七 月

摘要

近幾年來電腦暗棋程式逐漸發展，目前已具有一定程度的棋力。暗棋程式下棋時需要考慮走子和翻子之間的選擇。如果選擇翻子，還要考慮期望值的問題。

本程式會評估各個子力的價值，並使用搜尋尋求一個好的走法，另外有時候要做特殊處理以解決一些盤面上的問題。在翻子的問題上我們除了算出期望值之外，還做了一些額外的調整。

在殘局時由於棋子可走的空間通常較大，使得搜尋時每一層會有較多的步數。當每一層的步數變多時，能搜尋的層數就下降了。如果有殘局庫，程式搜尋的過程在找到殘局庫的內容後可以馬上知道結果，以提高殘局時程式搜尋的效率。我們實作出可以有一個暗子的五子以內的殘局庫，使得程式在殘局的處理上更有方向性。

本程式在 TCGA 2014 的比賽中得到銅牌。希望讀者能夠從本文所使用的方法中發現一些值得參考的地方。

關鍵字：電腦對局、暗棋、不完全資訊遊戲

ABSTRACT

Chinese Dark Chess programs have become more and more powerful for several years. When a program plays the game, it chooses among the normal moves and the revealing moves. If it chooses a revealing move, it should calculate its expectations.

Program finds a good move by evaluating the values of each pieces and by searching. Sometimes it makes special arrangements to deal with some problems on the board. In order to reveal a piece correctly, we did some adjustments besides calculating expectations in this thesis.

There is usually more space in the endgame, thus there are more moves for each side to play and the search space will become larger. If an endgame database exists, the program can find the results directly from the endgame database and then it can improve the efficiency in the endgame. In this study, we made an endgame database for the boards with at most five pieces containing zero or one dark piece. This makes our program more competitive in the endgame.

Our program has won the bronze medal of TCGA 2014. We hope that readers can find some benefits from the methods described in this thesis.

Keywords: Computer Games, Chinese Dark Chess, Imperfect Information Games

目錄

表目錄

圖目錄

第一章 緒論

1.1 研究背景

在電腦發展的領域裡有許多以機器代替人類智力的相關課題,而電腦對局是其中的課題之一。在許多遊戲裡，電腦程式已被拿來和人類的玩家對抗，例如圍棋、象棋、西洋棋、麻將和橋牌等等。西洋棋程式已可以打敗人類棋王；象棋程式已經具備人類頂尖高手的棋力；但是圍棋程式目前還難以和人類高手抗衡。

遊戲可分成完全資訊遊戲（Perfect Information Games）和不完全資訊遊戲（Imperfect Information Games）。前者是玩家可以完全看到遊戲的資訊，例如西洋棋和雙陸棋；後者是玩家不能完全看到遊戲的資訊，例如盲西洋棋和橋牌。這兩類型的遊戲又可以各自分成機率型的遊戲和非機率型的遊戲。雙陸棋和橋牌是機率型的遊戲，而西洋棋和盲西洋棋是非機率型的遊戲。在非機率型的完全公開資訊遊戲裡，電腦程式的實力很大程度上取決於遊戲的複雜度。暗棋屬於機率型不完全資訊遊戲，玩家們難以找到通用的制勝策略取勝。

目前台灣電腦對局學會（Taiwan Computer Game Association, TCGA）、中華民國人工智慧學會（Taiwanese Association for Artificial Intelligence, TAAI）和國際電腦對局學會（International Computer Games Association, ICGA）的電腦奧林匹亞比賽（Computer Olympiad）已經將暗棋列為比賽項目之一。

1.2 研究目的

暗棋是由象棋演變的一種玩法，使用象棋棋盤的一半，共有 32 格，剛好可以放置象棋雙方共 32 個棋子。暗棋的棋子一次只能走一格，可以吃對方的棋子中不大於自己的棋子，但是炮（包）的吃子步是隔一個棋子然後吃對方的棋子。有時候下一盤象棋可以耗費一個小時甚至更久，但是下一盤暗棋很難達到這麼長的時間。暗棋規則簡單，是人們打發時間的一種選擇。象棋是擒王戰，帥（將）的重要性遠大於其它棋子。暗棋是殲滅戰，帥（將）的重要性不像象棋裡那樣重要。另外，暗棋有運氣成分，有時候翻出一個暗子會對盤面產生巨大的變化。就遊戲的本質而言，暗棋和象棋是不同類型的遊戲。

電腦暗棋程式不斷改良，目前還難以證明能和人類高手的對局中能佔到上風。暗棋的對局結果取決於運氣和技術，運氣的部分難以掌握，而技術的部分可以分成翻子和走子兩方面。翻子或走子除了要考慮能不能吃對方的棋子之外，還要考慮能不能保護自己的棋子，或者佔據重要的位置。有時候一個棋子的存活與否，或者是盤面上棋子的佔位，會對局勢造成很大的影響。如果翻子和走子之中有幾個比較好的選擇，這時候要評估它們可能的效益和風險。另外，盤面的優劣程度或多或少會影響到策略的進取程度。

實作電腦暗棋程式的搜尋樹有兩種較常見的方式，一種是使用 Minimax 搜尋樹搭配 Alpha-beta Pruning（本程式使用的方式）或是類似的方式，另外一種則是

使用 MTCS（Monte-Carlo Tree Search）演算法[1]。前者可以找到確定的一組解，而後者則是運用模擬的方式搜尋，每次找出的解未必是相同的。

在沒有殘局庫的情況下，程式要搜尋才能找到對局的最後結果。在雙方剩下最後幾個棋子的時候，程式有可能找不到必勝的步或是沒有避開必輸的步。建立殘局庫可以幫助程式更快找到最後結果，在雙方剩下最後幾個棋子時的走步會變得更精準。

我們的目的是研究出一個具有一定棋力的程式，可以對當前的盤面做出評估，同時具有良好的出步速度。

1.3 研究意義

暗棋和象棋的不同之處之一，在於暗棋有運氣成分。象棋棋力差很多的兩人對弈，棋力較低的那方幾乎沒有勝算。但玩暗棋即使雙方棋力差很多，棋力較低的那方偶爾也能贏。暗棋程式除了在盤面上做出足夠深的搜尋之外，還要考慮暗子的不同種可能，以及在翻開暗子之後對於盤面的影響。

程式在搜尋時要找出所有可能的走步，以及這些走步產生的後續發展。搜索得愈快代表程式可以在有限的時間內搜索得愈深。另外，審局的準確性也是棋力強弱的一大指標。

對於人類來說，翻子和走子之間的選擇可以按照當時的局面來判斷，程度好的棋手也容易做出相對較好的選擇。目前程式和人類的高手比起來可能缺乏足夠

的大局觀，程式較難預測在十幾回合後潛在的機會或威脅。在短兵相接的盤面上程式也許能發揮得不錯，但是程式在某些盤面上的走步可能會缺乏一個好的方向。程式開發者設計的策略使程式可以應付許多盤面，但也可能使得程式在某些情況下走出很差的棋步。如何讓程式的策略不要出現大錯誤並且有良好的棋力是程式開發者得思考的問題。

1.4 論文架構

本論文共有五章。第一章介紹研究的背景、目的和意義。第二章介紹暗棋棋規、相關論文介紹以及近幾年 Computer Olympiad 暗棋項目的比賽結果。第三章介紹本程式使用的資料結構和搜尋演算法。第四章介紹本程式的改良方式，包括棋子的分數設置、預測搜尋時間、殘局庫、翻子策略和其它在盤面上的特殊處理方法。第五章是結論和未來的研究方向。後面還有附錄和參考資料。

第二章 電腦暗棋相關資料

2.1 暗棋棋規介紹

暗棋使用的棋子和象棋一樣。紅、黑雙方各16個棋子，分別有一個帥（將）、兩個仕（士）、兩個相（象）、兩個俥（車）、兩個傌（馬）、兩個炮（包）和五個兵（卒）。棋盤則是使用一半的象棋棋盤。象棋的棋子是佔據棋盤上的點，而暗棋的棋子是佔據棋盤上的格子。一半的象棋棋盤有 32 個格子，剛好可以容納所有暗棋棋子。

一開始棋盤上的棋子都是背面朝上，稱為「暗子」。先手方翻的第一個棋子顏色代表他的陣營，後手方的陣營為另一個顏色。棋子被翻開後會正面朝上，稱為「明子」。雙方輪流下棋，輪到自己時，可以翻一個暗子、移動一個自己的棋子到鄰近一格的空格上或是吃掉對方不小於自己的明子。但是炮（包）的吃子方式是以該棋為出發點，在一條直線上隔著第一個棋子去吃掉第二個棋子，第二個棋子必須是對方的明子。

除了炮（包）之外，暗棋的棋子大小依序為帥（將）＞仕（士）＞相（象）＞俥（車）＞傌（馬）＞兵（卒）。這六個兵種的吃子方式是將自己的棋子移動一格並吃掉對方的棋子。棋子可以吃對方同級的棋子。卒（兵）可以吃對方的帥（將），但是帥（將）不能吃對方的卒（兵）。炮（包）是暗棋的一個特殊兵種，對方除了卒（兵）以外的棋子都能吃它。

當一方吃掉對手的所有棋子或是使對方沒有合法步可走時，該方獲勝，對手

算輸。當雙方都無法達到上述條件時則和棋。

目前電腦暗棋棋規允許長捉對方的棋子，進入循環盤面三次後和棋。人類在下棋時可能會視長捉為禁手。如果電腦暗棋棋規規定長捉為禁手，則程式下棋時不能因為居於下風就長捉對方的棋子求和，如此來看較容易分出勝負。但是在讓程式判斷是否長捉的問題上，程式開發者得寫出較為複雜的程式。允許長捉與否是值得討論的問題。

2.2 相關論文介紹

第一篇論文為「電腦暗棋之設計及實作」[2]，由國立台灣師範大學資訊工程研究所謝曜安於 2008 年 6 月所撰。該篇論文在資料結構上使用棋子映射結構，以一維陣列紀錄棋盤上的位置。程式會用 Transposition Table 記錄之前搜尋過的節點分數，一般是用一個 Key 表示一個盤面。在遊戲策略上以靜態審局函數加上 Nega-max 演算法以及 Alpha-beta Pruning 搜尋樹做搜尋，再加上寧靜搜尋防止水平效應。

第二篇論文為「電腦暗棋程式與經驗法則之配合與實作」[3]，由國立東華大學資訊工程研究所賴學誠於 2008 年 7 月所撰。該篇論文用 Alpha-beta Pruning 和逐層加深（Iterative Deepening）搜尋樹。另外每個棋子都有自己的影響力，影響的程度依棋子之間的距離遞減，反應在分數上。這使得在選擇走步時有一個額外的依據。

第三篇為「暗棋中棋種間食物鏈關係之探討與實作」[4]，由國立台灣師範大學資訊工程研究所謝政孝於 2010 年 6 月所撰。該篇論文改良了[2]的做法，並且使用食物鏈的概念設定棋子的分數。棋子在不同盤面時，它們的價值往往也不一樣。食物鏈的概念使得棋子的分數更為準確，進而加強審局的效果。

第四篇為「電腦暗棋之人工智慧改良」[5]，由國立台灣師範大學資訊工程研究所勞永祥於 2011 年 6 月所撰。該篇論文使用棋形的判別來決定翻子和走子的策略。當我方某個棋子可以吃對方未受保護的另一個棋子時，使用距離分表示威脅程度。棋子之間的距離不是單純地用「曼哈頓距離」（Manhattan Distance）表示，而是根據盤面上對方能吃我方該棋子的棋子或是暗子加以判斷。

第五篇為「電腦暗棋程式 DarkCraft 的設計與實作」[6]，由國立台灣師範大學資訊工程研究所施宣丞於 2012 年 6 月所撰。該篇論文使用 Bitboard 的技術，這是電腦西洋棋已經發展出來的技術。使用 Bitboard 使得走步產生變快，效能有顯著的提升。當搜尋的過程中發現有兩個以上最好的走子步時，還會使用單向搜尋（Oneway Search）找尋好的吃子路徑。

第六篇為「電腦暗棋程式 Darkness 的設計與實作」[7]，由國立台灣師範大學資訊工程研究所詹凱翔於 2013 年 7 月所撰。該篇論文改良了[6]的做法，用了 Transposition Table 搭配 Zobrist Hashing 技術儲存重複盤面，加快程式的搜尋速度。在進入循環盤面且盤面上子力較優時，可能會改變走步以避免和棋。

第七篇為「電腦暗棋殘局庫之研究」[8]，由中原大學應用數學系林庭羽於

2013 年 7 月所撰。該篇論文用回溯分析演算法（Retrograde Analysis）和 DTC（Distance to Conversion）的方式實作出五子以內的殘局庫，殘局庫的棋子都是明子。每種殘局都有對應的檔名，檔案的大小則是依據該殘局的棋子數量而定。殘局庫有做鏡像處理，所需空間為未做鏡像處理的四分之一。

2.3 近年 Computer Olympiad 比賽結果

表 2-1、表 2-2 和表 2-3 為 2010 年、2011 年和 2013 年 Computer Olympiad 比賽結果[9]，2012 年未舉辦 Computer Olympiad。

表 2-1　2010 年 Computer Olympiad 比賽結果

Rank	Program	Country	Title
1	Modark	TWN	Gold medal
2	Dark Chess Beta	TWN	Silver medal
3	Leave-or-Lose	TWN	Bronze medal
4	Flipper	TWN	
4	Dark_chesser	TWN	
4	Sparrow	TWN	

表 2-2　2011 年 Computer Olympiad 比賽結果

Rank	Program	Country	Title
1	DarkCraft	TWN	Gold medal
2	Dark_chesser	TWN	Silver medal
3	Diablo	TWN	Bronze medal
4	TopGun	TWN	

表 2-3　2013 年 Computer Olympiad 比賽結果

Rank	Program	Country	Title
1	DarkKnight	TWN	Gold medal
2	Diablo	TWN	Silver medal
3	Yahari	TWN	Bronze medal
4	Flipper	TWN	
4	Darkness	TWN	
4	Dark_chesser	TWN	
4	Homere	FRA	
4	Tuxedocat	TWN	
4	Martis	TWN	
4	Rainbow	TWN	

第三章 資料結構與搜尋演算法

3.1 Bitboard

傳統的做法是使用一維陣列儲存盤面的資訊。用一個一維陣列儲存盤面上的棋子，然後用另一個一維陣列儲存棋子所在的位置。這樣可以用棋子查位置，也可以用位置查棋子，是一種簡單且實用的方法。檢查棋子時需要搜尋當前盤面所有位置的棋子種類。產生走步時還要搜尋棋子相鄰位置的狀態，然後判斷出能不能走步。

本程式是根據施宣丞的程式 DarkCraft[6]和詹凱翔的程式 Darkness[7]改良而成。根據[6]這篇論文，使用 Bitboard 可將暗棋的所有棋子按照種類或陣營用位元記錄該棋子在棋盤的格子上。當棋子產生走步時可以用邏輯運算產生，這樣可以使得搜尋更為快速。暗棋的棋盤是 4 × 8，共有 32 格，剛好能用 32 位元的整數儲存盤面的資訊。圖 3-1 和圖 3-2 為表示法。

3	2	1	0
7	6	5	4
11	10	9	8
15	14	13	12
19	18	17	16
23	22	21	20
27	26	25	24
31	30	29	28

圖 3-1　32 位元對應位置圖

O	O	O	O
		O	
O			O
	O	O	
O			
	O	O	O
			O

圖 3-2　16 進位表示法 0x1786920F

	傌	卒	
	士		
卒			馬
			包
	帥		

圖 3-3　範例盤面一

O		O	
	O		

圖 3-4　傌所在位置移動範圍的 16 進位表示法 0x4A

		O	
O			O
			O

圖 3-5　馬、包和卒所在位置的 16 進位表示法 0x10902

		O	

圖 3-6　生成傌吃子步的 16 進位表示法 0x2

如圖 3-3，要生成紅傌的吃子步，我們需要知道紅傌的位置。如果有兩個紅傌，要對它們所在的位置分別計算吃子步。圖 3-3 只有一個紅傌，位置為 0x4。然後找出這個位置在周圍都沒有棋子的狀況下，可以走的所有步是 0x4A，如圖

3-4 所示。在紅傌的周圍有棋子時，紅傌只能走到黑方馬、包和卒的位置上並吃掉它。黑馬的位置是 0x100；黑包的位置是 0x10000；黑卒的位置是 0x802。我們先把黑方馬、包和卒的位置做 OR 運算，得到 0x10902，如圖 3-5 所示。然後把 0x4A 和 0x802 做 AND 運算，得到 0x2 為紅傌的吃子步，如圖 3-6 所示。

3.2 Transposition Table

Transposition Table 是用來儲存已被搜尋過的盤面和該盤面的分數。當程式搜尋到相同盤面時就可以直接用 Transposition Table 裡的資料而不用重複搜尋。

傳統的做法是將被搜尋過的盤面資訊全部存到 Transposition Table 裡。但是這樣做會佔很多空間而且效能較差，所以我們用 Zobrist Hashing 的技術配合 Transposition Table[7]。Zobrist Hashing 是將一些亂數用 XOR 邏輯運算組合成一個新的亂數以表示一個盤面。

暗棋的盤面上有 15 種棋子狀態，有紅、黑雙方各七種棋子加上暗子。我們需要 15× 32 = 480 種亂數基底作為一個亂數表。該亂數表為二維陣列，表示法為 Array[棋種(0~14)][位置(0~31)]。棋種用 0 表示暗子，1 到 7 分別表示紅方的帥、仕、相、俥、傌、炮和兵，8 到 14 分別表示黑方的將、士、象、車、馬、包和卒。如果要把兵放置到 20 號位置，則需要將當前盤面的亂數和 Array[7][20]做 XOR 運算。這樣做速度會比較快。圖 3-7 該盤面 Zobrist Hashing 的值即為 Array [0][14] ⊕Array[7][6]⊕Array[7][20]。

		兵	
	■		
			兵

圖 3-7　範例盤面二

但是在 Transposition Table 裡如果不同盤面的 Zobrist Hashing 的值一樣，就可能有碰撞的問題。我們使用 64 個位元的 Zobrist Hashing Key 表示一個盤面，其中較高的 28 個位元作為 Transposition Table 的 Index，而較低的 36 個位元作為驗證碼。這樣 Collision Rate 會變得很低。

3.3 搜尋演算法

一個較簡單的搜尋演算法是 Minimax 搜尋演算法，如圖 3-8 所示[10]，可以看成一個搜尋樹。圖中的圓形節點為 Max 節點；而正方形節點為 Min 節點。搜尋到底層時會透過審局函數回傳一個值。Max 節點的第一個子節點回傳時，它會先記錄第一個子節點的分數，然後往下一個子節點搜尋。下一個子節點回傳的分數如果比該 Max 節點的分數高，那麼該 Max 節點會記錄這個回傳的分數。之後再往下一個子節點做相同動作直到沒有子節點為止。Max 節點會找尋所有的子節點，但只會記錄它的子節點中最大的分數。同樣，Min 節點只會記錄它的子節點

中最小的分數，其它的動作和 Max 節點相同。

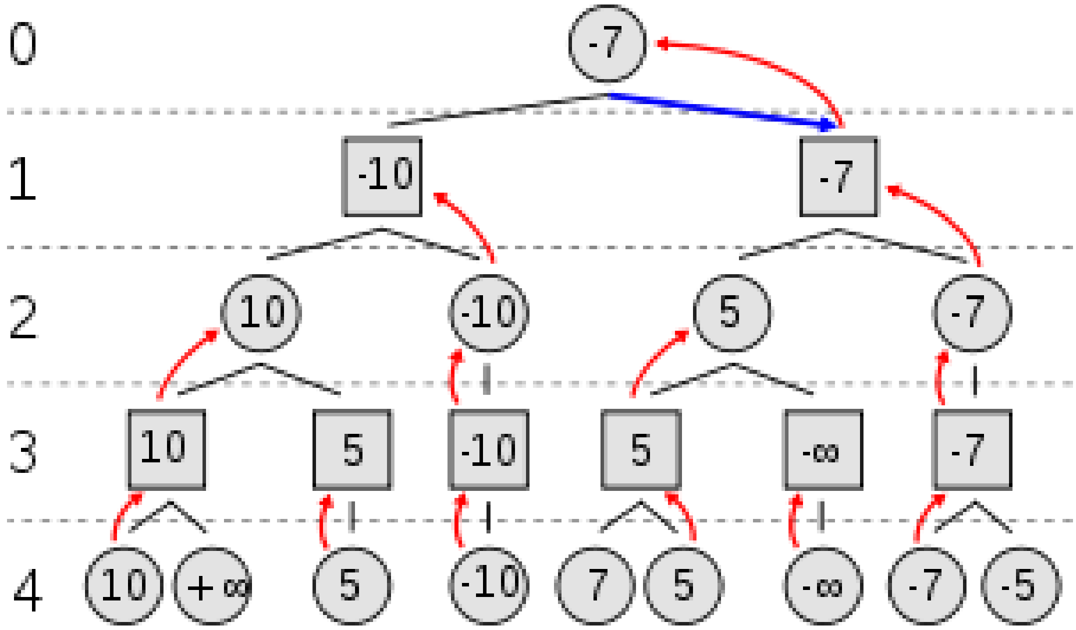


圖 3-8　Minimax 搜尋演算法的例子

圖 3-9 為 Minimax 搜尋演算法的虛擬碼[10]。

```
function minimax(node, depth, maximizingPlayer)
    if depth = 0 or node is a terminal node
        return the heuristic value of node
    if maximizingPlayer
        bestValue := -∞
        for each child of node
            val := minimax(child, depth - 1, FALSE)
            bestValue := max(bestValue, val)
        return bestValue
    else
        bestValue := +∞
        for each child of node
            val := minimax(child, depth - 1, TRUE)
            bestValue := min(bestValue, val)
        return bestValue
```

圖 3-9　Minimax 搜尋演算法的虛擬碼

Minimax 演算法可以完整地搜尋預設層數內的所有子節點。搜尋樹的節點數

是呈指數成長的。假設要搜尋 n 層，而每個節點各有 k 個分支，則要搜尋 k^n 個葉節點。如果雙方每步都有 20 種走步可走，那麼要加快約 20 倍才能在原來的基礎上多搜一層。但是有些節點可能是無用的，搜尋所有節點往往會浪費效能。

Alpha-beta Pruning 是 Minimax 搜尋演算法的改良。McCarthy John 在 1955 年提出了這個想法，後來 Alexander Brudno 在 1963 年發表[11]。Alpha-beta Pruning 的概念是當一個節點將要找尋的所有子節點都不能影響父節點的分數時，這些子節點就沒有搜尋的必要。

Alpha-beta Pruning 會用到參數 α 和 β，每個節點都有自己的 α 值和 β 值，在搜尋的過程中由父節點傳過來。α 表示目前搜尋路徑中 Max 節點的最大值，而 β 表示目前搜尋路徑中 Min 節點的最小值。Max 節點在更新自己的值後如果發現 α 值比自己的值小，會將 α 值變成自己的值。Min 節點在更新自己的值後如果發現 β 值比自己的值大，也會將 β 值變成自己的值。之後往下搜尋會將自己的 α 值和 β 值傳下去。要注意的是，Max 節點不會更新自己的 β 值；而 Min 節點也不會更新自己的 α 值。

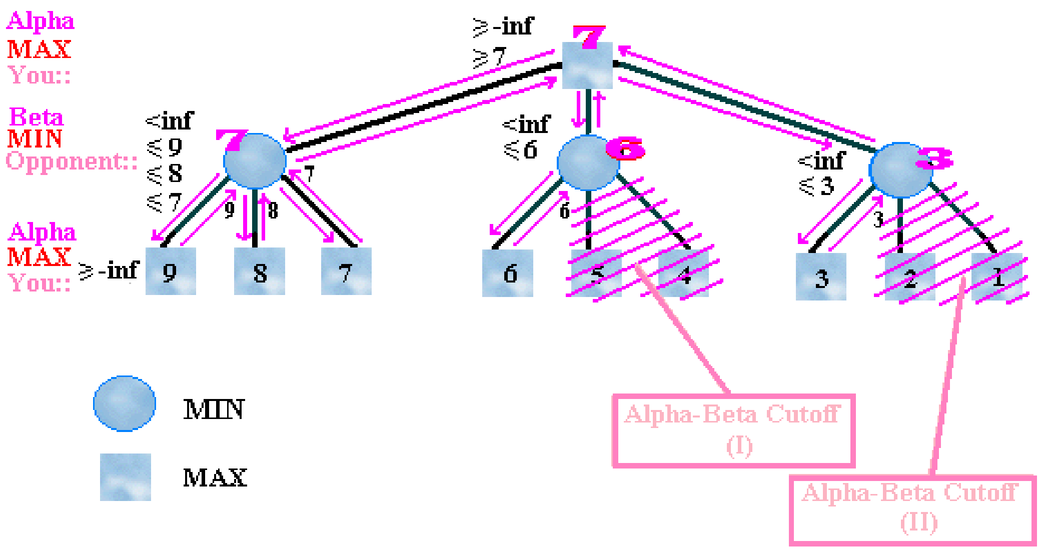


圖 3-10　Alpha-beta Pruning 演算法的例子[11]

如圖 3-10，正方形節點為 Max 節點；而圓形節點為 Min 節點。根節點的第一個子節點在搜尋的過程中不斷更新自己的值和 β 值，最後這兩個值都是 7。根節點收到回傳值後，根節點會更新自己的值和 α 值，這兩個值也都是 7，再將自己的 α 值和 β 值傳到第二個子節點。該節點搜尋到第一個子節點的回傳值是 6，小於自己的 α 值，表示再怎麼往下搜尋，根節點的分數也不會有任何改變，不需要再往下搜尋。這種情形叫做 Alpha-cut。之後根節點的第三個節點也同樣發生 Alpha-cut。Min 節點發現自己的值小於 α 值會發生 Alpha-cut；而 Max 節點發現自己的值大於 β 值則會發生 Beta-cut。

圖 3-11 為 Alpha-beta Pruning 演算法的虛擬碼[10]。

```
function alphabeta(node, depth, α, β, maximizingPlayer)
      if depth = 0 or node is a terminal node
          return the heuristic value of node
      if maximizingPlayer
          for each child of node
              α := max(α, alphabeta(child, depth - 1, α, β, FALSE))
              if β ≤ α
                  break (* β cut-off *)
          return α
      else
          for each child of node
              β := min(β, alphabeta(child, depth - 1, α, β, TRUE))
              if β ≤ α
                  break (* α cut-off *)
          return β
```

圖 3-11　Alpha-beta Pruning 演算法的虛擬碼

Nega-max 搜尋演算法可以用來簡化 Minimax 演算法，1975 年由 Knuth 和 Moore 提出[12]。Nega-max 的概念是自己得到的分數，傳給對手就加一個負號。圖 3-12 為 Nega-max 搜尋演算法的虛擬碼[10]。

```
function negamax(node, depth, color)
    if depth = 0 or node is a terminal node
        return color * the heuristic value of node
    bestValue := -∞
    foreach child of node
        val := -negamax(child, depth - 1, -color)
        bestValue := max( bestValue, val )
    return bestValue
```

圖 3-12　Nega-max 搜尋演算法的虛擬碼

後來還有 Alpha-beta Pruning 和 Nega-max 結合的版本，為 Nega-max with

Alpha-beta Pruning 搜尋演算法。圖 3-13 為兩者結合版本的虛擬碼[10]。

```
function negamax(node, depth, α, β, color)
    if depth = 0 or node is a terminal node
        return color * the heuristic value of node
    bestValue := -∞
    childNodes := GenerateMoves(node)
    childNodes := OrderMoves(childNodes)
    foreach child in childNodes
        val := -negamax(child, depth - 1, -β, -α, -color)
        bestValue := max( bestValue, val )
        α := max( α, val )
        if α ≥ β
            break
    return bestValue
```

圖 3-13　Nega-max with Alpha-beta Pruning 搜尋演算法的虛擬碼

如果設定固定層數做一次性的搜尋，那麼當盤面分支度很小時，可能很快搜完而不會繼續加深搜尋。當盤面分支度很大時，可能展開不完整個搜尋樹，導致浪費時間的情況。使用逐層加深(Iterative Deepening)搜尋的方法可以解決這些問題。它可以先展開一個深度很淺的搜尋樹，如果還有剩餘時間，還可以展開深度較上一次深的搜尋樹，直到用完時間或展開完預設層數的搜尋樹。

吃子可能會導致盤面的劇烈變化。如果程式搜到預設層數後就不再搜尋，那麼當葉節點的子節點有吃子步時，程式就很難發現盤面在預設層數後會有劇烈變化。使用寧靜搜尋(Quiescence Search)可以解決這個問題。它會搜尋葉節點子節點的所有吃子步節點，然後一直往下搜尋吃子步，直到沒有吃子步可搜尋為止。這樣做會多花一些時間，但是可以更準確地比較出各個走步之間的好壞。

第四章 改良方法

4.1 棋子的分數設置

審局函數對於棋力影響很大，好的審局函數可以幫助程式找到潛在的機會或威脅，不好的審局函數會使得程式在搜尋的過程中引導程式進入錯誤的方向裡。在相同的時間裡，程式處理審局函數的時間會影響搜尋的速度。簡單的審局函數可以讓程式搜得較深，但審局可能會有缺陷。複雜精巧的審局函數可以讓程式對於當前的盤面分析得較為準確，但也可能使得程式的搜尋速度下降。

舊版的程式 DarkCraft[6]和 Darkness[7]是用靜態的分數設置，但是 Darkness 會在對方帥(將)還存活的情況下依據我方兵（卒）的數量調整它們的分數。

傳統的做法是將所有棋子各自用一個分數表示它們的重要性。這個分數是固定的，從遊戲開始到最後都不會改變。通常會把帥（將）的分數設定為最高，然後依序是仕（士）、相（象）、俥（車）、傌（馬）和兵（卒）。至於炮（包）的分數，通常是設定為仕（士）和相（象）之間，或者是設定為和相（象）相當。

審局函數是靜態的，棋子的分數是固定的。這樣的審局函數較為簡單。但是在不同盤面裡棋子的價值往往不同。這樣做的缺點是如果當我方的帥（將）存活且對方的卒（兵）只剩一個時，那麼程式就不會用傌（馬）去兌對方最後一個卒（兵）了。但通常兌掉是值得的，因為兌掉後我方的帥（將）就沒有天敵了，而犧牲一個傌（馬）通常不會對我方產生巨大的影響。如圖 4-1，用紅傌兌黑卒會

和棋，否則黑方逃卒之後紅方難以守和。

傌	卒		
	士	象	
	帥		

圖 4-1　可用傌兌卒的盤面(輸紅方)

我們的做法是偵測根節點的盤面上紅方的帥和黑方一個或以上的卒是否一起存活，或者是黑方的將和紅方一個或以上的兵是否一起存活。如果其中有一個條件達成，則程式的分數在搜尋的過程中會進入動態設置模式，否則為靜態設置模式。此時帥（將）的分數最高，仕（士）、相（象）、俥（車）和傌（馬）的價值依序降低，兵（卒）的分數最低，炮（包）的分數則介於仕（士）和相（象）之間。

在圖 4-1 的盤面裡紅方的帥和黑方的一個卒還存活著，在搜尋過程中棋子的分數會進入動態設置模式。圖 4-2 和圖 4-3 的盤面裡沒有滿足動態設置分數的條件，在搜尋過程中棋子的分數會進入靜態設置模式。

		馬	
			將
	帥		
		傌	

圖 4-2　沒有滿足動態設置分數條件的盤面一

		馬	
	士		
	帥		傌
		兵	

圖 4-3　沒有滿足動態設置分數條件的盤面二

當程式進入靜態設置模式時，分數的設置如表 4-1。

表 4-1　靜態設置模式子力分數表

棋種	分數
帥（將）	11000
仕（士）	4900
相（象）	2400
俥（車）	1100
傌（馬）	500
炮（包）	3000
兵（卒）	50

除了炮（包）的棋種，原則上我們把每個棋種的子力價值設定為它下一級的兩倍再多一些，而傌（馬）則是超過兵（卒）的五倍。原本我們設定帥（將）的價值為 10000 分。後來考量到沒有天敵的棋子的重要性可能更高，我們增加到 11000 分。

我們把炮（包）設定為 3000 分。炮（包）在暗棋裡面是一個特殊的兵種，可以隔一個子去吃對方的任何一子。因此，炮（包）在有自己大子的掩護下往往有強大的力量。另外，有時候炮（包）可以在遠距離保護自己的子不讓對方吃掉。所以炮（包）的價值很高。

當程式的分數進入動態設置模式時，除了炮（包）的棋子分數根據盤面上的棋子會被炮（包）以外的多少個棋子吃掉，而給定一個分數。棋子的天敵愈少，分數就愈高。

在此模式時，仕（士）、相（象）、俥（車）、傌（馬）和兵（卒）分數的設定如下：

Score[天敵(0~8)] = {11000, 4900, 2400, 1100, 500, 240, 110, 50, 20}

炮（包）仍然設定為 3000 分。

我們決定讓帥（將）的分數的增幅和一般的棋子不同。帥（將）在暗棋中有較為獨特的地位，帥（將）站在好位置時可以限制對方士（仕）和象（相）的移動。

帥（將）的分數的設定如下：

Score[天敵(0~5)] = {11000, 6000, 4000, 3000, 2300, 2000}

棋局剛開始時，程式的分數一定會進入動態設置模式，此時各個棋種的分數如表 4-2。

表 4-2　棋局剛開始的子力分數表

棋種	分數
帥（將）	2000
仕（士）	4900
相（象）	1100
俥（車）	240
傌（馬）	50
炮（包）	3000
兵（卒）	20

棋局剛開始時，仕（士）只有對方的將（帥）一個天敵，這時候仕（士）的分數是 4900 分。如果對方的將（帥）被吃了，那麼我方的仕（士）會變成 11000 分。

天敵每少一個，棋子的分數會變成原來的兩倍再多一些。我們可以用下面兩張圖表示紅方相的分數差異：

如圖 4-4，此時紅方相的分數為 2400 分，仕的分數為 4900 分，兩個相的分數加起來會略低於一個仕。如圖 4-5，紅方仕和相的分數相當，因為它們都只有一個天敵。

馬			
	士		將
	帥		仕
相		相	

圖 4-4　仕的分數大於兩個相

馬			
			將
	帥		仕
相		相	

圖 4-5　仕的分數等於相

當對方的一個子被吃時，除了扣掉對方那個棋子的價值外，還會增加我方會被那個子吃掉的棋子的分數。例如，當對方的一個士（仕）被吃掉時，除了扣掉對方的士（仕）的分數外，我方的相（象）、俥（車）、傌（馬）和兵（卒）都會加分。這樣，對方的一個棋子被吃時除了影響對方的分數之外，往往還會增加我方的分數。

		象	
	士	將	
	帥		
			仕
		相	

圖 4-6　用帥兌士會吃虧（輸紅方）

		卒	
	士	卒	
	帥		
			仕
		相	

圖 4-7　用帥兌士會得利（輸紅方）

圖 4-6 的盤面裡紅帥沒有天敵，分數為 11000 分。黑士有一個天敵，分數為

4900 分。雙方的相（象）各有兩個天敵，分數為 2400 分。紅方用帥兌士後，雙方的相（象）各有一個天敵，分數變成 4900 分，紅方會吃虧。但是圖 4-7 的盤面裡紅帥有兩個天敵，分數為 4000 分。黑士有一個天敵，分數為 4900 分。紅相有一個天敵，分數為 4900 分。紅方用帥兌士後，紅相沒有天敵，分數變成 11000 分，紅方會得利。

圖 4-8　TCGA 2014 實戰盤面一

圖 4-8 是 TCGA 2014 本程式與 SGTDark 的一個實戰盤面。此盤本程式執紅方下棋。這個盤面輪到紅方，程式用紅帥兌黑士，之後再用紅炮吃黑象。此後黑方的將、士和象都被吃光，紅方的相、俥、傌和兵都沒有天敵，紅方可勝。

圖 4-9　TCGA 2014 實戰盤面二

圖 4-9 是 TCGA 2014 本程式與 Flipper 的一個實戰盤面。此盤本程式執黑方下棋。這個盤面輪到黑方，程式沒有逃將，而是用黑包吃紅仕。在紅兵吃黑將之後再用黑士吃紅俥。雖然黑方損失黑將，但暗子裡的四個黑卒都沒有天敵了，翻開 4 個黑卒後黑方可勝。

圖 4-10 的盤面裡紅帥和紅兵各有兩個天敵，分數都為 2400 分。黑方的車和馬各有一個天敵，分數都為 4900 分。黑卒沒有天敵，分數為 11000 分。這個盤面裡黑方有優勢，而黑卒的價值是相對較高的。

	兵		兵
		帥	
車		馬	
	卒		卒

圖 4-10　此時卒的價值相對較高

當一方某一個棋種（炮（包）除外）的棋子數量比對方多，且對方還有該棋種的棋子時，我方該棋種的分數會下降三分。這是為了讓兩個以上相同棋種的棋子能去兌掉對方數量較少的同級棋子，這樣對優勢方會有好處。

			車
		士	車
	仕		
		仕	
俥			
俥			

圖 4-11　紅方可用仕兌士

如圖 4-11，紅方取勝的途徑是和黑方兌士，然後再各個擊破。如果把紅仕的分數設定為和黑士一樣，則程式當紅方時難以找到制勝的途徑。把紅仕的分數設

定為黑士的分數再減去三分，就可以避免這種問題。

俥			士
	馬		
相	仕		
仕			車
俥			
			士
			兵

圖 4-12　用紅仕吃黑馬會得利（輪紅方）

把子力分數在任何情況下都設為動態設置模式是否可行？如圖 4-12 的盤面，雙方的帥（將）都已陣亡，沒有滿足動態設置模式的條件。進入靜態設置模式後，紅兵的分數為 50 分，黑馬的分數為 500 分。輪紅方下棋時程式會用紅仕吃黑馬。如果在這個盤面使用動態設置模式，紅兵有四個天敵，分數為 500 分。黑馬有五個天敵，分數為 240 分。此時因為紅兵的分數較高，輪紅方下棋時將會逃兵而不用紅仕吃黑馬。雖然紅兵的天敵較少，實際上它的價值卻低於黑馬，在這個盤面裡用靜態設置模式較佳。

4.2 先吃較重要的子

在搜尋的過程裡，有時候會經過不同的順序，然後走到相同的盤面。這些順序可能包含好幾個吃子步。但是，用搜尋樹搜尋時往往沒有考慮到中間有翻子的

變化，以致於先吃掉某個子時，對方突然翻了一個暗子影響了局勢。為了避免在吃掉一個較不重要的子之後，對方翻子導致對方較重要的棋子存活或者我方必須立刻保護自己的棋子的情況發生。在吃兩個以上棋子的總步數差距不大的情況下，我們往往會優先吃掉對方較重要的棋子，即使之後會略慢吃掉對方較不重要的棋子。

程式搜尋到吃子時會額外加分，增加的分數為(預設搜尋層數 – 當前搜尋層數 + 1) × (1 + 被吃子在根節點的分數 / 1000)。

■	■	■	■
■	仕	■	■
■		士	■
■			■
■		帥	■
■		馬	■
■	俥	■	■
■	■	■	■

圖 4-13　可用帥先去吃士再回來吃馬

（輪紅方）

在圖 4-13 的盤面裡搜尋時，紅帥可以在第一層先吃黑馬並在第七層再吃黑士，或者是在第三層先吃黑士並在第九層再吃黑馬。雖然選擇前者吃子較快，但是給了黑方翻兩個暗子保護黑士的機會。選擇後者黑方只有翻一個棋子的機會保護黑士。目前的分數為動態設置模式。假設預設搜尋層數是 10 層。此時黑士在有一個天敵為紅帥，分數為 4900 分。黑馬至少有三個天敵，除了紅帥、紅仕和

紅俥各一個之外，暗子裡面很可能還有黑馬的天敵。再假設黑馬只有三個天敵，分數為 1100 分。紅帥先吃黑馬再吃黑士的這種加分為$(10 - 1 + 1) \times (1 + \lfloor 1100 / 1000 \rfloor) + (10 - 7 + 1) \times (1 + \lfloor 4900 / 1000 \rfloor) = 40$ 分；而紅帥先吃黑士再吃黑馬的這種加分為$(10 - 3 + 1) \times (1 + \lfloor 4900 / 1000 \rfloor) + (10 - 9 + 1) \times (1 + \lfloor 1100 / 1000 \rfloor) = 44$ 分。此例中舊版的程式 DarkCraft[6]和 Darkness[7]會用紅帥先吃黑馬再吃黑士，本程式則會用紅帥先吃黑士再吃黑馬。

4.3 預測搜尋時間

在逐層加深搜尋時，有時候會出現搜到某一層時，在還沒有搜尋完成的情況下因為預設的時間已到而被迫中斷搜尋。這樣搜尋最後一層的時間就白白耗費掉了，而這層的時間往往是最久的。

本程式會從兩層開始逐層加深搜尋，每次加深兩層。從搜尋四層開始，在每次加深搜尋完之後，我們會根據這兩層所花的時間和前兩層所花的時間之比值，再根據這層的時間和剩餘的時間，來決定要不要搜尋後兩層。如果不繼續搜尋下去，則中斷搜尋然後得到一個結果。這樣可以節省一些搜尋的時間，使得有更多的時間可以做之後盤面的搜尋。當搜尋時間和預測的時間總和超過預設時間時會停止搜尋。

在逐層加深的過程中除了做走子搜尋之外，還會搜尋第一層是翻子後走子的期望值，我們稱為翻子搜尋。翻子搜尋會產生許多分支，在相同的層數下搜尋時

間會比走子搜尋多得多。我們會先做這兩種搜尋到達某種層數，如果搜尋的時間加上預測加深兩層的搜尋時間多餘預設搜尋時間的四分之一，則接下來只做走子搜尋。這樣走子搜尋時間還很充裕，可做進一步的搜尋。

假設預設搜尋時間是 20 秒，一開始程式會做走子搜尋和翻子搜尋。如果搜尋兩層的時間是半秒，搜尋四層的時間是兩秒，則在搜尋四層後會預測搜尋六層的時間是 $2 \times (2 / 0.5) = 8$ 秒。此時搜尋的時間加上預測加深兩層的搜尋時間為 $0.5 + 2 + 6 = 8.5$ 秒，超過 $20 \times (1 / 4) = 5$ 秒。從六層之後，程式就不再做翻子搜尋的部分，而只做走子搜尋的部分。當搜尋時間和預測的時間總和超過 20 秒時，程式會停止搜尋。

4.4 走步排序

搜尋時為了儘快砍掉沒有用的分支來加快搜尋速度，做走步排序是有用的。如果能先搜尋到最好的節點，那麼之後搜尋到較差的節點時就可以較快地砍掉節點。我們的做法是根據上一次的逐層加深搜尋樹來看在這一次的逐層加深搜尋樹應該先搜尋哪個節點，這樣較容易快速找到好的分支來加快搜尋速度。

本程式做逐層加深搜尋會每次會加深兩層，所以做八層搜尋的前六層時，會在產生走步的過程中到 Transposition Table 裡，找做六層的搜尋時這個節點的分數來排序，分數愈高的就排愈前面。如果上一輪的這個節點因為 Beta-cut 而沒有被搜尋到時，就把這個走步排到最後。

傌			將
		帥	
	馬		
			兵
		相	
	車		士

圖 4-14　範例盤面三

如圖 4-14，如果輪紅方，我們生成走步是由紅帥先開始的，但走紅兵可以在搜尋四層再進入寧靜搜尋發現能吃黑將。使用走步排序後，搜尋兩層時會先搜尋紅帥的走步。但程式會逐層加深搜尋，從搜尋六層開始，會先搜尋紅兵往上走的棋步。

4.5 依棋子的自由程度加分

如果有一方的棋子能走的位置較多，那麼通常那一方的活動空間也較大。在搜尋時我們考慮倒數兩層(不含寧靜搜尋)的分支數，給每個分支加 50 分，讓程式在搜尋時較容易擴展我方的活動空間，或是壓縮對方的活動空間。

如圖 4-15，如果輪紅方，程式會讓紅仕往下走去「壓著」黑士，而不會漫無目的的走其它棋步。

■	■	■	■
■	■	■	■
■		■	
■		仕	
■	兵		士
■	■	■	■
■	■	■	■

圖 4-15　範例盤面四

4.6 殘局庫

在沒有殘局庫時，程式進入殘局往往沒有很好的方向感去處理目前的盤面。因此殘局庫可以幫助程式在雙方剩最後數子時發現盤面的勝負狀況。我們的方法是做出可以有一個暗子的五子以內的殘局庫。

殘局庫的生成是用回溯分析演算法(Retrograde Analysis)。它的做法如下：

1. 找出已經分勝負的盤面，並給一個代表贏或輸的分數。
2. 搜尋未知結果的盤面，並找出它的所有走步的子盤面。如果子盤面中有一個盤面是贏棋，則將此未知結果盤面設為贏棋。如果子盤面中所有的盤面都是輸棋，則將此未知結果盤面設為輸棋。
3. 不斷重複步驟 2 直到所有未知結果的盤面都無法更新。此時將所有未知的盤面設為和棋。

對於分出勝負的盤面，除了設定贏棋或輸棋之外，還要設定走步的步數。走步的步數可分為 DTM(Distance to Mate)和 DTC(Distance to Conversion)兩種。DTM 記錄的是距離贏或輸的步數；而 DTC 記錄的是距離吃子或翻子的步數。

我們用 DTM 的方式記錄殘局庫的分數，每個 Index 用兩個位元儲存當前盤面的分數。

我們搜尋雙方在二到五子時的所有殘局組合，然後扣除已經分出勝負的組合，例如紅方兩個兵對黑方沒有棋子的情況。最後給剩下來的組合做編號。

我們統計出從兩子到五子的殘局種類數量，如表 4-3。

表 4-3　兩子到五子的殘局種類數量

殘局棋子數量	種類數量
2	103
3	522
4	2031
5	5432

在計算殘局庫的組合時，我們根據紅方的帥、仕、相、俥、傌、炮和兵，再來是黑方的將、士、象、車、馬、包和卒，排序一個殘局組合的所有棋子。例如「俥、兵、將、士」就不會表示成「將、士、俥、兵」。以四子殘局為例，會依序由「帥、仕、仕、將」找到「兵、卒、卒、卒」。

		帥	
			仕
			卒
象			

圖 4-16　「帥、仕、象、卒」的殘局組合

假設一個殘局組合有 n 個棋子，對於每一個棋子而言，我們用 n 個位數表示這些棋子和其它棋子的關係。如圖 4-16，「帥、仕、象、卒」的殘局組合有四個棋子，每個棋子我們要用四個位數表示它們之間的關係，最高位數可能是 0。由高到低的第 i 個位數就表示這個棋子與第 i 個棋子的關係。然後我們用一個數字表示棋子與第 i 個棋子的關係，如表 4-4。

表 4-4　用一個數字表示殘局組合中棋子之間的關係

數字	棋子和第 i 個棋子的關係
0	該棋子不能吃第 i 個棋子。
1	該棋子是炮(包)，它能吃第 i 個棋子。
2	該棋子不是炮(包)，它和第 i 個棋子同級但不同陣營。
3	該棋子不是炮(包)，它是第 i 個棋子的天敵。

圖 4-16 的盤面裡紅帥只能吃黑象。紅仕可以吃黑象和黑卒，而且不會被反吃。黑象不能吃任何子。黑卒只能吃紅帥。所以帥、仕、象和卒這四個棋子的數字分別為 30、33、0 和 3000。

雙方的某一些組合可以視為同一種殘局組合。如圖 4-17，「帥、仕、象、卒」的殘局組合就和「帥、仕、車、卒」的殘局組合一樣。在搜尋到「帥、仕、車、卒」的殘局組合時，由於這四個棋子的數字分別和「帥、仕、象、卒」一樣，因此「帥、仕、車、卒」只需記錄它可以轉換成「帥、仕、象、卒」即可，不需要生成另一份殘局庫。

		帥	
			仕
			卒
車			

圖 4-17　「帥、仕、車、卒」的殘局組合

		將	
			士
			兵
俥			

圖 4-18　「俥、兵、將、士」的殘局組合

如圖 4-18，「俥、兵、將、士」的殘局組合也可以轉換為「帥、仕、象、卒」的殘局組合。在搜尋到「俥、兵、將、士」時，如果找不到可以轉換的殘局庫，則「俥、兵、將、士」可以暫時轉換成「將、士、俥、兵」，發現這四個棋子的數字分別和「帥、仕、象、卒」一樣。因此「俥、兵、將、士」記錄它可以經由紅、黑棋子互換後，再轉換成「帥、仕、象、卒」即可，不需要生成另一份殘局庫。

如果程式下棋遇到「俥、兵、將、士」的盤面時，會套用「帥、仕、象、卒」

的殘局庫。此時要暫時將棋子紅黑互換並且輪當前走子方的對方走，再套用該殘局庫。

我們統計經過簡化後從兩子到五子的殘局種類數量，如表 4-5。這些殘局的編號和組合放在附錄 A。

表 4-5　經過簡化後兩子到五子的殘局種類數量

殘局棋子數量	種類數量
2	5
3	25
4	98
5	294

我們把每一種組合都用若干個位元儲存盤面的 Index。棋子在盤面上有 32 種可能的位置，所以一個棋子需要五個位元儲存 Index。我們依照棋子的編號依序儲存棋子的 Index。另外，我們還需要用一個位元表示當前盤面是由哪一方下棋。在相同的盤面下，輪紅或輪黑下棋可能有不同的結果。

為了縮小殘局庫所需的空間，可以對盤面做鏡像處理，包含左右對稱、上下對稱和 180 度翻轉[8]。把圖 4-19 作為一個原始盤面，則圖 4-20 是左右對稱盤面，圖 4-21 是上下對稱盤面，圖 4-22 是 180 度翻轉盤面。根據盤面上編號最小的一個棋子判斷是否做鏡像處裡，如此一來這個子只需處理八種可能的位置，用三個位元儲存 Index 即可。

			傌
馬			

圖 4-19　鏡像處理盤面一

傌			
			馬

圖 4-20　鏡像處理盤面二

馬			
			傌

圖 4-21　鏡像處理盤面三

			馬
傌			

圖 4-22　鏡像處理盤面四

綜上所述，在只考慮所有棋子都是明子的狀況下，每種二子殘局組合需要一個位元表示當前盤面由哪一方下棋、三個位元表示第一個棋子的位置(因有鏡像處理)以及五個位元表示第二個棋子的位置，一共有九個位元的 Index。每多一個子的殘局需要多五個位元儲存 Index。這樣，每種五子殘局組合需要 24 個位元的 Index。

這五子除了都是明子的情況外，還可以有一個棋子是暗子的情況。在圖 4-23 的盤面裡，由於只剩一個暗子，所以可以確定那個暗子是什麼棋子，這個盤面也可以被做成殘局庫的一部分。

	包		
		帥	
		■	
			卒

圖 4-23　有一個暗子的殘局

我們需要一些位元記錄沒有暗子或是第幾個棋子是暗子的情況，還要盡量節省空間與方便記錄。因此我們的 Index 編碼方式做了一些調整，分為二到四子以及五子兩種情況。

二到四子殘局的 Index 編碼方式如下(位元由低到高)：

前 18 個記錄棋子的位置。

第 19 個位元記錄輪紅或輪黑。

第 20 到 26 個位元記錄殘局的種類編號。

第 27 到 29 個位元記錄第幾個子是暗子。沒有暗子時設為 000，第一個棋子是暗子時設為 001，以此類推。

經過轉換後，二到四子殘局種類加起來是 5 + 25 + 98 = 128 種，編號是 0 到 127。沒有暗子或第幾個子是暗子的情況最多有五種。前文提到每個 Index 用兩個位元儲存當前盤面的分數。所以所需的空間為 5 × 64 M × 2 bytes = 640 MB = 0.625 GB。

我們需要預留 5 × 64 M = 320 M 的 Index 給二到四子的殘局庫使用，然後五子殘局庫再銜接上去。經過轉換後，五子殘局有 294 種，編號是 128 到 421。

五子殘局的 Index 編碼方式如下(位元由低到高)：

前 23 個記錄棋子的位置。

第 24 個位元記錄輪紅或輪黑。

第 25 到 35 個位元用$(i - 128) \times 6 + j + 20$ 表示殘局的種類編號和第幾個子是暗子。其中 i 為殘局的種類編號，j 為第幾個子是暗子，如果 $j = 0$ 表示沒有暗子。

對於五子殘局庫而言，沒有暗子或第幾個子是暗子的情況有 6 種。所需的空間為 294 × 6 × 16 M × 2 bytes = 55.125 GB。

二到四子與五子殘局合併後，所需的空間為 0.625 GB + 55.125 GB = 55.75 GB。我們實驗室有 64 GB 記憶體的電腦，有足夠的空間將殘局庫全部載入到記憶體裡面。使用殘局庫後，程式的棋力並沒有明顯地增強。有的盤面在中局就結束，而有的盤面即使沒有殘局庫也能靠搜尋獲勝或守和，使用殘局庫得利的情況很少見。

		仕	仕
			卒
將			
	帥		

圖 4-24　五個明子的殘局裡優勢方最長吃子路徑盤面（輪紅方）

另外，根據[8]這篇論文，在五子都是明子的情況下，優勢方最長吃子路徑的盤面如圖 4-24 所示。紅方需要走 89 步，雙方共走 177 步，紅方才能吃黑方的棋子。該篇論文使用 DTC 的方式記錄殘局庫的走步，而我們是用 DTM 的方式記錄。我們發現五子以內可勝的殘局中，優勢方最多需要走 133 步，雙方共走 265 步，優勢方才能獲勝，附錄 B 有完整的盤面和棋步。

4.7 翻子策略

如圖 4-25，在翻子時，我們會鼓勵翻我方仕（士）或相（象）旁邊的暗子。雖然翻我方仕（士）或相（象）旁邊的暗子可能使得我方仕（士）或相（象）被吃掉，但也增加了我方仕（士）或相（象）控制局勢的機會。我們在那些棋子的旁邊暗子增加翻子的分數。如果那個暗子的四周有兩個一樣的棋子則不會累加翻子分數。

■	■	(3)	■
■	■	將	■
(5)	■	■	(1)
兵	■	■	仕
■	(2)	■	■
■	相	■	■
■	■	■	傌
俥	■	■	(4)

圖 4-25　翻子盤面（輪紅方）

我方的仕（士）旁邊的暗子增加的翻子分數為⌊1000 × (暗子數量 – 對方將（帥）和士（仕）的暗子數量) / 暗子數量⌋；我方的相（象）旁邊的暗子增加的翻子分數為⌊500 × (暗子數量 – 對方將（帥）、士（仕）和象（相）的暗子數量) / 暗子數量⌋。圖 4-25 中(1)的加分為⌊1000 × (26 – 2) / 26⌋ = 923 分；(2)的加分為⌊500 × (26 – 4) / 26⌋ = 423 分。

當對方的將（帥）還存活時，程式可能會翻對方將（帥）旁邊的棋子，這麼做較容易會產生大好或大壞的結果。另外，翻我方兵（卒）旁邊的棋子，也有希望有機會吃掉對方的將（帥）。但是兵（卒）也容易被吃掉，使得到後面在對方的將（帥）還存活的情況下因為兵（卒）的數量較少而吃虧。為了不讓程式做過多的冒險，我們在對方將（帥）旁邊的暗子以及我方兵（卒）旁邊的暗子減少翻子的分數。另外，傌（馬）也算相對容易被吃的棋子，我們也在我方傌（馬）旁邊的暗子略微減少翻子分數。如果那個暗子的四周有兩個一樣的棋子則不會重複

減去翻子分數。

對方的帥（將）旁邊的暗子減去的翻子分數為⌊1500 × (暗子數量 – 我方兵（卒）的暗子數量) / 暗子數量⌋。我方的傌（馬）旁邊的暗子減去的翻子分數為 10 × (對方將（帥）、士（仕）、象（相）、車（俥）和馬（傌）的暗子數量)；我方的兵（卒）旁邊的暗子減去的翻子分數為 70 × (對方士（仕）、象（相）、車（俥）、馬（傌）和卒（兵）的暗子數量)。圖 4-25 中(3)的扣分為⌊1500 × (26 – 4) / 26⌋ = 1269 分；(4)的扣分為 10 × 8 = 80 分；(5)的扣分為 70 × 13 = 910 分。

俥（車）控制局勢的能力不如仕（士）和相（象），但其生存能力也比傌（馬）和兵（卒）來得好，因此我們沒有調整俥（車）旁邊暗子的分數。

相對而言，舊版的程式 DarkCraft[6]和 Darkness[7]較容易翻對方帥（將）旁邊的暗子或是我方兵(卒)旁邊的暗子，而本程式較容易翻我方仕（士）和相（象）旁邊的暗子。這兩種策略各有優缺點，見仁見智。

第五章 結論與未來研究方向

5.1 結論

棋子分數的動態設置使得本程式除了評估子力本身的價值外，還會考慮一個棋子被吃了之後，其它棋子是否會加分。經過評估後，有時候程式會用帥（將）去兌對方的士（仕）。這種策略在一些盤面上會比靜態設置分數來得更有針對性。

可包含一個暗子的五子以內的殘局庫記錄了五子以內的殘局中所有已知盤面的結果。生成殘局庫時可以用一些殘局組合分別代表其它的殘局組合。這樣不用生成所有的殘局組合，可以節省殘局庫的空間。

表 5-1　TCGA 2014 比賽結果

TCGA 2014 Chinese Dark Chess Tournament

No.	Program Name	Game I		Game II		Game III		Game IV		Game V		Game VI		Scores	SOS	Rank
		Opp.	Res.	Opp.	Res.	Opp.	Res.	Opp.	Res.	Opp.	Res.	Opp.	Res.			
1	Yahari	2	2.0	3	1.0	7	2.0	6	1.5	9	2.0	8	2.0	10.5	-	**1**
2	Rainbow	1	0.0	4	1.0	5	0.5	10	0.5	8	0.0	9	0.0	2.0	-	**10**
3	DarkKnight	4	2.0	1	1.0	9	2.0	7	1.0	6	2.0	10	1.0	9.0	-	**2**
4	Homere	3	0.0	2	1.0	10	0.5	5	2.0	7	0.0	6	0.0	3.5	-	**9**
5	Flipper	6	0.5	9	0.5	2	1.5	4	0.0	10	1.0	7	1.0	4.5	-	**8**
6	TuxedoCat	5	1.5	7	0.5	8	1.5	1	0.5	3	0.0	4	2.0	6.0	40.5	**4**
7	Observer	8	1.5	6	1.5	1	0.0	3	1.0	4	2.0	5	1.0	7.0	-	**3**
8	SGTDark	7	0.5	10	2.0	6	0.5	9	1.0	2	2.0	1	0.0	6.0	37.0	**6**
9	Diablo	10	1.5	5	1.5	3	0.0	8	1.0	1	0.0	2	2.0	6.0	37.5	**5**
10	Darkness	9	0.5	8	0.0	4	1.5	2	1.5	5	1.0	3	1.0	5.5	-	**7**

表 5-1 為 TCGA 2014 比賽結果。此次參賽隊伍共有 10 隊，比六輪瑞士制，一輪雙方分先下兩盤。每一盤棋贏棋得一分，和棋得半分，輸棋不得分。表中的 SOS（Sum of Opponents' Scores）為對手分，是比賽中遇到所有對手的積分總和。經過六輪的比賽，最後本程式得到銅牌的成績。

5.2 未來研究方向

有時候程式的兌子策略會出現值得商榷的地方甚至是吃虧，如何再改進各個子力分數的設置有待進一步的研究。在計算子力分數時，如果根節點盤面是紅方的帥、仕和相對黑方的象和雙卒，此時紅方的仕和相都沒有天敵，所以都會設定為 11000 分。但是因為還有一個黑象存活，實際上紅仕的價值仍比紅相高，可以考慮如何在這種情況下調整子力的分數。另外，程式在某些盤面上會做一些特殊處理，雖然發現了其中的好處，但在額外加分上仍然需要找尋和子力設置分數的一個良好的比例。

對於某些組合(例如紅方的帥和仕對黑方雙象) 可用搜尋找到取勝的途徑，在未來生成殘局庫時可以考慮不生成這類型的殘局，以節省殘局庫的空間。

翻子的問題是暗棋裡很重要的一環。我們增加了讓我方的仕（士）和相（象）擴展地盤的機會，而減少了利用翻子去吃對方將（帥）的可能性。如何在翻子策略上取得一個好的平衡點是未來值得探討的問題。

附錄 A 經過簡化後殘局的編號和組合

經過簡化後二子殘局的編號和組合

0. 帥將	1. 帥士	2. 帥包
3. 炮包	4. 炮卒	

經過簡化後三子殘局的編號和組合

5. 帥仕將	6. 帥仕士	7. 帥仕象
8. 帥仕包	9. 帥仕卒	10. 帥相士
11. 帥炮將	12. 帥炮士	13. 帥炮包
14. 帥炮卒	15. 帥兵將	16. 帥兵包
17. 帥兵卒	18. 帥士士	19. 帥士包
20. 帥包包	21. 帥包卒	22. 仕仕士
23. 仕炮卒	24. 仕包卒	25. 炮炮包
26. 炮炮卒	27. 炮兵包	28. 炮兵卒
29. 炮卒卒		

經過簡化後四子殘局的編號和組合

30. 帥仕仕將	31. 帥仕仕士	32. 帥仕仕象
33. 帥仕仕包	34. 帥仕仕卒	35. 帥仕相士
36. 帥仕相象	37. 帥仕俥象	38. 帥仕炮將
39. 帥仕炮士	40. 帥仕炮象	41. 帥仕炮包

42. 帥仕炮卒	43. 帥仕兵將	44. 帥仕兵包
45. 帥仕兵卒	46. 帥仕將士	47. 帥仕將象
48. 帥仕將包	49. 帥仕將卒	50. 帥仕士士
51. 帥仕士象	52. 帥仕士包	53. 帥仕士卒
54. 帥仕象象	55. 帥仕象包	56. 帥仕象卒
57. 帥仕包包	58. 帥仕包卒	59. 帥仕卒卒
60. 帥相相士	61. 帥相炮士	62. 帥相士士
63. 帥相士象	64. 帥相士車	65. 帥相士包
66. 帥相士卒	67. 帥炮炮將	68. 帥炮炮士
69. 帥炮炮包	70. 帥炮炮卒	71. 帥炮兵將
72. 帥炮兵士	73. 帥炮兵包	74. 帥炮兵卒
75. 帥炮將包	76. 帥炮將卒	77. 帥炮士士
78. 帥炮士包	79. 帥炮士卒	80. 帥炮包包
81. 帥炮包卒	82. 帥炮卒卒	83. 帥兵兵將
84. 帥兵兵包	85. 帥兵兵卒	86. 帥兵將卒
87. 帥兵士包	88. 帥兵士卒	89. 帥兵包包
90. 帥兵包卒	91. 帥兵卒卒	92. 帥士士象
93. 帥士士包	94. 帥士士卒	95. 帥士包包
96. 帥士包卒	97. 帥包包卒	98. 帥包卒卒
99. 仕仕相士	100. 仕仕炮士	101. 仕仕炮卒
102. 仕仕士士	103. 仕仕士象	104. 仕仕士包
105. 仕仕包卒	106. 仕炮炮卒	107. 仕炮兵士
108. 仕炮兵卒	109. 仕炮士卒	110. 仕炮象卒

111. 仕炮包卒	112. 仕炮卒卒	113. 仕兵包卒
114. 仕象包卒	115. 仕包包卒	116. 仕包卒卒
117. 炮炮兵包	118. 炮炮兵卒	119. 炮炮包包
120. 炮炮包卒	121. 炮炮卒卒	122. 炮兵兵包
123. 炮兵兵卒	124. 炮兵包卒	125. 炮兵卒卒
126. 炮卒卒卒	127. 兵兵兵卒	

經過簡化後五子殘局的編號和組合

128. 帥仕仕相將	129. 帥仕仕相士	130. 帥仕仕相象
131. 帥仕仕相車	132. 帥仕仕相包	133. 帥仕仕相卒
134. 帥仕仕俥象	135. 帥仕仕炮將	136. 帥仕仕炮士
137. 帥仕仕炮象	138. 帥仕仕炮包	139. 帥仕仕炮卒
140. 帥仕仕兵將	141. 帥仕仕兵包	142. 帥仕仕兵卒
143. 帥仕仕將士	144. 帥仕仕將象	145. 帥仕仕將包
146. 帥仕仕將卒	147. 帥仕仕士士	148. 帥仕仕士象
149. 帥仕仕士包	150. 帥仕仕士卒	151. 帥仕仕象象
152. 帥仕仕象包	153. 帥仕仕象卒	154. 帥仕仕包包
155. 帥仕仕包卒	156. 帥仕仕卒卒	157. 帥仕相相士
158. 帥仕相相象	159. 帥仕相俥象	160. 帥仕相炮士
161. 帥仕相炮象	162. 帥仕相將士	163. 帥仕相將象
164. 帥仕相士士	165. 帥仕相士象	166. 帥仕相士車
167. 帥仕相士包	168. 帥仕相士卒	169. 帥仕相象象

170. 帥仕相象車	171. 帥仕相象包	172. 帥仕相象卒
173. 帥仕俥俥象	174. 帥仕俥炮象	175. 帥仕俥將象
176. 帥仕俥士象	177. 帥仕俥象象	178. 帥仕俥象車
179. 帥仕俥象馬	180. 帥仕俥象包	181. 帥仕俥象卒
182. 帥仕炮炮將	183. 帥仕炮炮士	184. 帥仕炮炮象
185. 帥仕炮炮包	186. 帥仕炮炮卒	187. 帥仕炮兵將
188. 帥仕炮兵士	189. 帥仕炮兵象	190. 帥仕炮兵包
191. 帥仕炮兵卒	192. 帥仕炮將士	193. 帥仕炮將象
194. 帥仕炮將包	195. 帥仕炮將卒	196. 帥仕炮士士
197. 帥仕炮士象	198. 帥仕炮士包	199. 帥仕炮士卒
200. 帥仕炮象象	201. 帥仕炮象包	202. 帥仕炮象卒
203. 帥仕炮包包	204. 帥仕炮包卒	205. 帥仕炮卒卒
206. 帥仕兵兵將	207. 帥仕兵兵包	208. 帥仕兵兵卒
209. 帥仕兵將士	210. 帥仕兵將象	211. 帥仕兵將包
212. 帥仕兵將卒	213. 帥仕兵士包	214. 帥仕兵士卒
215. 帥仕兵將士	216. 帥仕兵將象	217. 帥仕兵將包
218. 帥仕兵包卒	219. 帥仕兵卒卒	220. 帥仕將象象
221. 帥仕將象包	222. 帥仕將象卒	223. 帥仕將包包
224. 帥仕將包卒	225. 帥仕將卒卒	226. 帥仕士士象
227. 帥仕士士包	228. 帥仕士士卒	229. 帥仕士象象
230. 帥仕士象包	231. 帥仕士象卒	232. 帥仕士包包
233. 帥仕士包卒	234. 帥仕士卒卒	235. 帥仕象象車
236. 帥仕象象包	237. 帥仕象象卒	238. 帥仕象包包

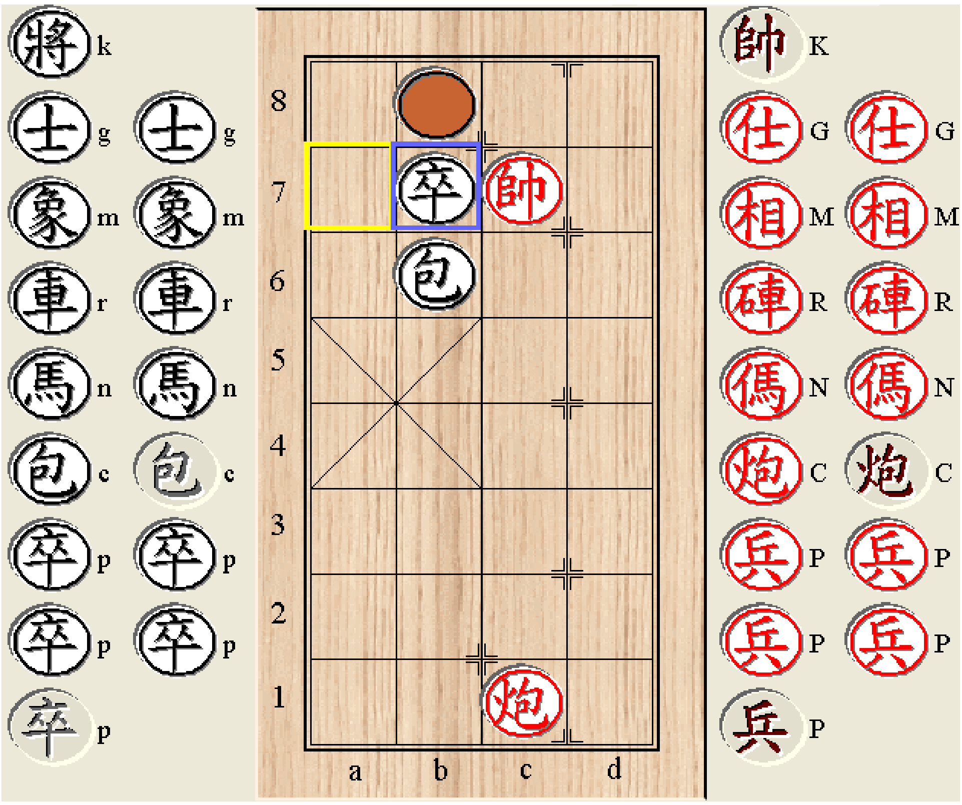
239. 帥仕象包卒	240. 帥仕象卒卒	241. 帥仕包包卒
242. 帥仕包卒卒	243. 帥仕卒卒卒	244. 帥相相俥士
245. 帥相相炮士	246. 帥相相士士	247. 帥相相士象
248. 帥相相士車	249. 帥相相士包	250. 帥相相士卒
251. 帥相俥士象	252. 帥相俥士車	253. 帥相傌士車
254. 帥相炮炮士	255. 帥相炮兵士	256. 帥相炮士士
257. 帥相炮士象	258. 帥相炮士車	259. 帥相炮士包
260. 帥相炮士卒	261. 帥相兵士包	262. 帥相兵士卒
263. 帥相士士象	264. 帥相士士車	265. 帥相士士包
266. 帥相士士卒	267. 帥相士象象	268. 帥相士象車
269. 帥相士象包	270. 帥相士象卒	271. 帥相士車車
272. 帥相士車包	273. 帥相士車卒	274. 帥相士包包
275. 帥相士包卒	276. 帥相士卒卒	277. 帥俥士士象
278. 帥炮炮兵將	279. 帥炮炮兵士	280. 帥炮炮兵包
281. 帥炮炮兵卒	282. 帥炮炮將包	283. 帥炮炮將卒
284. 帥炮炮士士	285. 帥炮炮士包	286. 帥炮炮士卒
287. 帥炮炮包包	288. 帥炮炮包卒	289. 帥炮炮卒卒
290. 帥炮兵兵將	291. 帥炮兵兵士	292. 帥炮兵兵包
293. 帥炮兵兵卒	294. 帥炮兵將包	295. 帥炮兵將卒
296. 帥炮兵士士	297. 帥炮兵士包	298. 帥炮兵士卒
299. 帥炮兵包包	300. 帥炮兵包卒	301. 帥炮兵卒卒
302. 帥炮將卒卒	303. 帥炮士士象	304. 帥炮士士包
305. 帥炮士士卒	306. 帥炮士包包	307. 帥炮士包卒

308. 帥炮士卒卒	309. 帥炮包包卒	310. 帥炮包卒卒
311. 帥炮卒卒卒	312. 帥兵兵兵將	313. 帥兵兵兵包
314. 帥兵兵兵卒	315. 帥兵兵將卒	316. 帥兵兵士包
317. 帥兵兵士卒	318. 帥兵兵包包	319. 帥兵兵包卒
320. 帥兵兵卒卒	321. 帥兵士士包	322. 帥兵士士卒
323. 帥兵士包包	324. 帥兵士包卒	325. 帥兵士卒卒
326. 帥兵包包卒	327. 帥兵包卒卒	328. 帥兵卒卒卒
329. 帥士士象象	330. 帥士士象包	331. 帥士士象卒
332. 帥士士包包	333. 帥士士包卒	334. 帥士士卒卒
335. 帥士包包卒	336. 帥士包卒卒	337. 帥包包卒卒
338. 帥包卒卒卒	339. 仕仕相相士	340. 仕仕相炮士
341. 仕仕相炮卒	342. 仕仕相士士	343. 仕仕相士象
344. 仕仕相士車	345. 仕仕相士包	346. 仕仕相包卒
347. 仕仕俥士象	348. 仕仕炮炮士	349. 仕仕炮炮卒
350. 仕仕炮兵士	351. 仕仕炮兵卒	352. 仕仕炮士士
353. 仕仕炮士象	354. 仕仕炮士包	355. 仕仕炮士卒
356. 仕仕炮象卒	357. 仕仕炮包卒	358. 仕仕炮卒卒
359. 仕仕兵士包	360. 仕仕兵包卒	361. 仕仕士象象
362. 仕仕士象包	363. 仕仕士包包	364. 仕仕士包卒
365. 仕仕象包卒	366. 仕仕包包卒	367. 仕仕包卒卒
368. 仕相炮兵士	369. 仕相炮士卒	370. 仕相炮象卒
371. 仕相兵士包	372. 仕相士包卒	373. 仕相象包卒
374. 仕俥炮象卒	375. 仕俥象包卒	376. 仕炮炮兵士

377. 仕炮炮兵卒	378. 仕炮炮士卒	379. 仕炮炮象卒
380. 仕炮炮包卒	381. 仕炮炮卒卒	382. 仕炮兵兵士
383. 仕炮兵兵卒	384. 仕炮兵士包	385. 仕炮兵士卒
386. 仕炮兵象卒	387. 仕炮兵包卒	388. 仕炮兵卒卒
389. 仕炮士卒卒	390. 仕炮象象卒	391. 仕炮象包卒
392. 仕炮象卒卒	393. 仕炮包包卒	394. 仕炮包卒卒
395. 仕炮卒卒卒	396. 仕兵兵兵卒	397. 仕兵兵包卒
398. 仕兵象包卒	399. 仕兵包包卒	400. 仕兵包卒卒
401. 仕兵卒卒卒	402. 仕象象包卒	403. 仕象包包卒
404. 仕象包卒卒	405. 仕包包卒卒	406. 仕包卒卒卒
407. 炮炮兵兵包	408. 炮炮兵兵卒	409. 炮炮兵包包
410. 炮炮兵包卒	411. 炮炮兵卒卒	412. 炮炮包卒卒
413. 炮炮卒卒卒	414. 炮兵兵兵包	415. 炮兵兵兵卒
416. 炮兵兵包卒	417. 炮兵兵卒卒	418. 炮兵卒卒卒
419. 炮卒卒卒卒	420. 兵兵兵兵卒	421. 兵兵兵卒卒

附錄 B 優勢方需要最多步數獲勝的殘局

五子以內殘局裡，優勢方需要最多步數獲勝的殘局是「帥、炮、兵、包、卒」輪優勢方（紅方）下棋的一個盤面，暗子為紅兵，盤面如下圖所示。



紅方需要走 133 步，雙方共走 265 步，紅方才能獲勝。雙方的棋步如下表所示。目前電腦暗棋棋規規定雙方共 180 步不吃子也不翻子為和棋。下表中紅方在第 46 回合翻子，此時是雙方的第 91 步。之後紅方在第 132 回合吃到黑方的棋子，此時是雙方的第 263 步。雙方由翻子到吃子的過程經過了 263 – 91 = 172 步，所以此例不會被判和棋。

1. c7-c6 b6-a6	35. b5-c5 b4-b5	69. d4-d3 b4-c4	103. c4-b4 c6-c7
2. c6-c5 b7-b6	36. c5-d5 b3-b4	70. d3-d2 a4-a5	104. b6-b7 c7-c6
3. c5-c4 b6-b5	37. c4-c5 b4-b3	71. c2-b2 c4-b4	105. b4-b5 c5-c4
4. c4-c3 b5-b4	38. d5-d4 b3-b2	72. d2-d3 a5-b5	106. a7-a6 c4-c5
5. c3-c2 b4-b3	39. d4-d3 b5-b4	73. b2-a2 b4-c4	107. b7-b8 c5-c4
6. c2-d2 b3-c3	40. c5-c4 b4-b3	74. d3-d2 c4-b4	108. a6-a5 c4-d4
7. c1-c2 c3-d3	41. c4-c3 b3-b4	75. a2-a3 b5-b6	109. a5-a4 d4-d5
8. d2-d1 d3-d2	42. d3-d2 b4-b3	76. c3-c4 b6-c6	110. b8-b7 d5-d6
9. d1-c1 d2-d1	43. d2-d1 b2-a2	77. d2-d3 c6-c7	111. b7-c7 d6-d7
10. c1-b1 d1-c1	44. c3-c2 a2-b2	78. d3-d4 c7-b7	112. a4-a5 d7-d8
11. b1-b2 c1-b1	45. d1-c1 b3-c3	79. d4-d5 b7-b6	113. c7-d7 d8-c8
12. b2-b3 b1-b2	46. b8(P) b2-b3	80. d5-d6 b4-b5	114. a5-a6 c8-b8
13. b3-b4 b2-b3	47. b8-b7 b3-b4	81. c4-c5 b6-a6	115. a6-a7 b8-c8
14. b4-b5 b3-b4	48. b7-b6 c3-d3	82. a3-b3 b5-b6	116. a7-b7 c8-d8
15. b5-a5 a6-b6	49. b6-b5 d3-c3	83. d6-d5 a6-a7	117. b7-b8 c6-c7
16. a5-a6 b6-c6	50. b5-a5 b4-a4	84. b3-b4 a7-b7	118. b5-b6 c7-c8
17. a6-b6 c6-c5	51. c1-b1 a4-b4	85. b4-a4 b6-c6	119. b8-b7 c8-b8
18. b6-c6 c5-c4	52. a5-a4 c3-b3	86. d5-d4 c6-b6	120. b7-c7 b8-c8
19. c2-c3 b4-b5	53. b1-c1 b4-b5	87. a4-a5 b7-b8	121. c7-c6 c8-c7
20. c6-d6 c4-b4	54. c1-d1 b3-c3	88. c5-c6 b8-c8	122. c6-c5 d8-c8
21. d6-d5 b5-c5	55. a4-a3 c3-d3	89. d4-d5 c8-b8	123. c5-b5 c8-b8
22. d5-d4 c5-c4	56. d1-c1 d3-c3	90. d5-d6 b8-a8	124. b5-a5 b8-c8
23. d4-d3 c4-d4	57. a3-a2 c3-b3	91. d6-d7 a8-a7	125. d7-d8 c8-b8
24. d3-d2 b4-c4	58. c2-c3 b5-b6	92. a5-a4 b6-b7	126. d8-c8 c7-d7
25. c3-b3 d4-d3	59. c1-c2 b6-b5	93. c6-c7 b7-b6	127. b6-c6 b8-b7
26. d2-c2 d3-c3	60. c2-d2 b5-a5	94. d7-d8 b6-a6	128. a5-b5 d7-d8
27. c2-b2 c3-c2	61. d2-d3 a5-a4	95. a4-b4 a6-b6	129. c6-d6 b7-c7
28. b2-a2 c4-c3	62. a2-a1 b3-b2	96. c7-b7 b6-c6	130. b5-b6 c7-d7
29. b3-b4 c2-b2	63. d3-d4 b2-b3	97. d8-c8 c6-b6	131. b6-b7 d7-c7
30. a2-a3 b2-b3	64. a1-b1 b3-b4	98. b4-c4 b6-c6	132. b7-c7 d8-d7
31. a3-a4 b3-a3	65. d4-d3 b4-b3	99. c8-b8 a7-a6	133. d6-d7
32. a4-a5 c3-b3	66. b1-c1 a4-b4	100. b8-a8 a6-a5	
33. b4-c4 a3-a4	67. d3-d4 b4-a4	101. b7-b6 a5-b5	
34. a5-b5 a4-b4	68. c1-c2 b3-b4	102. a8-a7 b5-c5	

參考文獻

[1] 龔長偉，蒙地卡羅樹狀搜尋在暗棋上的應用，國立東華大學資訊工程研究所碩士論文，2011。

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

[3] 賴學誠，電腦暗棋程式與經驗法則之配合與實作，國立東華大學資訊工程研究所碩士論文，2008。

[4] 謝政孝，暗棋中棋種間食物鏈關係之探討與實作，國立臺灣師範大學資訊工程研究所碩士論文，2010。

[5] 勞永祥，電腦暗棋之人工智慧改良，國立臺灣師範大學資訊工程研究所碩士論文，2011。

[6] 施宣丞，電腦暗棋程式 DarkCraft 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，2012。

[7] 詹凱翔，電腦暗棋程式 Darkness 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，2013。

[8] 林庭羽，電腦暗棋殘局庫之研究，中原大學應用數學系碩士論文，2013。

[9] ICGA Tournaments 網站，http://www.grappa.univ-lille3.fr/icga/，2014。

[10] 維基百科，http://zh.wikipedia.org/，2014。

[11] Chess Programming 網站，https://chessprogramming.wikispaces.com/，2014。

[12] D. E. Knuth and R. W. Moore, “An Analysis of Alpha-Beta Pruning”, Artificial Intelligence 6(4), pp. 293-326, 1975.