

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

電腦暗棋之人工智慧改良

Artificial Intelligence Improvement of Chinese  
Dark Chess

研究生： 勞永祥 撰

中華民國一百年六月

## 摘要

一直以來電腦棋類人工智慧的發展主要集中在完全資訊的遊戲，完全資訊的棋類遊戲，盤面的資訊能完全掌握，審局資訊充足，並不含機率的部分。

電腦暗棋是屬於不完全資訊含機率性的棋類遊戲，不像西洋棋、中國象棋是屬於完全資訊的棋類遊戲，如果用一般遊戲樹進行搜尋，在走棋與翻棋夾雜的情況下，若需要對未翻棋子也要作走步搜尋，則需要對所有的未翻棋子都作假設模擬，以求得一個接近的結果。但並不容易準確的審出結果。

經過 ICGA 2010、TAAI 2010 及台大資工所 game theory 課程等多次電腦暗棋比賽，由國立東華大學資訊工程所、國立台灣師範大學資訊工程所以及國立臺灣大學資訊工程所等所開發的電腦暗棋程式都有著共同問題，就是走子或翻棋。

由於無法合理地走子或翻棋，導致走閒步，棋局無進展。這樣的結果使得在電腦暗棋的比賽中，往往優勢的一方也因為無目標，局勢無法進展，而變成平手結果。

本論文主要提出電腦暗棋的一套新的策略以解決局勢無法順利進展的問題。另外提出更準確的棋子間距離影響力之計算方法。實測結果顯示，本程式 Black Cat 比起去年 ICGA2010 及 TAAI2010 的亞軍程式 Dark Chess Beta（本校研究生謝政孝所研發）約有五成六的贏率。

關鍵字：電腦暗棋、不完全資訊、人工智慧

# ABSTRACT

Computer chess-playing is an area of artificial intelligence research. Most of the time, it focuses on the design of chess-playing software to play games with complete information in which all of the players know the other players' preferences.

Chinese dark chess is an incomplete information game with probabilities, which is not the same as complete information games, such as chess or Chinese chess. If we use conventional game-tree searching techniques to play Chinese dark chess, then the number of branches will be very large because there are lots of moves for both “dark pieces” and “bright pieces”. Hence, it is impractical to generate all possible moves in a given board position in order to find a good move.

During and after participating in the competitions of ICGA 2010, TAAI 2010 and final project of NTU game theory course, we found that today most state-of-the-art Chinese dark chess programs, including those developed by NTNU, NTU, and NDHU, still could not play well in moving the “bright pieces” or flipping the “dark pieces”. Due to this phenomenon, most of the Chinese dark chess matches result in a draw even if one player has a large material advantage over the other.

In this thesis, we propose some approaches to fix the problem. We provide a path tracing method to compute more accurately the influence among the chess pieces according to their walking distances.

The experimental results indicate that our program "Black Cat" obtained a winning rate of 56.0% against the program "Dark Chess Beta" which won the Silver medals at the ICGA 2010 and TAAI 2010.

Keywords: artificial intelligence, Chinese dark chess, incomplete information game

# 目 錄

摘 要.....	i
ABSTRACT.....	ii
目 錄.....	iii
表格目錄.....	iv
圖表目錄.....	iv
<b>第一章 緒論.....</b>	<b>1</b>
第一節 暗棋人工智慧的現況.....	1
第二節 暗棋的簡單及困難之處.....	3
第三節 文獻探討.....	4
<b>第二章 使用的技術介紹.....</b>	<b>11</b>
第一節 對局樹.....	11
第二節 Alpha-Beta 搜尋演算法.....	13
第三節 Transposition Table.....	20
第四節 允許空步.....	24
第五節 Iterative Deepening Search.....	25
<b>第三章 我們對審局評分的作法 .....</b>	<b>26</b>
第一節 翻棋位置的選擇.....	26
第二節 距離威脅力計算.....	32
第三節 動態分數調整.....	40
第四節 配分設計.....	42
第五節 審局函數設計.....	44
第六節 走步或翻棋的選擇方法.....	45
<b>第四章 結論與未來研究方向 .....</b>	<b>48</b>
第一節 結論.....	48
第二節 未來研究方向.....	51
<b>參考著作.....</b>	<b>53</b>

## 表格目錄

表 1-1 ICGA 2010 的比賽結果.....	2
表 1-2 棋盤資料結構表.....	4
表 2-1 Zobrist Hash 以相同盤面作兩次測試的結果表.....	24
表 3-2 棋盤位置配分.....	43
表 4-1 實測結果.....	50

## 圖表目錄

圖 1-1 棋子編號圖.....	5
圖 1-2 靈氣權重表.....	7
圖 1-3 靈氣權重的例子圖.....	7
圖 1-4 在角落及邊上扣除權重分數.....	8
圖 1-5 Flood Fill 的方式.....	9
圖 1-6 採用 Manhattan Distance 計算.....	9
圖 1-7 固定位置權重分.....	10
圖 2-1 對局樹.....	11
圖 2-2 井字遊戲圖解.....	12
圖 2-3 Min-Max 搜尋示意圖.....	14
圖 2-5 Alpha-Beta 搜尋法示意圖.....	15
圖 2-6 Alpha-Beta 搜尋法以深度優先從左至右拜訪的樹狀圖.....	16
圖 2-7 Alpha-Beta 搜尋演算法最少必須搜尋的節點示意圖.....	17
圖 2-8 MinMax-Alpha-Beta 搜尋演算法的虛擬碼.....	18
圖 2-9 Nega-Max 搜尋演算法虛擬碼.....	19
圖 2-10 Nega-Max-Alpha-Beta 搜尋演算法虛擬碼.....	19
圖 2-11 Transposition Table 初始盤面.....	20
圖 2-12 路徑 1.....	20
圖 2-13 路徑 2.....	21
圖 2-14 結果盤面.....	21
圖 2-15 允許空步例 1.....	24
圖 2-16 允許空步例 2.....	25
圖 3-1 棋子種類及子數.....	26
圖 3-2 機率分數計算範例盤面.....	27
圖 3-3 位置 1 的黑方安全棋子.....	27
圖 3-4 位置 1 的紅方安全棋子.....	27
圖 3-5 位置 2 的黑方安全棋子.....	27
圖 3-6 位置 2 的紅方安全棋子.....	27
圖 3-7 位置 3 的黑方安全棋子.....	28

圖 3-8 位置 3 的紅方安全棋子 .....	28
圖 3-9 位置 4 的黑方安全棋子 .....	28
圖 3-10 位置 4 的紅方安全棋子 .....	28
圖 3-11 位置 5 的黑方安全棋子 .....	28
圖 3-12 位置 5 的紅方安全棋子 .....	28
圖 3-13 位置 6 的黑方安全棋子 .....	29
圖 3-14 位置 6 的紅方安全棋子 .....	29
圖 3-15 位置 7 的黑方安全棋子 .....	29
圖 3-16 位置 7 的紅方安全棋子 .....	29
圖 3-17 位置 8 的黑方安全棋子 .....	29
圖 3-18 位置 8 的紅方安全棋子 .....	30
圖 3-19 位置 9 的黑方安全棋子 .....	30
圖 3-20 位置 9 的紅方安全棋子 .....	30
圖 3-21 期待翻出炮的位置 .....	30
圖 3-22 期待被包吃的位置及吃包的位置 .....	31
圖 3-23 對紅炮的反制點 .....	31
圖 3-24 期待炮/包位置範例 .....	32
圖 3-25 距離威脅盤面 .....	33
圖 3-26 控制域 .....	33
圖 3-27 best fit search 距離 .....	34
圖 3-28 best fit search 的執行過程 .....	36
圖 3-29 控制域產生 .....	36
圖 3-30 炮/包動態控制域處理 .....	37
圖 3-31 炮/包動態控制域處理-包架移動 .....	37
圖 3-32 包口棋的預想狀況 .....	38
圖 3-33 包口棋的實際狀況 .....	38
圖 3-34 通過包後的壓迫 .....	38
圖 3-36 暗棋的子力大小 .....	40
圖 3-37 贏棋規律 1 .....	41
圖 3-38 贏棋規律 2 .....	42
圖 3-39 贏棋規律 3 .....	42
圖 3-40 最長走步距離 .....	43
圖 3-41 需要翻棋的盤面 .....	45
圖 3-42 兌子的盤面 .....	47
圖 4-1 TAAI 2010 平手盤面 1 .....	48
圖 4-6 將士兵帥的殘局 .....	51
圖 4-7 三卒二兵的殘局 .....	51

# 第一章 緒論

## 第一節 暗棋人工智慧的現況

經過 ICGA 2010、TAAI 2010 及由徐讚昇老師於台大開授的課程「電腦對局理論」的期末比賽等電腦暗棋比賽，由國立東華大學資訊工程所、國立台灣師範大學資訊工程所以及國立臺灣大學資訊工程所等所開發的電腦暗棋都有著共同的缺點—對局勢無法有效掌握，而導致走閒步的問題。這樣的結果造成了電腦暗棋比賽時往往變成平手結果。

表 1-1 為 ICGA 2010 的參賽隊伍及比賽結果。

程式名稱	所屬單位
Dark chesser	東華大學
Modark	東華大學
Flipper	台灣大學
Leave or Lose	東華大學
Dark Chess Beta	台灣師範大學

	Dark Chess Beta	Dark chesser	Flipper	Leave Or Lose	Modark	Score	
1	Dark Chess Beta	X	2	1	1.5	0.5	5 (6.5)
2	Dark chesser	0	X	1.5	0	0.5	2
3	Flipper	1	0.5	X	1	1	3.5
4	Leave or Lose	0.5	2	1	X	1	4.5
5	Modark	1.5	1.5	1	1	X	5 (7)

表 1-1 ICGA 2010 的比賽結果

其中 2 分為全勝、0 分為全敗、1 分為雙方平手。

從 ICGA 2010 的 10 場比賽的 20 局對局中，平手的對局佔了 12 局，優勢方無法合理的攻擊導致平手的問題相當明顯，有著 60% 的對局並無法像人類的玩暗棋的順利進行到合理地分出勝負。

像西洋棋、中國象棋等完全資訊的棋類遊戲 (Perfect Information Game)，由於可完全掌握盤面資訊，故審局函數可以作出較準確的判斷，可以使電腦有很高的棋力。

而暗棋為不完全資訊帶機率性的棋類遊戲 (Imperfect Information Game)，故需深入探討如何運用已知的資訊，輔以統計的方式，對盤面所剩的未翻棋子計算出合理的推論結果，以供審局使用而作出合理的判斷。

使用統計的方式產生出來的結論，以作為推論的資訊，同樣可以應用在多人遊戲中，例如德州撲克、大老二及麻將等這一類型的遊戲中。



這種類似人類記牌，猜牌的方式更接近人類的思考模式，跟人類實際在打撲克牌或麻將的應用非常類似。

## 第二節 暗棋的簡單及困難之處

暗棋玩法相對於中國象棋，簡單不少，並無複雜走法或局面需判斷，只需單純走步或翻子，最終目的將對手的棋子吃光即可。而且棋盤也相對較小，只有 32 個格子。

但暗棋為不完全資訊含機率性的棋類遊戲，不像中國象棋是屬於完全資訊的棋類遊戲。完全資訊棋類遊戲可以清楚的掌握盤面資訊，以達到較準確的審局處理結果。但暗棋由於其資訊不完全，只能針對現在的盤面進行審局。雖然知道剩下未翻的棋子為何，但由於需要翻棋才能有效，就算是當下是優勢的狀況，在錯誤的翻棋或運氣不好的情況下，情勢是可以出現大逆轉的情形。故翻棋策略亦為暗棋重要的一環。

### 第三節 文獻探討

電腦暗棋程式相關的論文首次出現於 2008 年，包括由國立台灣師範大學資訊工程研究所，謝曜安的論文「電腦暗棋之設計及實作」[10]及國立東華大學資訊工程研究所，賴學誠的論文「電腦暗棋程式與經驗法則之配合與實作」[11]，謝政孝的論文「暗棋中棋種間食物鏈關係之探討與實作」[12]，陳柏年的論文「電腦對局知識取得與應用」[13]。

謝曜安的論文「電腦暗棋之設計及實作」主要是介紹棋類所使用的技術，包括棋盤資料結構，如表 1-2 所示。

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

表 1-2 棋盤資料結構表

棋盤用一維陣列表示法，沒有採用較為直觀的二維陣列原因是：一維陣列表示法擁有比較快的存取速度，因為高階語言中的二維陣列，經編譯程式處理之後，仍然是被轉換成一維的記憶體。

棋子編號如圖 1-1。

空格	將	黑士		黑象		黑車		黑馬
0	1	2	3	4	5	6	7	8
黑馬	黑包		黑卒					紅帥
9	10	11	12	13	14	15	16	17
紅仕		紅相		紅俤		紅馮		紅炮
18	19	20	21	22	23	24	25	26
紅炮	紅兵					暗棋		
27	28	29	30	31	32	33		

圖 1-1 棋子編號圖

他使用的演算法簡述如下：

1. Alpha-Beta 搜尋演算法：Alpha-Beta 搜尋演算法必須使用兩個參數，分別是  $\alpha$  和  $\beta$ 。其中  $\alpha$  是記錄最大層節點目前的最大值，而  $\beta$  是記錄最小層節點的最小值，兩個參數以傳值 (call by value) 的方式傳遞給下層的子樹。如果在取最大值的時候，發現了一個大於等於  $\beta$  的值，就不用再對其它分枝進行搜尋，這就是所謂的  $\beta$  截斷；同理，在取最小值的時候，發現了一個小於等於  $\alpha$  的值，也不用再對其它分枝進行搜尋，這就是所謂的  $\alpha$  截斷。
2. 寧靜搜尋：當剩餘深度為零時不立刻返回審局函數的評估分數值，而是根據盤面只針對吃子步進行展開，直到最後的寧靜盤面，也就是不再有棋子被吃為止，才返回審局函數的值。這樣的做法多少可以克服

因水平效應而造成的搜尋不準確的結果。

3. Transposition Table：是一個可以將已搜尋過的節點資訊記錄起來的雜湊表。節點資訊通常包括：盤面 key 值、節點搜尋深度、節點分數、節點類型…等。一般使用 Zobrist Hash 的方式來進行盤面 key 值的生成，以達到快速檢測當前節點是否已經搜尋過的目的。
4. 循環剪裁：用一個小型的 Hash Table，其大小為 128 個 entry，只記錄當前路徑所走過的每個節點的盤面 key 值，節點返回上一層時會移除它在 Hash Table 中的 key 值。當要替目前節點展開所有合理著法進行子樹的搜尋前，我們會先探測在這小型的 Hash Table 裡，是不是已經有相同的盤面了，如果有的話則呼叫審局函數，返回審局分數，而不用再對子樹進行搜尋，進而提高搜尋效率，這就是所謂的循環剪裁。
5. 允許空步：暗棋程式實作中，我們在搜尋時賦予走空步另一層意義，走空步只是為了在還有暗棋的盤面時，如果走的每一步棋都會使情況更糟，這時不應該強迫走棋，因為在現實的狀況中，它可能會去做翻棋的動作。
6. 疊代加深搜尋：在 alpha-beta search 時，每深入一層前先對當前的所有合法走步作審局，並按審局分作排序，以利選擇更好的路徑並方便 alpha-beta search 作截斷。
7. 走棋與翻棋的抉擇：走棋與翻棋的抉擇提出的方式為將搜尋分為兩段處理，先對明棋作「走棋」搜尋，再對每個未翻的棋子都賦予其每一個可能出現的兵種，然後再對盤面作淺層的明棋搜尋，以取出最有利

的位置。若翻棋的位置比較有利，則選擇翻棋，否則走棋。

賴學誠的論文「電腦暗棋程式與經驗法則之配合與實作」的主要論述為翻牌系統，其概念以經驗法則為主，其規律為隔棋期待翻炮/包吃對方的棋或翻出被對方炮/包攻擊較小的位置。靈氣系統為各子對附近的棋子的影響力，由近而遠的遞減，最大的距離為 5，距離 5 以上則無法影響，靈氣的影響包括了正向和負向兩種。正向為子力大者對子力小者的保護，故分數會增加。負向為對對手子力的影響，為減分作用。

棋子	將(帥)	士(仕)	象(相)	車(俸)	馬(偶)	炮(炮)	卒(兵)
帥(將)	120	0	0	0	0	0	108
仕(士)	108	106	0	0	0	0	0
相(象)	60	60	58	0	0	0	0
俸(車)	36	36	36	34	0	0	0
偶(馬)	24	24	24	24	22	0	0
炮(包)	48	48	48	48	48	0	0
卒(兵)	0	12	12	12	12	0	10

圖 1-2 靈氣權重表

20 12 12	馬	30 12 12				仕	
60 16 24	30 48 12	象	30 12 8	60 9 6	20 4	15	
20 48 8	60 24 24	30 48 8	60 16 6	20 12 4	15 9		
15 24 6	炮	60 24 6					

圖 1-3 靈氣權重的例子圖

如圖 1-3，上方的分數為象對士的靈氣權重、中間的分數為包對士的

靈氣權重、最下方的分數為馬對士的靈氣權重。

搜尋為採用 Iterative Deepening 的 Alpha-Beta search，和謝曜安的相同。

配以經驗得來的位置分表，輔助審局作出更正確的判斷。

如圖 1-4。

-10	-5	-5	-5	-5	-5	-5	-10
-5							-5
-5							-5
-10	-5	-5	-5	-5	-5	-5	-10

圖 1-4 在角落及邊上扣除權重分數

2010 年國立台灣師範大學資訊工程研究所，謝政孝的論文「暗棋中棋種間食物鏈關係之探討與實作」的其主要想法承接了謝曜安的作法，並改良了棋子分數的計算為針對盤面尚存的子（包括明棋和未翻之棋），去計算每顆棋子還能夠吃多少顆子，且吃的子價值多少，將之加總起來再乘以那個棋子本身的權重，最後所有尚存之子再加上一個共同的基本分數得出各子的權重的分數。

再來使用 Flood Fill 的方式來偵測各子的相連程度，若相連則計算其距離威脅度及被威脅度而得出盤面各子彼此間的距離分。

如圖 1-5 所示。

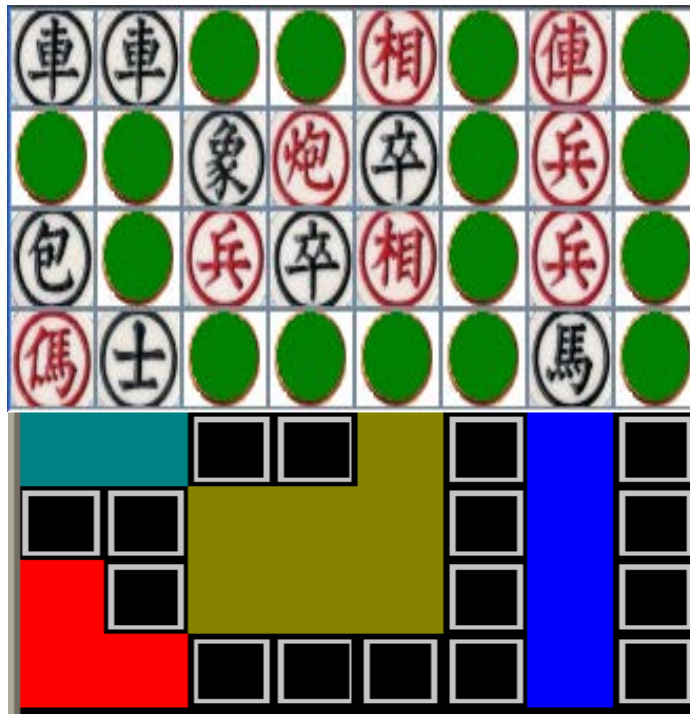


圖 1-5 Flood Fill 的方式

棋子之間的距離採用 Manhattan Distance 計算，如圖 1-6，其中距離分為 2 的(兩子間距離+1)次方。

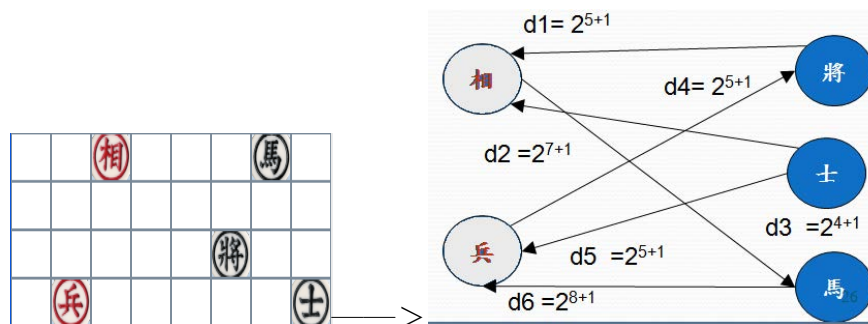


圖 1-6 採用 Manhattan Distance 計算

再給與盤面的固定位置權重分，如圖 1-7 所示。

2	3	3	3	3	3	3	2
3	4	4	5	5	4	4	3
3	4	4	5	5	4	4	3
2	3	3	3	3	3	3	2

圖 1-7 固定位置權重分

最後將權重分、距離分和位置權重分相加作為棋子的最終分數，作為審局函數的使用。

其翻棋策略採經驗法則以保守式的翻棋，選擇無損失的位置翻棋，例如隔子翻期待炮/包，翻斜角以保子。

陳柏年的論文「電電腦對局知識取得與應用」的論文中對暗棋的搜尋複雜度作了分析，點出了開局處理、翻棋的策略、盤中審局函數的設計及殘局的處理等等的問題及想法。



## 第二章 使用的技術介紹

### 第一節 對局樹

人類下棋時，一般採取的思考方式為：如果我走了這一步棋，對手下哪一步作回應，再來我走的又一步，對手再走另一步等。如此交錯的推演，直到棋手認為推演夠了，就會審視推演後的局面是否夠好，若夠好，則下出推演中的第一步棋。若不夠好，則從頭再找另一步棋，再作推演一次。直到找到所有走法中最好的一步。

利用樹狀結構，將對奕的過程給表現出來，稱為對局樹或遊戲樹。如圖 2-1。

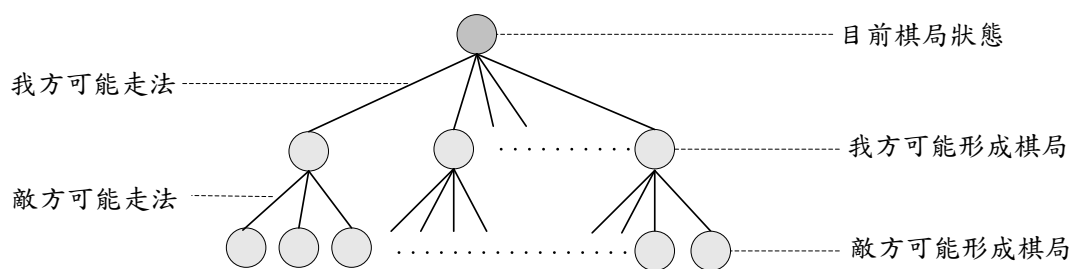


圖 2-1 對局樹

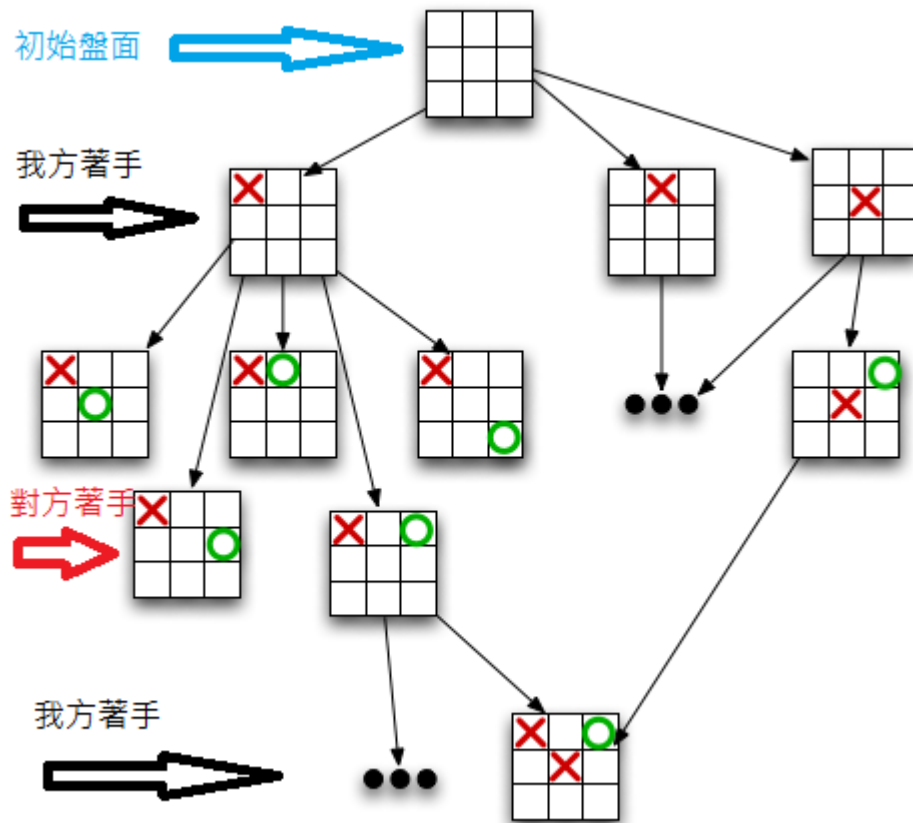


圖 2-2 井字遊戲圖解

再以井字遊戲為例，從圖 2-2 中不難發現，對局樹的節點數會隨著搜尋的層數增加，而呈指數的成長。像井字遊戲這樣的棋類遊戲，因為棋盤上可以走的位置較少，且會隨著所下的棋子增加而減少，因此可以在合理的搜尋時間找出最好的答案。像中國象棋或西洋棋這類的遊戲，並不會有棋子增加而走步減少外，還可能會因為棋子減少反而使合法走步增加，這樣若每一步棋有  $k$  種合法走步，雙方各推演 5 步的話，便需要對  $k^{10}$  個走步作搜尋。這樣子的搜尋量，一般來說，無法在合理的時間範圍內搜尋出來。所以通常只能做有限層數的搜尋，或者想辦法不要搜尋所有可能的走法而仍能保有一定的棋力。

當對局樹搜尋到底層的葉節點時，程式必須呼叫審局函數，來替盤面打分

數，然後由底層的葉節點將分數往上依序傳回給父節點，一直到最後整棵對局樹都搜尋完了，根節點將會得到最有利我方的分數值，進而找出最有利我方的一步棋。

## 第二節 Alpha-Beta 搜尋演算法

Alpha-Beta 搜尋演算法最早在 1956 年由 McCarthy John 於 Dartmouth Conference 提出此概念。後來由 Alexander Brudno 在 1963 發表。

Alpha-Beta 搜尋演為改進 Min-Max 搜尋法的效能。

Min-Max 搜尋演算法是假設敵我雙方在走步推演時，都是選擇對己方最有利的走步。它採取深度為優先的搜尋演算法。從根節點出發，對所有合法的走法，以深度優先的方式向下進行節點的拜訪，當拜訪到給定深度後，在葉節點處使用審局函數判斷盘面好壞。

Min-Max 搜尋演算法在我方走步時，選擇能得到最大分數的走法。在對手走步時，則是選取得到最小分數的走法。因此 Min-Max 搜尋演算法在對局樹不同層中，交互使用取 Max 值和 Min 值的走法。例如圖 2-3。

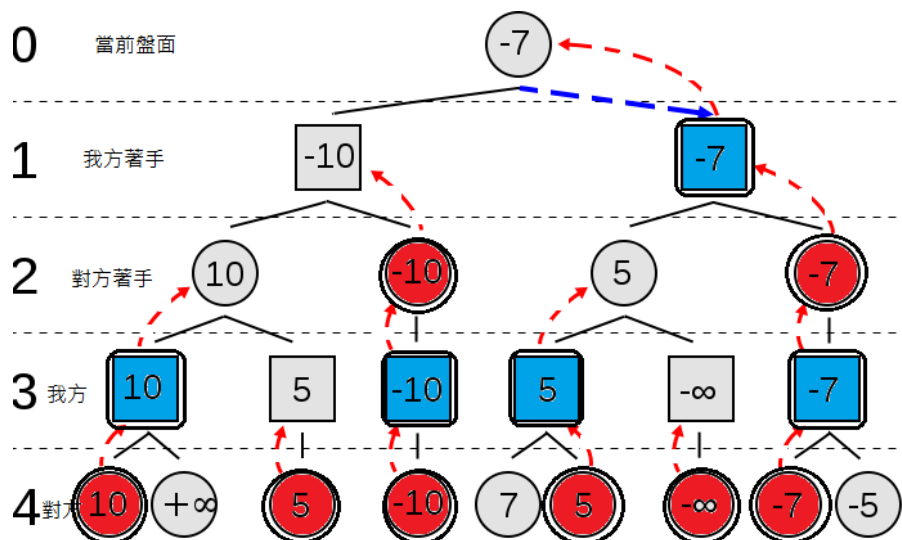


圖 2-3 Min-Max 搜尋示意圖

根據審局函數會回傳不同的分數，每個節點根據我方或對方走步，分別選擇能夠得到最大或最小分數的走法，如此以深度優先的方式進行著，在葉節點計算審局分，並將結果按所屬的層回傳相應的數值，最後便能得到在根節點處對我方而言最好的走法。

Min-Max 搜尋演算法的虛擬碼，如圖 2-4。

```

function MINI-MAX(N) is
begin
  if N is a leaf then
    return the estimated score of this leaf
  else
    Let N1, N2, ..., Nm be the successors of N;
    if N is a Min node then
      return min{MINI-MAX(N1), .., MINI-MAX(Nm)}
    else
      return max{MINI-MAX(N1), .., MINI-MAX(Nm)}
end MINI-MAX;

```

圖 2-4 Min-Max 搜尋演算法的虛擬碼

Min-Max 算法的最大的問題在於每個節點都需要被搜尋一遍，雖然可以解決問題，但相當的沒有效率。

而 Alpha-Beta 搜尋法的核心思想是：在搜尋的過程中，若發現無論如何都無法改變對方目前的最佳分數時，就可以提早放棄，不必浪費時間搜尋其它的著法。如圖 2-5。

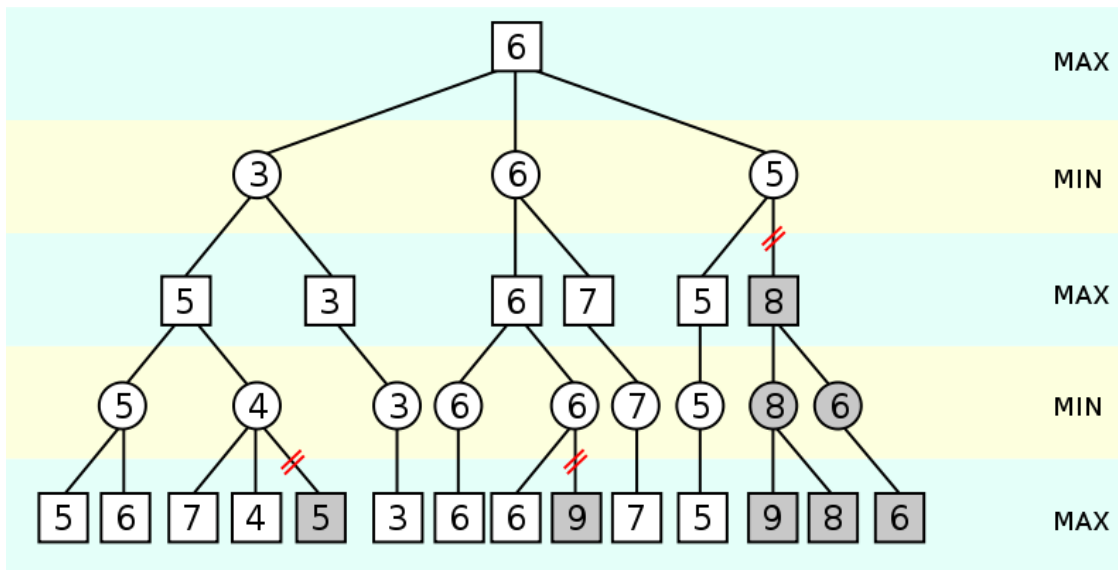


圖 2-5 Alpha-Beta 搜尋法示意圖

圖 2-5 中，方形節點會從其子節點中取最大值，圓形節點會從其子節點中取最小值，而葉節點則會傳回審局函數值。其中根節點，經由深度優先搜尋左子樹，得到左子樹的值為 3，由於根節點會從其子節點中取最大值，所以必需大於 3，否則不會被根節點選取。故搜尋中間子樹時，得到 6，所以根節點換成 6，最後在搜尋右子樹時得到 5。最後，根節點的值為中間子樹的答案，為 6。

Alpha-Beta 搜尋演算法必須使用兩個參數，分別是  $\alpha$  和  $\beta$ 。其中  $\alpha$  是記錄最大層節點目前的最大值，而  $\beta$  是記錄最小層節點的最小值，兩個參數以傳交錯的

方式傳遞給下層的子樹。所以，在最大層取最大值的時候，若發現了一個大於等於  $\beta$  的值，就不用再對其它分枝進行搜尋，這就是所謂的  $\beta$  截斷。同理，在最小層取最小值的時候，發現了一個小於等於  $\alpha$  的值，也不用再對其它分枝進行搜尋，這就是所謂的  $\alpha$  截斷。

圖 2-6 是 Alpha-Beta 搜尋另一個比較複雜的範例，此圖是以深度優先從左至右拜訪的樹狀圖。

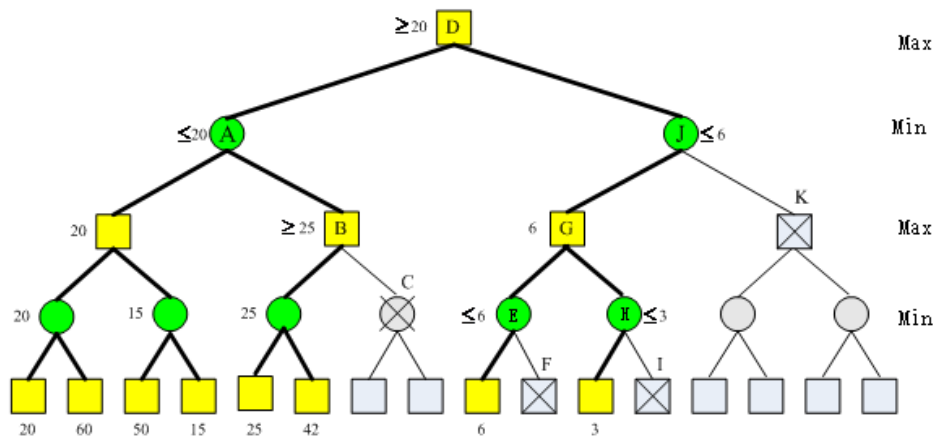


圖 2-6 Alpha-Beta 搜尋法以深度優先從左至右拜訪的樹狀圖

其節點截斷順序如下：

1. B 節點取值 25 的時候， $25 \geq 20$ ，造成 C 節點的截斷 ( $\beta$  截斷)。
2. E 節點取值 6 的時候， $6 \leq 20$ ，造成 F 節點的截斷 ( $\alpha$  截斷)。
3. H 節點取值 3 的時候， $3 \leq 6$ ，造成 I 節點的截斷 ( $\alpha$  截斷)。
4. J 節點取值 6 的時候， $6 \leq 20$ ，造成 K 節點的截斷 ( $\alpha$  截斷)。

很顯然地，Alpha-Beta 搜尋演算法的搜尋效率與走法的排列順序有極密切的關係。如果總是先嘗試壞的走法的話，那最終將會與 Min-Max 搜尋演算法一樣，完全沒有任何截斷的機會。倘若能將好的走法排在前面優先嘗試的話，就可以儘

早發生截斷，省掉對其它節點不必要的搜尋。

以圖 2-6 為例，共 31 個節點，若使用 Alpha-Beta 搜尋法，最終只搜尋了 19 個節點，省了 12 個節點。在搜尋走法時，對走法作排序時，Knuth 和 Moore 證明了 Alpha-Beta 搜尋演算法最少必須搜尋的節點數目為：

$$n = 2b^{d/2} - 1 \quad (d \text{ 為偶數})$$

$$n = b^{(d+1)/2} + b^{(d-1)/2} - 1 \quad (d \text{ 為奇數})$$

其中 n 代表搜尋節點數目，b 代表 branch factor，d 代表搜尋深度。節點示意圖，

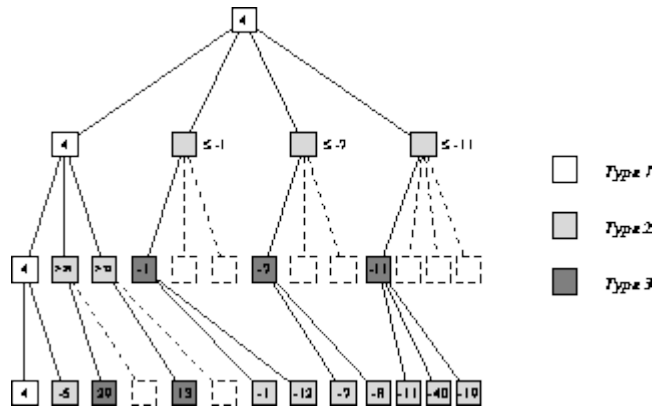


圖 2-7 Alpha-Beta 搜尋演算法最少必須搜尋的節點示意圖

並將搜尋樹的節點分為 3 類：

Type 1：principal variation，根節點為 Type 1，

每個 Type 1 節點的第一個子節點為 Type 1。

Type 2：每個 Type 1 節點的除第一個子節點外，其它為 Type 2。

Type 3：Type 2 的子節點為 Type 3，而 Type 3 的子節點為 Type 2。

對 Type 1 及 Type 2 作走步排序對加速搜尋有效，而對 Type 3 節點排序則沒幫助。

MinMax-AlphaBeat 搜尋演算法的虛擬碼：

```
function MINI-MAX-AB(N, A, B) is ;; Here A is always less than B
begin
  if N is a leaf then
    return the estimated score of this leaf
  else
    Set Alpha value of N to -infinity and
      Beta value of N to +infinity;
    if N is a Min node then
      For each successor Ni of N loop
        Let Val be MINI-MAX-AB(Ni, A, Min{B, Beta of N});
        Set Beta value of N to Min{Beta value of N, Val};
        When A >= Beta value of N then
          Return Beta value of N endloop;
      Return Beta value of N;
    else
      For each successor Ni of N loop
        Let Val be MINI-MAX-AB(Ni, Max{A, Alpha value of N}, B);
        Set Alpha value of N to Max{Alpha value of N, Val};
        When Alpha value of N >= B then
          Return Alpha value of N endloop;
      Return Alpha
a value of N;
end MINI-MAX-AB;
```

圖 2-8 MinMax-Alpha-Beta 搜尋演算法的虛擬碼

Knuth 和 Moore 在 1975 年提出了 Nega-Max 搜尋演算法，消除了要去判斷現在是哪一方而取最大值還是最小值的差別，其主要的構想在於，遊戲本身為零和遊戲，即我方有利的走步，亦即為對方不利的走步。



Nega-Max 搜尋演算法虛擬碼，如圖 2-9。

```
int nega-Max( int depth ) {
    if ( depth == 0 ) return evaluate();
    int max = -oo;
    for ( all moves ) {
        score = -nega-Max( depth - 1 );
        if( score > max )
            max = score;
    }
    return max;
}
```

圖 2-9 Nega-Max 搜尋演算法虛擬碼

以 Nega-Max 形式寫成的 Alpha-Beta 搜尋演算法，當  $\alpha$  和  $\beta$  兩個參數在傳遞給下層的時候，是將兩個順序對調，並加上負值，這樣做使得  $\alpha$  截斷也能簡化成以  $\beta$  截斷的形式表現出來，消除了 Min-Max 形式的 Alpha-Beta 搜尋演算法在  $\alpha$  截斷與  $\beta$  截斷判斷形式上的不同，並且讓程式碼更簡潔。

Nega-Max-Alpha-Beta 搜尋演算法虛擬碼：

```
int alpha-Beta( int alpha, int beta, int depthleft ) {
    if( depthleft == 0 ) return quiesce( alpha, beta );
    for ( all moves ) {
        score = -alpha-Beta( -beta, -alpha, depthleft - 1 );
        if( score >= beta )
            return beta; // beta-cutoff
        if( score > alpha )
            alpha = score; // alpha acts like max in MiniMax
    }
    return alpha;
}
```

圖 2-10 Nega-Max-Alpha-Beta 搜尋演算法虛擬碼

### 第三節 Transposition Table

在遊戲樹的搜尋過程中，雖然走不同的走步，但最終盤面的狀態可能是完全相同，如圖 2-11。



圖 2-11 Transposition Table 初始盤面

不管是圖 2-12 的走路徑，或是圖 2-13 的路徑，都會到達圖 2-14 的同一個盤面。



圖 2-12 路徑 1



陣列  $Zobrist[\text{棋子類型}][\text{棋子位置}]$ 。有了  $Zobrist[\text{棋子類型}][\text{棋子位置}]$  這樣的二維陣列以後，採取漸進式更新，只需要將當前 key 值（假設叫： $ZobristKey$ ）做以下步驟：

1.  $ZobristKey^{\wedge}=Zobrist[\text{移動棋子類型}][\text{移動棋子原位置}]$ 。
2.  $ZobristKey^{\wedge}=Zobrist[\text{移動棋子類型}][\text{移動棋子新位置}]$ 。
3. 如果著法是吃子著法的話，需要再做： $ZobristKey^{\wedge}=Zobrist[\text{被吃棋子類型}][\text{被吃棋子位置}]$ 。

由於二維的  $Zobrist$  Hash 並沒有附帶玩家資訊，而為了避免相同的盤面在不同的玩家方有一樣的 key 值的情況，我們還需要一個大隨機數  $ZobristPlayer$  來代表輪走方，每走一步就做  $ZobristKey^{\wedge}=ZobristPlayer$  運算，最後的  $ZobristKey$  才是真正代表盤面且加上玩家方訊息的 key 值。

由於實際記憶體空間有限，而搜尋的節點又很多，所以我們不可能替 Hash Table 設置一個無限大的空間供作使用，因此我們必須對所產生的節點是否存入 Hash Table 做些限制，避免一些無用的節點資訊佔據了 Hash Table 的空間，而這些無用的節點通常是一些距離根節點很遠的節點，因為它們被重複搜尋的機率很低。

在 Alpha-Beta 搜尋演算法的過程中，任何節點的分數都是下列三種情況之一：

- (一) 節點分數  $\geq \beta$ ，即所謂的  $\beta$  節點。
- (二) 節點分數  $\leq \alpha$ ，即所謂的  $\alpha$  節點。

(三)  $\alpha < \text{節點分數} < \beta$ ，即所謂的 PV (Principal Variation) 節點。

一般來說，只有 PV 節點的分數，才可以當作是節點的準確值存入 Hash Table，其餘的  $\beta$  節點、 $\alpha$  節點的分數只能表示是節點分數的一個邊界而已。但存入這樣的邊界分數，仍有助於我們下次搜尋到同樣盤面節點的時候，進行截斷的動作。所以既然三種節點的分數都可以存入 Hash Table，我們也就必須在存入分數的同時，把該分數所代表的意義，也就是節點類型給存下來。當目前節點探測 Hash Table 成功時，如果已存入的節點類型是 PV 的話，則直接返回節點分數，如果已存入節點類型是  $\beta$  或  $\alpha$  的話，需要進行邊界值的比較，決定是否能截斷。

由於 Hash Table 是有限的大小，故採用“深度優先覆蓋”的策略，所謂的“深度優先覆蓋”即待存入 Hash Table 的節點的深度必須大於或等於已存入 Hash Table 節點的深度才覆蓋。

表 2-1 是 Zobrist Hash 的以相同盤面，作兩次測試的結果。

深度	測試一	測試二
四	Evaluated Times:3006 Hashed Times:0	Evaluated Times:3006 Hashed Times:0
六	Evaluated Times:50164 Hashed Times:573	Evaluated Times:50173 Hashed Times:564
八	Evaluated Times:1082276 Hashed Times:39071	Evaluated Times:1077848 Hashed Times:36263
十	Evaluated Times:11731256 Hashed Times:653625	Evaluated Times:15936053 Hashed Times:843083

表 2-1 Zobrist Hash 以相同盤面作兩次測試的結果表

由表 2-1 可以知道，搜尋深度夠深，Zobrist Hash 的作用才會比較明顯。

#### 第四節 允許空步

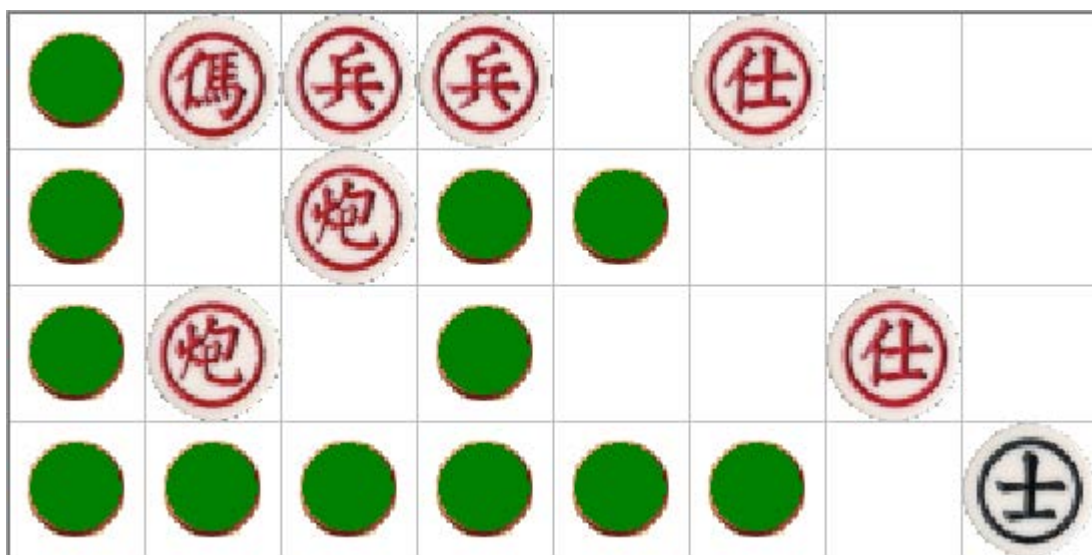


圖 2-15 允許空步例 1

在圖 2-15 的盤面中，無論黑方如何走步，都會只有損失一隻士的結果。允許空步是為了讓搜尋可以進行下去，當搜尋的結果一直都是負分，就讓黑方不走

步，讓搜尋走下去。若等到不走步的結果比走步好，則表示：是翻棋的時候了。

允許空步可以在搜尋的任何一層應用。

如圖 2-16。若為黑方先著手，則因為移開將後，並沒有不損失子的走步了，所以在第三層開始，走空步，直到搜尋結束，得到走將是最好的走步。



圖 2-16 允許空步例 2

若為紅方先著手，則因為黑方可以移將，導致無子可吃，故在選擇走步時，還是以先吃將為最大利益。

## 第五節 Iterative Deepening Search

如圖 2-7 所示，由 Knuth 所證明的，若對 Type 1 節點作走步排序，好的走步排在前面，則對搜尋時的截斷是很有幫助的。又由於在產生走步時，並不知道哪一走步比較有利，所以採取 Iterative Deepening Search，先對當前盤面作四層，即各推兩手的搜尋，將搜尋結果提供給根節點，用以對根節點中的各走步作排序。最後以排序後的走步，作為更深層的搜尋使用，以提高深層搜尋的效率。

# 第三章 我們對審局評分的作法

## 第一節 翻棋位置的選擇

### 1. 統計的方式

暗棋棋子共 32 子，紅、黑雙方各 16 子，其中兵、卒為 5 子，帥、將為 1 子

以外，其餘均為 2 子。如圖 3-1 所示。












子數	5	2	2	2	2	2	1
棋子							
							

圖 3-1 棋子種類及子數

所以開局第一手，翻出為兵或卒的機率為  $5/32$ 。若第一子未翻出兵或卒，則第二手翻出兵或卒的機率就變成  $5/31$ 。

安全分：為盤面上每個未翻開的棋子，求出翻出不能吃周圍的我方棋子並且不被周圍的對方棋子吃的機率。



圖 3-2 機率分數計算範例盤面

如圖 3-2，各位置的安全的棋子數（紅色格子中的棋子）

位置 1：

黑方翻棋：只有黑將或紅帥 2 種棋子為安全的，所以安全分為  $4/27=0.148$ 。



圖 3-3 位置 1 的黑方安全棋子

紅方翻棋：10 種棋子，共(4 卒+2 包+2 馬+1 車+2 象+2 炮+1 馮+2 俥+1 相+1 仕)

18 子，所以安全分為  $18/27=0.666$ 。



圖 3-4 位置 1 的紅方安全棋子

位置 2：

黑方翻棋：14/27=0.518。



圖 3-5 位置 2 的黑方安全棋子

紅方翻棋：24/27=0.888。



圖 3-6 位置 2 的紅方安全棋子

位置 3：

黑方翻棋：(1+5+2+1)=9， $9/27=0.333$ 。



圖 3-7 位置 3 的黑方安全棋子  
紅方翻棋：14/27=0.518。



圖 3-8 位置 3 的紅方安全棋子  
位置 4：

黑方翻棋：22/27=0.814。



圖 3-9 位置 4 的黑方安全棋子  
紅方翻棋：17/27=0.629。



圖 3-10 位置 4 的紅方安全棋子  
位置 5：

黑方翻棋：16/27=0.592。



圖 3-11 位置 5 的黑方安全棋子  
紅方翻棋：22/27=0.814。



圖 3-12 位置 5 的紅方安全棋子

位置 6：

黑方翻棋：16/27=0.592。



圖 3-13 位置 6 的黑方安全棋子

紅方翻棋：19/27=0.703。



圖 3-14 位置 6 的紅方安全棋子

位置 7：

黑方翻棋：7/27=0.259。



圖 3-15 位置 7 的黑方安全棋子

紅方翻棋：13/27=0.481。



圖 3-16 位置 7 的紅方安全棋子

位置 8：

黑方翻棋：2/27=0.074。



圖 3-17 位置 8 的黑方安全棋子

紅方翻棋：共 4 卒+2 包+1 相+1 仕=8 子，8/27=0.296。



圖 3-18 位置 8 的紅方安全棋子

位置 9：

黑方翻棋： $11/27=0.407$ 。



圖 3-19 位置 9 的黑方安全棋子

紅方翻棋： $9/27=0.333$ 。



圖 3-20 位置 9 的紅方安全棋子

對上述位置所得之分數，再乘以 1000 再取整數部分即為翻棋位置的安全分。

如圖 3-2 中沒標數字的位置為絕對安全的位置，因為不管翻出什麼樣的棋子，都不會有任何的棋子的損失。所以可以給個固定的分數，表示絕對安全分，例如：2000 分。

## 2. 經驗法則

### 2.1 期待翻出炮／包

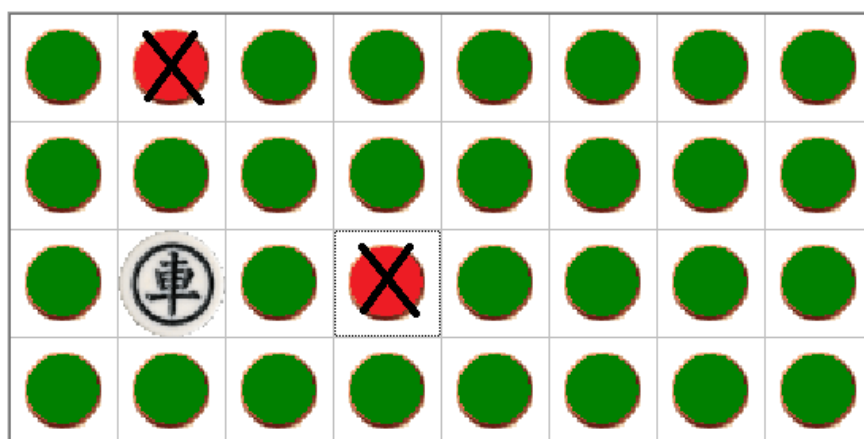


圖 3-21 期待翻出炮的位置

圖 3-21 中 X 的位置為紅方翻棋的好位置，其用意為期待這兩個位置能翻出紅

炮。

### 2.2 期待吃炮／包

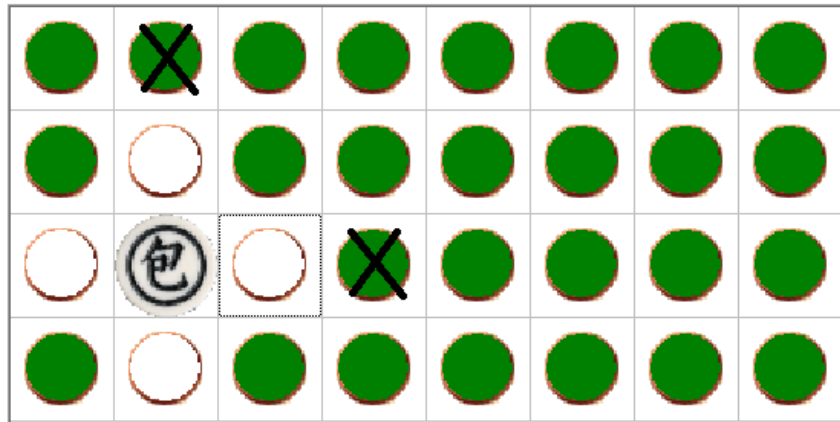


圖 3-22 期待被包吃的位置及吃包的位置

圖 3-22 中紅方選擇翻出的好點為白色的四點位置，期待能翻出紅馬或更大子力的子來吃包。但有 X 位置，則因為是包的攻擊位，所以若為紅方，則必定不翻此位置。反之黑方則可選擇翻此位置，因為只有  $2/31=0.064$  的機會會被吃掉。

### 2.3 對炮／包期待反制

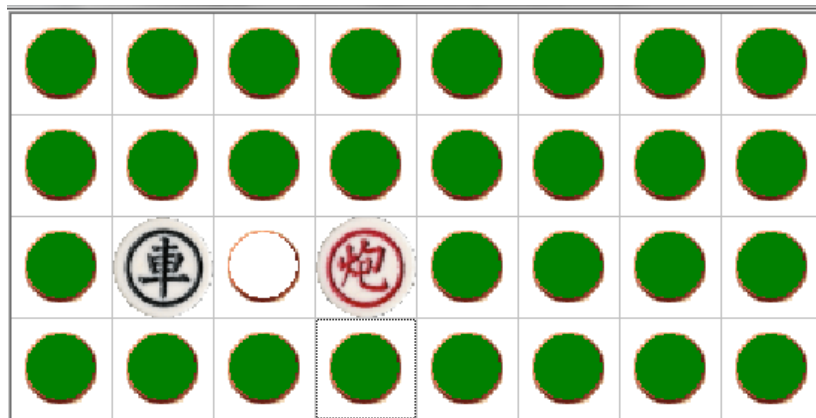


圖 3-23 對紅炮的反制點

圖 3-23 中遇到對手翻出紅炮，黑方則選擇翻白圈的位置，期待能翻出一黑馬或更大的子力，哪麼就算黑車被吃，也可以吃回紅炮。

以上經驗法測的位置，可以配合統計的方式進行加分，以保證在最有利的位置

置翻棋。

如圖 3-24。



圖 3-24 期待炮/包位置範例

白圈為對紅方有利的翻棋點，由於紅方的炮未被翻出，故白圈的位置可以再多加翻出紅炮的分數（炮的機率 $\times$ 1000），作為積極的加分。

有 **X** 的位置黑點則為對黑方有利的加分位置，處理方式同白圈。

最後，為針對有 **X** 的白圈位置，該位置雖為安全位，但不管紅方或黑方，若翻出對手的炮則會對我方的棋子不利，所以該位置可以扣分，以表示對我方不利的位置。

## 第二節 距離威脅力計算

計算距離威脅力的目的有二，

1. 為了解走子步是否有意義，如盤面的狀況是否為連通。
2. 為讓棋子有目標可追，且追有價值的棋步。
3. 為避凶，遠離有威脅的子力。

而根據經驗上的看法，可歸納下列的規則。

1. 未翻開的棋子當然為障礙物。
2. 若要到達目標棋子需通過的對方的棋子。但該棋子的子力較大，則視為障礙物。



圖 3-25 距離威脅盤面

如圖 3-25 所示，由於黑士要通過紅仕才能吃到紅兵，而在靠近紅仕時，會被紅仕吃掉，故不會認為黑士對紅兵產生影響力。

3. 目標棋子的子力等於本身子力者，具有影響力。同上圖 3-25，但黑士可以靠近紅仕，以阻止紅仕的活動，所以對紅仕依然有影響力。
4. 控制域：棋子的攻擊範圍。



圖 3-26 控制域

如圖 3-26，紅仕的上方及左方為紅仕的控制範圍，在下棋時，棋手會知道若黑士想吃紅兵，則需繞過暗棋，才有辦法吃到兵。

#### 5. 使用 best fit search。

使用 best fit search 的方式來計算距離，以求得更精確的距離，以便精確計算威脅分。

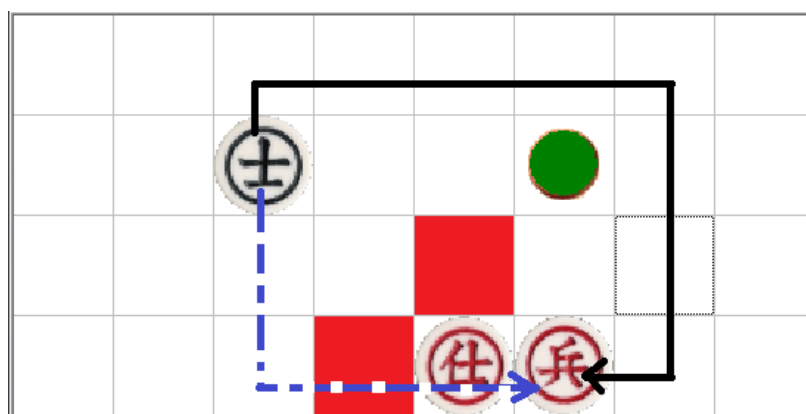


圖 3-27 best fit search 距離

如圖 3-27，黑士到紅兵的距離使用 Manhattan distance 的距離為 5（虛線條的距離），但從人類的認知上，這並不正確，並不會有人認為黑士能通過紅仕而壓迫到紅兵，故黑士到紅兵的距離，認知上並不會只有 5 步。而使用上述的規則及 best fit search 我得到距離為 9（實線條的距離），這樣為實際走步可以走到的距離為人類認知上的距離，使影響力計算更為精確。

以圖 3-27 為例，其 best fit search 的執行過程如圖 3-28：（棋子旁的數字為該格子到紅兵的 Manhattan distance）



1		2	
3		4	
5		6	
7		8	
9		10	

1			6		4	3	4	
	6	士	4/4	士	士	士	4	
	5	士	士	士	●	士	3	
	4	士	士	仕	兵	1		
12			6		4	3	4	
	6	士	4/4	士	士	士	4	
	5	士	士	士	●	士	3	
	4	士	士	仕	兵	士		

圖 3-28 best fit search 的執行過程

best fit search 跟控制域的整合：

1. 產生控制域：將盤面上對手的棋子產生其對應的控制域，按照所在位置產生四個方向的控制域如圖 3-29 左方所示，採取大子力者蓋過小子力的方式，如圖 3-29 右方所示，填入控制域該有的子力，但不對炮作處理。就可以得到一張控制域的表格。

綠色棋子為未翻棋子。

白底棋子為一般棋子。

紅底棋子為該子的控制域。

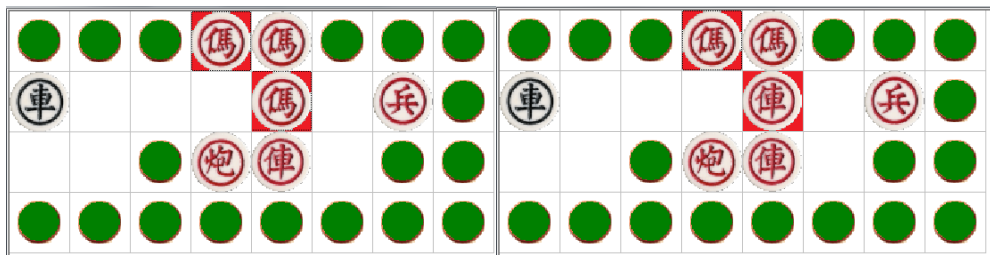


圖 3-29 控制域產生

2. 炮/包動態控制域處理：



圖 3-30 炮/包動態控制域處理

如圖 3-30，若照上述的方式處理，則會表示，紅相對紅俥旁的黑卒有影響力。

情形 1：包架棋，如圖 3-31。



圖 3-31 炮/包動態控制域處理-包架移動

但實際結果是若紅相離開了位置，則變成紅俥為砲架，而紅相又只能由上往下走去吃卒，故必被黑包攻擊到才能到達卒的位置。所以紅相對黑卒不應有影響力。

情形 2：包口棋，如圖 3-32。看起來，中間的紅俥對黑馬的影響力距離應該

是 2，但實際上中間的紅俾位於黑包的攻擊線上，所以不可能走直線去壓迫黑包。

故實際情形只能是如圖 3-33 的迂迴前進。



圖 3-32 包口棋的預想狀況



圖 3-33 包口棋的實際狀況

情形 3：通過包後的壓迫，如圖 3-34。



圖 3-34 通過包後的壓迫

若採取一般棋子的控制域產生方式來產生包的攻擊線，如圖 3-35。



圖 3-35 包的靜態產生的控制域

打 X 的位置為包的攻擊線，若照前文所說的 best fit search 的方式執行，則造成紅相對左上角的黑包無影響力，但實際上人類手並不會這樣子看待。

距離分比例計算：

由於暗棋有吃子距離的特性，若紅黑雙方棋子各只有一顆且其的距離為偶數，則走子方的強子無法捉到對手的弱子，若為奇數距離，則走子方的強子可以捉到對手弱子。

由於以上的結果，所以在距離分數設定上，採取奇數距離的分數增加比較高，偶數距離的分數增加比較小，以使棋子能抓準奇偶距離的吃子步。

### 第三節 動態分數調整

暗棋的大小順序如下，如圖 3-36。



圖 3-36 暗棋的子力大小

暗棋吃子的大小順序如下：

以下為紅方對黑方的吃子關係：

帥：將、士、象、車、馬、包。

仕：士、象、車、馬、包、卒。

相：象、車、馬、包、卒。

俥：車、馬、包、卒。

馮：馬、包、卒。

炮隔一子可以吃掉所有黑方的棋子。

兵：將、卒。

以下為黑方對紅方的吃子關係：

將：帥、仕、相、俥、馮、炮。

士：仕、相、俥、馮、炮、兵。

象：相、俥、馮、炮、兵。

車：俥、馮、炮、兵。

馬：馮、炮、兵。

包隔一子可以吃掉所有紅方的棋子。

卒：帥、兵。

暗棋有絕對的大小關係，除炮／包以外，小的不能吃大的。

而當沒有更小的棋子時，該子就可以放棄，因為留著該子的用途並不是很大。

而兵／卒則採取不同的處理方式，由於它是唯一可以克將／帥的棋子，所以當兵／卒的數量在減少時而對方的將／帥還存在時，可以提高兵／卒的分數，以保兵／卒用來克將／帥。

有了以上的想法，在實作上，在每個吃子步發生後，都計算一次吃該子的得分，並將得分傳給審局函數作為計算分數的依據之一。依我方吃子為正分，對方吃子為負分的計算。

另外，暗棋由於玩法簡單，在盤面各子都是已知時，有一個簡單的規律。只要符合以下圖 3-37 至圖 3-39 條件，一般上來說都保證贏棋。

- 我方最大子大於對方最大子，且次大子也大於對方最大子。

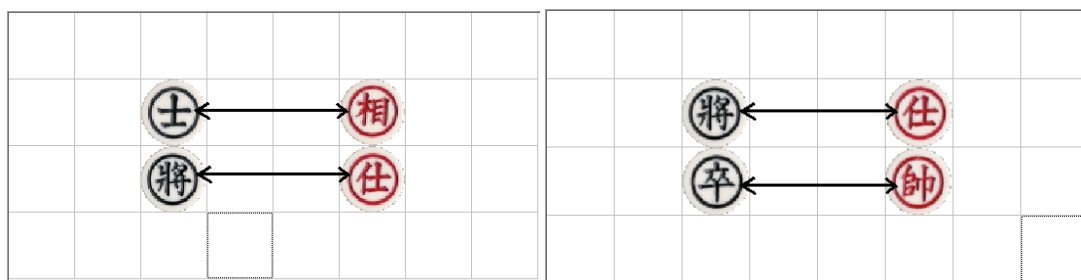


圖 3-37 贏棋規律 1

- 我方最大子大於對方最大子，且次大子等於對方最大子。

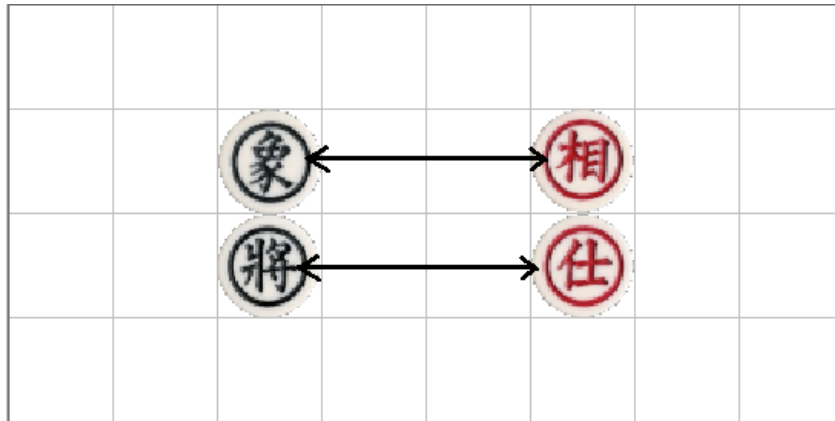


圖 3-38 贏棋規律 2

- 我方最大子等於對方最大子，但次大子大於對方最大子。

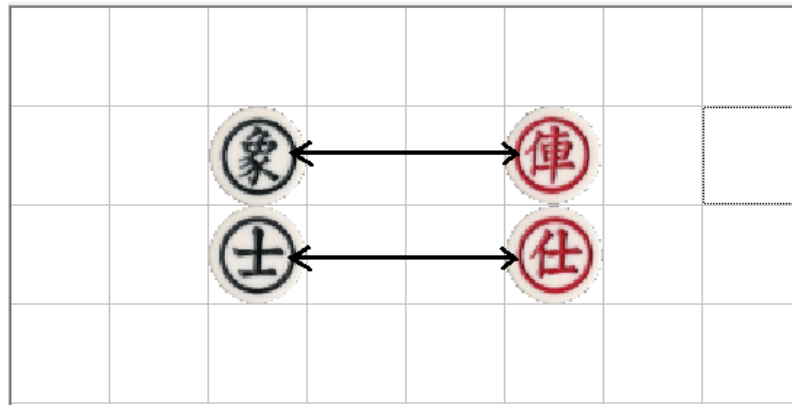


圖 3-39 贏棋規律 3

有了以上的規律後，採取的方式為，先吃掉對方次大子而又能保存我方次大子或兒子但讓我方最大子跟對方最大子保持偶數距離。

#### 第四節 配分設計

棋盤的位置分設計，如表 3-2 所示，這樣子的安排為每位置的可移動的方向的個數。



2	3	3	3	3	3	3	2
3	4	4	4	4	4	4	3
3	4	4	4	4	4	4	3
2	3	3	3	3	3	3	2

表 3-2 棋盤位置配分

距離分的配置設計方法的說明如下。暗棋的走步距離最長為 20，如圖 3-40 所示。

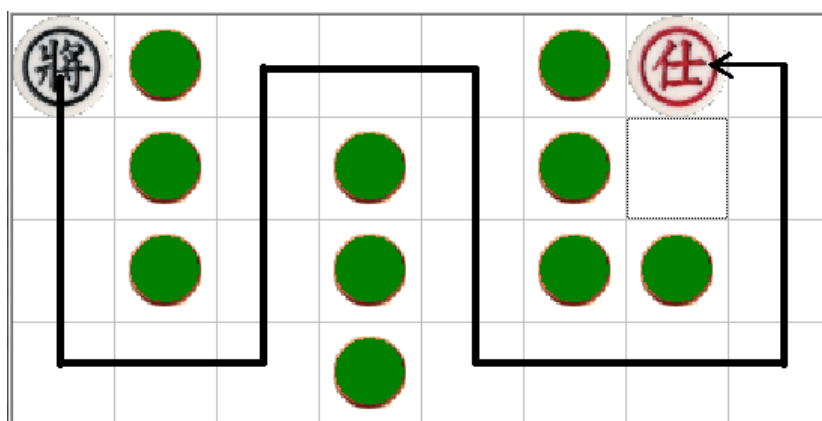


圖 3-40 最長走步距離

所以在距離分數的配置設計上，準備一個長度為 20 的 array 儲存各距離的得分。而各距離的得分採取漸進式。

另外，由於距離影響力對所有的棋子的距離分數都固定，所以距離分再加上目標子的棋子種類，則可以讓距離對較大的目標子的影響力增加，使之先追擊對方棋子中子力較大者。

另外，距離分的設計要跟棋盤的位置分不同，以區分得分的來源。

棋子配分設計方法：

例如，兵的配分為 1000 分，則馬的配分為 2000 分，每上一級都為前一級的兩倍，以保證不會亂兒子，畢竟兩隻馬也比不上一隻車。

我們現在的棋子配分如表 3-3。

棋子	配分
卒/兵	200
包/炮	420
馬/馮	380
車/俥	761
象/相	1523
士/仕	3047
將/帥	6095

表 3-3

## 第五節 審局函數設計

由於暗棋為不完全資訊的遊戲，故審局函數的設計上，採取累進的方式。在進行搜尋前，先對現在的盤面作審局，得到一個當前的分數，在搜尋到葉節點時作審局後，將該分數減去搜尋前的分數加上搜尋過程中吃子步所得到的分數，便是這一個分枝的審局分。

舉例，如圖 3-41 為初始盤面，為黑方著手。先對盤面進行審局，黑方現在盤

面的得分為 5139 分。現舉例，只搜尋走步不翻棋的其中一個分枝如下：黑將右一、紅俾左一、黑將下一、紅兵右一（吃黑將）、黑車上一（吃紅兵）、紅俾左一（吃黑車）。

經過上述的走步搜尋，吃子的得分為-6095 損失黑將，+200 吃一紅兵，但因動態調節，在黑將已歿時紅兵的分數打折，故實得+50，最後再損失一黑車-761。而最終盤面審局得分（黑 200 分，紅 200+761+420）為-1181，本分支的得分為：  
 $-1181-5139+(-6095+50-761) = -6806$  分。

## 第六節 走步或翻棋的選擇方法

翻棋走步的產生，分成兩類

1. 空步：只要盤面上還有未翻棋子，則除 root 外，每一層都會產生出。

有這樣的設計，是因為實際遊戲中我們不能不走步，只能走一步棋而且，而使用搜尋的目的是為了走步推演，由於程式的深度為固定，有時會造成錯誤的結果。



圖 3-41 需要翻棋的盤面

如圖 3-41，黑方第一步，在 root 產生黑將右移一步，然後往下的搜尋黑方只套用走空步即可，也就是黑方不走任何棋子。

如果不這樣處理的話，會造成黑方亂走，因為黑方除了黑將右移外，其它任何走步都會產生損失，而且盤面上可走的位置，並不足以應付 6 層的推演，更別說更深的搜尋。

若不允許空步及翻棋的走步，則搜尋的結果將不太合理，例如，黑將右移→紅俾左移，接下來無論黑車上移，或黑將左移或下移，都損失一子。之後再推演一步，則其整個走步必為：黑將右移→紅俾左移→黑車上移→紅俾吃黑車→黑將吃紅俾→紅炮吃黑將為黑方最大的利益，但人類棋手實際並不會這樣的亂走。

2. 翻棋：只在 root 產生翻棋走步，以供模擬翻棋走步搜尋。而 root 以下的節點，則除非只剩二子未翻，否則不產生翻棋走步，而且只對我方的走步產生翻棋走步。

經由安全分的資訊，我們得到模擬的位置，模擬的棋子，則採取，若我方的炮還未被吃光的話，則找比 2000 分還高的位置，以炮為模擬的對象。若炮已被吃光，則採取我方最大可以對手或對方最小可被我吃。如圖 3-2，的位置 2，從圖 3-5 得知，若為黑方翻該位置的棋，只有模擬翻出黑將，才安全且可以繼續搜尋走步下去。而從圖 3-6 得知，若為紅方翻該位置，只模擬翻出黑象或比黑象更小子力者即可。

也由於上述的方式，倘若雙方都互相模擬翻棋，經實測，搜尋的結果往往會

變成翻棋搜尋的結果比走步好太多，而變成大家都想得太美好的情況。

也由於只是模擬的，故所得的分數並不準確，依測試的經驗，將模擬翻棋走步搜尋的得分打 9 折，作為翻棋走步最後的得少，以平行跟實際走步的對比。

最後，在走步和空步或翻棋的選擇上，採用的方法為，若走步分為正分，則不作空步，而在 root 則是若走步分為正分，則不作翻棋模擬，但這有時會造成容易兌子的情形，所以再加入兩個條件，若走步為負分且走步中有非吃子步，則作翻棋模擬。

如圖 3-42，若無上述的規則，則造成紅仕吃黑象，黑將吃紅仕的狀況，主因是因為走步上只有紅仕走黑象。



圖 3-42 兒子的盤面

## 第四章 結論與未來研究方向

### 第一節 結論

TAAI 2010 比賽中，雙方不翻棋而平手的盤面。

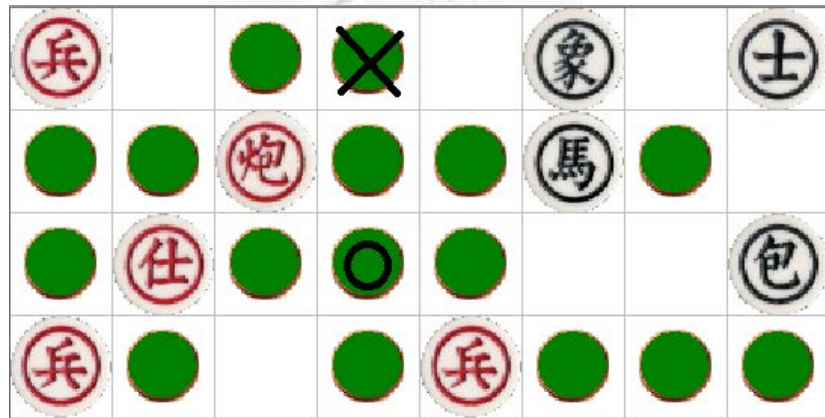


圖 4-1 TAAI 2010 平手盤面 1

如圖 4-1 的盤面，在比賽中，結果為黑包及左上角的紅兵循環走來走去而平手。但若執行本程式，紅方先著手會翻開 **X** 的位置，而黑方先著手就翻開 **○** 的位置因而避免和棋的結局。

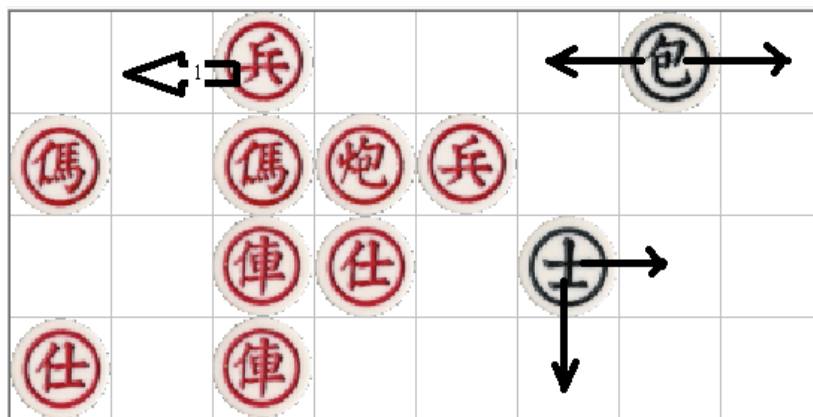


圖 4-2 TAAI 2010 平手盤面 2

如圖 4-2 的盤面，比賽結果為黑士上下來回移動及中間的紅仕上下來回移動

造成循環而平手。本程式會先移動上方的紅兵，將紅兵左移，以開路讓紅馬往上  
 出去。故黑士往下或往右，則紅仕會往右壓迫，黑包不論左移或右移，都使紅馬  
 追到黑包。

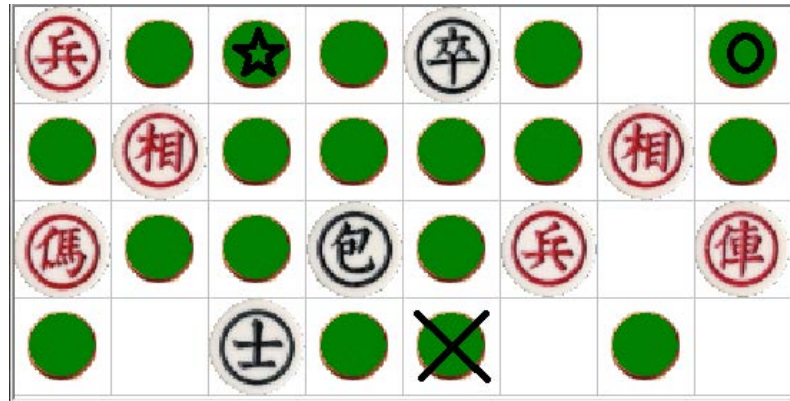


圖 4-3 ICGA 2010 平手盤面 1

圖 4-3 的盤面的比賽結果為黑士左右來回移動及紅相上下回來移動的循環走  
 步而造成和局。本程式紅方會作期待翻紅炮的翻開可以打黑士的  $\times$  位置，若為黑  
 方則翻  $\circ$  的位置。倘若依然翻不出炮/包，且同時翻出對手的棋子，則紅方再翻  
 $\star$  的位置。

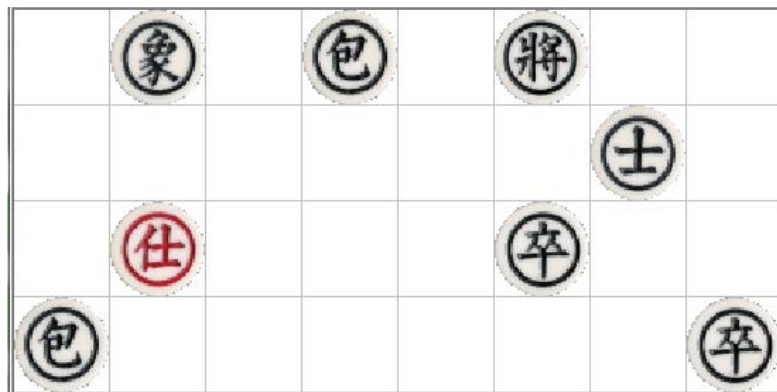


圖 4-4 ICGA 2010 平手盤面 2

圖 4-4 的盤面由於黑方沒有放棄黑包，故平手，但我們的程式會棄包，故黑

方會直接移動將往左下移動去壓迫紅仕而贏得勝利。

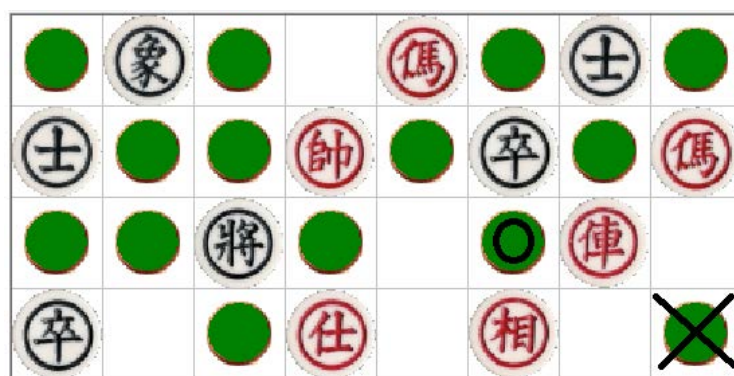


圖 4-5 ICGA 2010 平手盤面 3

圖 4-5 的盤面為黑卒紅仕左右來回移動而平手，但其實  $\times$  及  $\circ$  分別為黑方及紅方很好的包及炮位。所以本程式在黑方著手會翻  $\times$ ，若為紅方翻  $\circ$  位置。

從以上因距離、翻棋及棄子問題測試，驗證了本方法可以解決了無法翻棋及路徑的問題。

表 4-1 為對最後 Dark Chess Beta 及人類玩家的測試結果， Dark Chess Beta 為 ICGA 2010 及 TAAI 2010 比賽的亞軍。

	勝	敗	和
vs Dark Chess Beta	14	11	2
vs 人類玩家	8	16	4

表 4-1 實測結果



## 第二節 未來研究方向

對暗棋的殘局處理，現在有了距離影響力的分數後，棋子已經可以進行催殺了，但仍有很多殘局的狀況，依然沒有辦法跟人類棋手相比，由前面所述，暗棋只要保持至少最大子等於對手最大子，而次大子大於對手即可，但在配分的影響下，導致仍無法完全解決問題。

如圖 4-6。人類棋手會兌子，以士換兵，原因是因為將帥的距離剛好是奇數，則將必然能捉帥。



圖 4-6 將士兵帥的殘局

另外，如圖 4-7。



圖 4-7 三卒二兵的殘局

黑方先走三卒對二兵的情況下，人類棋手可以實施包圍然後兌子，即可輕鬆

獲勝。但本程式則由於雙方的配分及距離影響分的計算仍不足去找出正確的走法而平手。

## 參考著作

- [1] “chessprogramming”，網址：<http://chessprogramming.wikispaces.com/>.
- [2] “象棋百科全書”，網址：<http://www.elephantbase.net/index.htm>.
- [3] 吳身潤，“人工智慧程式設計”，維科圖書，2002年3月。
- [4] 王小春，“人機博奕”，重慶大學出版社，2002年6月。
- [5] 林子哲，“「深象」象棋軟體平行化之研究”，國立臺灣師範大學資訊工程研究所碩士論文，2007。
- [6] 涂志堅，“電腦象棋的設計與實現”，中山大學碩士論文，2004。
- [7] 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。
- [8] 郭哲宇，“電腦象棋擴大空步剪裁演算法的設計及實作”，國立臺灣師範大學資訊工程研究所碩士論文，2007。
- [9] 方裕欽，“UCT 算法的適用性及改進策略研究—以黑白棋為例”，國立臺灣師範大學資訊工程研究所碩士論文，2008。
- [10] 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，台北，2008。
- [11] 賴學誠，電腦暗棋程式與經驗法則之配合與實作，碩士論文，國立東華大學資訊工程研究所，台東，2008。
- [12] 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。
- [13] 陳柏年，“電腦對局知識取得與應用”，國立臺灣大學資訊工程研究所博士論文，2011。