國立臺灣大學電機資訊學院資訊工程學系

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering & Computer Science

National Taiwan University

master thesis

# 遊戲演算法在撲克牌遊戲-大老二上的應用與比較

# The game theory's application and comparison on pokergame Bigtwo

黃源鱗

Yuan-Lin Huang

指導教授：呂育道 教授

Dr. Yuh-Dauh Lyuu

中華民國 98 年 6 月

June 2009

# 國立臺灣大學碩士學位論文

# 口試委員會審定書

論文中文題目
論文英文題目

本論文係○○○君（○學號○）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國○○年○○月○○日承下列考試委員審查通過及口試及格，特此證明

口試委員：

_____

（指導教授）

_____ _____

_____ _____

_____ _____

_____ _____

系 主 任 _____

# 致謝

感謝徐讚昇老師開的課「電腦對局理論」啟發提供了我論文的方向

感謝指導老師呂育道的同意與支持讓我可以研究這個題目

感謝我的家人的贊助不然我不會來念研究所

感謝我的朋友、同學給予我鼓勵和支持讓我能夠順利完成這論文

# 中文摘要

遊戲演算法在電腦人工智慧中一直扮演著重要的角色之一，藉由嘗試讓電腦更有智慧的玩遊戲，探討何謂人類的智慧。遊戲演算法的研究從早期的雙人遊戲、完全資訊遊戲，延伸到多人遊戲、不完全資訊遊戲，有各式各樣的演算法。本文使用其中一些演算法，套用在撲克牌遊戲-大老二，比較其優劣並加以討論。

關鍵字：

遊戲演算法，人工智慧，多人遊戲，$max^n$演算法，撲克牌遊戲-大老二

# Abstract

Game algorithms play an important role of artificial intelligence, we try to find human intelligence by making computer play games more wisely. Researches of game algorithms from earlier two-player games and perfect information games extend to multi-player games and imperfect information games. There are many kinds of game algorithms. In this thesis, we apply some of these algorithms to the pokergame Bigtwo. We also analyze the various algorithms.

Key words:

Game algorithm, artificial intelligence, multi-player game, $max^n$ algorithm, pokergame Bigtwo

# 目　錄

# 圖目錄



# 表目錄



# 第一章　簡介

人工智慧在電腦研究的領域上一直是熱門的研究之一，學者們都致力於找尋如何讓電腦行動得更貼近人類的行為的方法，希望能讓電腦更有智慧的幫助人類，而遊戲電腦 AI 的研究就是其中一個研究的方向。藉由如何讓電腦玩遊戲玩得厲害，和人類相比又是如何，來探討人類的智慧與機器的智慧的異同。

遊戲演算法從最早期 brute-force 純暴力搜尋，到後來的 alpha-beta 改良搜尋法以及一些規則化的定義，都使得電腦在玩遊戲上越來越強，甚至凌駕於人類之上。1997 年西洋棋機器人-深藍(Deep Blue)打敗了世界棋王更可說是登上了一時的高峰。

而許多早期遊戲演算法主要是建立在雙人完全資訊的遊戲上(two-player perfect information game)，但也有越來越多延伸的演算法開始著眼於多人遊戲(multi-player game) ，以及非完全資訊遊戲(imperfect information game)。

本文嘗試將截至目前為止出現的一些演算法，套用在多人的撲克牌遊戲-大老二上，並比較其優劣，期望能找出改進演算法的方向。

在第二章裡，會介紹一些從以前到現在所出現遊戲演算法的相關研究。第三章說明撲克牌遊戲-大老二的遊戲規則及玩法。第四章是實驗的設置及數據。第五章是討論及結論。

# 第二章　遊戲演算法的歷史

## 2.1 最早期的遊戲研究

一般遊戲演算法的學者認為，最早開始創立有系統的遊戲演算法的始祖是夏農於 1949 年所發表的文章［8］。其為一篇研究如何使用程式下西洋棋的文章，此篇文章不僅僅探討了如何下西洋棋，更點出了所有遊戲演算法所應該考量的各項要素。

首先，一個棋局可以被細分為許多資訊來代表現在的盤面，例如棋子的位置與個數、輪到黑子或白子、截至目前盤面為止的棋子走法…等等。對於兩個玩家來說，這些資訊是完全公開的(perfect information)。

既然資料是完全公開的，那麼我們就能針對對手下一步的每一種可能性做搜尋。若是他走這一手，輪到我們時我們有那些步數可以選擇，接著他又可以如何走下一步，最後找出一定對自己有利的步數。理論上這是辦得到的，但事實上要計算出所有的步法，我們會需要計算大約有 $10^{120}$種變化，即使每一步我們只花一毫秒($10^{-6}$s)，我們仍需要 $10^{90}$年才能計算完下一步該如何走。另一種方法是算出所有可能的盤面，藉此找出如何進行到所謂「勝利」的盤面，而所有的盤面總共有 $64!/ 32!(8!)^2(2!)^6$ 種，將近 $10^{43}$種，所以這也是

不切實際的。

既然不可能列出所有的可能，那麼我們的策略是，盡可能的從目前的情勢中找出好的下一步。在此他也提出了一項非常重要的設定，那就是審局函數(evaluation function)。審局函數的功用，在於判斷目前盤面兩方勢力的優劣，例如棋子剩越多的一方分數應該比較高、持有不被阻擋的皇后(或城堡、主教)應該有較高的分數等等。在策略步法搜尋裡，審局函數扮演的角色就是，當我們決定走這一步到下一個盤面時，我們如何知道這一步對我們來說是好的還是壞的？此時就要靠審局函數來判斷了。

有了審局函數之後，策略的搜尋就變成一種 max-min 的搜尋。當輪到我們行動時，我們要尋找如何最佳化我們的分數的步法(max)，反之當輪到對手行動時，他會尋找如何最佳化他的分數的步法，相對來說即是使我們分數最低的步法(min)。

對所有可能的步法做搜尋的方式，我們稱之為 Type-A。事實上這樣的演算法是非常沒有效率，執行出來的電腦強度是非常的弱的，因為在有限的時間內他為了搜尋所有可能的步法，使得他所能探索的深度是非常淺的，他只能判斷接下來的兩三步盤面局勢的變化。

一個好的人類玩家也不會對所有可能步法做延伸考量，他只會思考其中幾種步法。但甚麼是好的步法，一個比較簡單的判別例子就是：可以吃掉對方棋子的步法應該是好的。針對這些好的步法做搜尋，我們稱之為 Type-B。

挑一些看來比較重要的步走下去，可以節省很多時間，增加效率。

最後，我們應該避免每次在同樣的盤面下，讓電腦做完全相同的決定。因為就最後結果來說，這一步未必是最正確的選擇，甚至可能是導致失敗的原因。我們可以增加一些隨機變數，使電腦在做決定時不會每次搜尋的順序和選擇的方式都相同。

## 2.2 雙人完全資訊遊戲的研究

有了基本的系統架構之後，後來研究遊戲的學者們，都在想辦法使程式的效能提高。大致上的研究方向是：使用更精確的審局函數，使用更快速搜尋的演算法。其中有個最泛用，幾乎必備的一個演算法就是 alpha-beta pruning [5]。

Alpha-beta pruning 是一個基於 min-max 搜尋延伸出來加快搜尋效率的一個演算法，其概要是盡量減少需要搜尋的節點數目，對於不會影響到搜尋結果的節點就先剪掉不去做搜尋。它主要分成兩種剪裁法: alpha 剪裁 & beta 剪裁 。

Alpha 剪裁：產生於 max node，假設在 max node 的第一個分支搜尋完後得到了一個 value，接著蒐尋第二個分支，則第一個分支的 value 可以形成一個做為判別的 bound。當分支 2.1 傳回一個值＜bound，那麼接下來分支 2.2 及之後的分支就沒必要做搜尋了，因為分支 2 的值至少會＜bound (min

node)。對分支 1 來說，它的值至少會＞bound，所以 2.2 以後搜尋出來的值都是沒有意義的。詳見圖一。(白色節點代表 max node，黑色節點代表 min node 。)

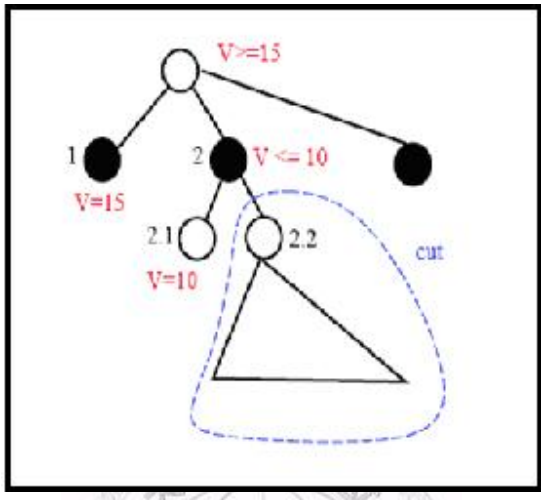


圖 1 alpha pruning。

Beta 剪裁：產生於 min node，假設在 min node 的第一個分支搜尋完後得到了一個 value，接著蒐尋第二個分支，則第一個分支的 value 可以形成一個做為判別的 bound。當分支 1.2.1 傳回一個值＞bound，那麼接下來分支 1.2.2 及之後的分支就沒必要做搜尋了，因為分支 1.2 的值至少會＞bound (max node)。對分支 1 來說，它的值至少會＜bound，所以 1.2.2 以後搜尋出來的值都是沒有意義的。詳見圖二。

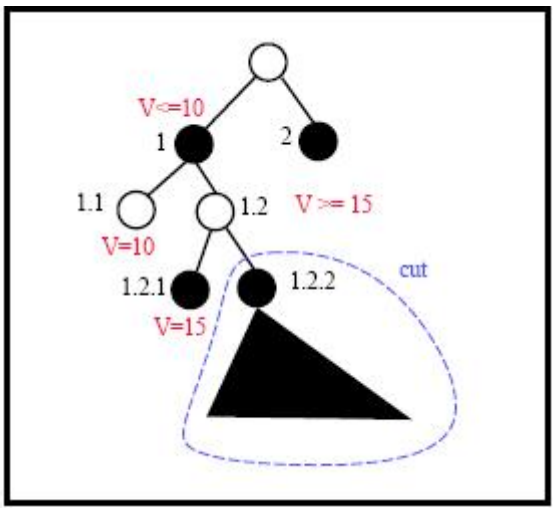


圖 2 beta pruning。

在學會 alpha-beta pruning 後，我們開始研究如何讓一棵 game tree 所需搜尋的節點數減到最少，也就是能夠做最多的剪裁。根據研究我們發現，若是一棵樹它的排列夠好的話，它就可以做最多的剪裁。甚麼是一個好的排列？就是盡量讓會有比較好的結果的節點做為分支中的第一個節點。我們從上面的定義中可以看出，若是在第一個節點中我們就找到一個很好的值，那在搜尋其後的分支時，我們常常就可以剪掉一些不必要搜尋的節點。於是，如何做一個好的排列的搜尋樹(good move ordering)，就成為往後研究的重點之一。

其實這跟我們平常下棋或打牌時的思考模式是非常類似的，通常我們在思考下一步該如何走時，一定會先挑一些看起來比較有利可圖的步法去想下一步之後的局勢。例如下了這一步可以威脅對方的 king，就會優先考慮。而

比較不重要的步法，往往擺在最後才去思考其可能性。例如走了一步對棋局無關緊要的士兵。

除了如何減少節點增加搜尋效率以外，也有些人專注於不同方向的研究上，像是對手模組(opponent model)[2]。其探討的是如何根據對手不同，建立更有效的策略搜尋。好比說有些人下象棋時很會用馬，那麼也許如何吃掉他的馬會是比較重要的策略。有關 opponent model 的建立也有許多分支，強化學習法(reinforcement learning)[10]就是其中一種，主要都是根據不停的 training 來調動分數，使策略最佳化。

近來還有一個十分嶄新的演算法，叫做 monte-carlo [1]。此演算法除了可以應用在遊戲策略搜尋上，也可以廣泛地運用在其他例如金融計算上面，其主體大致上是利用大量模擬數據做統計，藉由大數理論，找到一個趨近於最佳解的解。套用在遊戲樹上，它有一個完全不同於夏農的地方，就是它不需要審局函數。它樹上節點的價值是經由模擬到終點時，看結果是贏或輸(或一個值)，然後再將結果往回傳到每一個經過的節點做累積。

而隨著電腦連結速度變快，以及多核心漸漸普及，平行演算法也開始被應用結合到其他演算法裡[3]，都是為了能更提升程式計算的速度。

## 2.3 多人遊戲的研究

除了雙人遊戲外，也越來越多人開始投入研究多人遊戲的領域，多人遊

戲有許多不同於雙人遊戲的挑戰[9]，無法完全套用雙人遊戲的經驗在其上。

首先光是max-min search，就不是可以直接套用的演算法，因為此時玩家已經變成 3 人 4 人。因此有人提出了$max^n$ algorithm[6]，其概念跟max-min相當類似。

首先取代了原本節點上只有單一的值，每個節點上現在改成有一個N-tuple 值的陣列，第 i-tuple 的值就代表著玩家 i 的分數。根據 max 節點的原則，在玩家 1 的節點上，對其子節點做搜尋時，就挑選使玩家 1 有最高分數的節點(tuple-1 分數最高的)。其他節點則依此類推。如圖三。

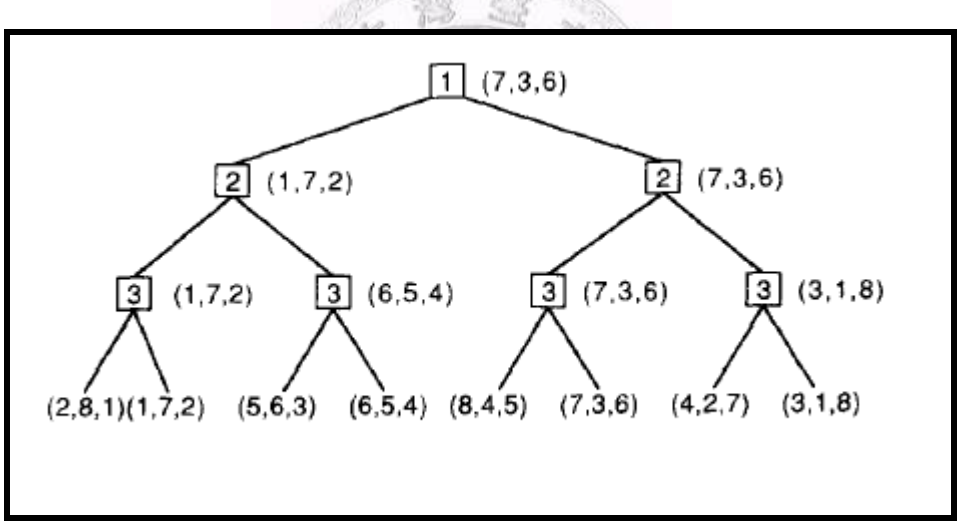


圖 3 $max^n$。

$Max^n$ 也有所謂的alpha-beta剪裁法，條件是基於tuple值的總和須固定(上限)以及其值至少要大於 0(下限)。那麼假設節點a的tuple-1 的值加上節點f的tuple-2 的值已經大於總和，那節點f的其餘子節點是可以忽略不做搜尋的。詳見圖四。

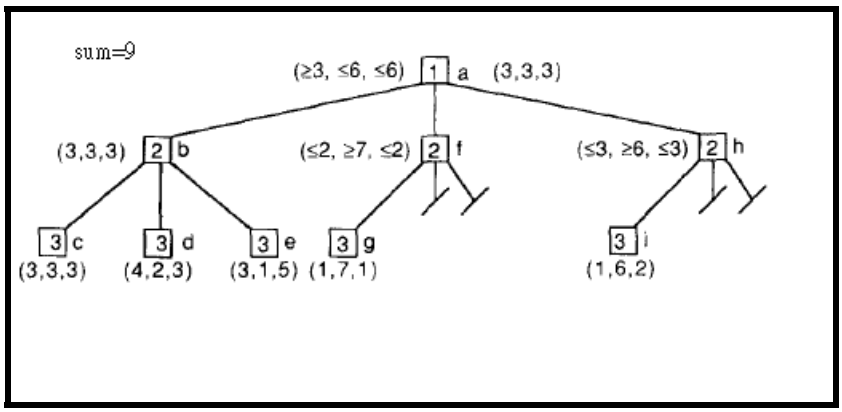


圖 4 max$^n$ pruning。

但是 alpha-beta 到了多人遊戲也是有例外的，其原因有二，一個是即使被剪裁掉的節點仍然有可能影響到整棵遊戲樹的值；其二在於平衡點取捨的不同，會造成整個遊戲樹的不同。其一，詳見圖五，若根據 alpha-beta 剪裁，f 應該是要被剪掉的，但是依據 f 的值的不同，整棵樹的結果會不同。

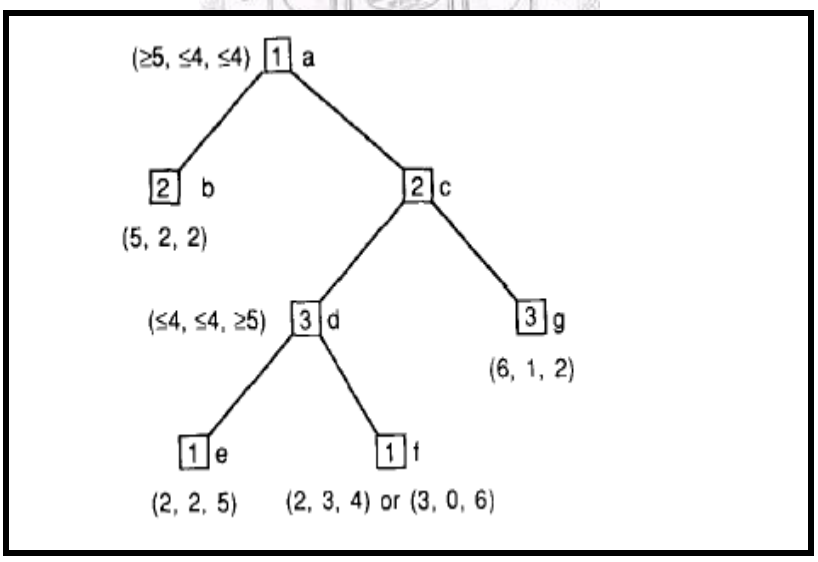


圖 5 max$^n$ exception。

其二，所謂的平衡點是指當某一節點向下搜尋其子節點時，遇到兩個(或以上)子節點同分的情況。在雙人遊戲時，因為總和相同又只有兩個玩家，所

以在其中一個玩家的分數相同的情況下，另一個玩家的分數也會相同。但到了多人遊戲，一個玩家分數相同，其餘的玩家分數卻不一定相同。假設玩家 3 的節點 a 的子節點其值分別是(3,1,6)和(4,0,6)，對於該節點，兩個的值都同樣使玩家 3 有 6 分，但對於玩家 1 和玩家 2 來說就有差了。所以在平衡點如何挑選節點也是非常需要思考的。

除了搜尋演算法的不同，多人遊戲還有一個常出現而必須考慮的問題，就是不完全資訊(imperfect information)。一般多人遊戲裡，玩家只能知道自己所擁有的牌，以及檯面上出過的牌，但是對手擁有甚麼牌是無法得知的，那麼所謂的搜尋演算法，勢必又必須做一些相對應的更改。使用 Monte Carlo 做模擬延伸推測對手牌組是個還不錯的方法[4]。

# 第三章　大老二的玩法與規則

本章將介紹多人撲克牌遊戲-大老二的玩法。

首先玩家通常是 2-4 人，將一副撲克牌 52 張牌平均分配給每個人，即可準備開始遊戲。

**遊戲的玩法:**

首先開局由持有梅花 3 的玩家負責出牌，他可以從手中打出單張或任何牌組，但一定得包含梅花 3。在他之後的玩家，必須跟著出同樣牌型並且比他的牌大的牌組。若是他無法出比目前出牌大的牌組，或是不想出牌，則 pass 換下一個玩家。若是一個人出牌後，其他三家(假設是四人)若是都 pass，則他可以決定接著要出何種類型的牌組，從他手中打出該牌組，而其他人必須跟著出同樣牌型的牌。如此依序輪流出牌，直到有人將手中的牌全部打出去即為勝利。

**牌的大小:**

花色為黑桃＞紅心＞方塊＞梅花

點數 2＞A＞K＞Q＞…＞3

**遊戲的牌型:**

分為單張、一對、五隻的特殊牌型

- **單張**（Single）：任何一隻手中的牌，出牌時必須大於上家出的一隻牌。例如，♠A 比♥A，♠A 又大於♠K、♠Q 等，以此類推。

- **一對**（Pair）：任何兩隻同數字或字母的牌，出牌時跟單隻一樣，必須大於上家出的牌。例如♥A♠A 是大於♥K♠K。當點數相同時，比較較大的花色。

五張牌組合，級別由低至高分別為：

- **順子**（Straight）：任何五隻牌連續順序但不同花，是五種中最低級別的。有多種計算大小的方法，台灣常見的有：A-2-3-4-5 當作最小，最大的則是 2-3-4-5-6。不可出 J-Q-K-A-2、Q-K-A-2-3、K-A-2-3-4。

- **葫蘆**（Full House）：三條加一對。以三條排大小，「一對」對於等級是無影響的，例如 K-K-K-2-2 和 K-K-K-3-3 在等級上是絕對一樣的，不過此情況不可能出現（因為牌中不會有 6 張相同的）。

- **鐵支**（Four of a Kind）：四隻數字一樣的再加任何一隻牌。跟葫蘆一樣，第五隻牌對等級是毫無作用的。換句話說，4-4-4-4-3 是大於 3-3-3-3-K 的。

- **同花順**（Straight Flush）：五種中最強大的組合。它結合了同花和順的特性，也就是說五隻連續的牌，而五隻都是同一種花。最強的同花順是♠2-3-4-5-6。

- **一條龍** 也就是手中 13 張牌完全沒有重複。不過此種組合機率非常低。

出牌時，必須依照上家打出的牌型出同樣類型的牌，單張就單張，一對就一對，順子就順子，葫蘆就葫蘆。唯鐵支、同花順、一條龍稱之「怪物」，在任何時候皆可出牌。若是上家出了鐵支、同花順，則你只能出比他大的鐵支或同花順(ps. 同花順可以壓鐵支)，一條龍則該玩家已經出完牌直接勝利。

**計分方式:**

當有人出完牌時，則其餘三家每人每有一張手牌算 1 分，總和加到贏的玩家的積分裡。例:玩家一出完牌，玩家二還有 3 張牌、玩家三還有 6 張牌、玩家四還有 2 張牌，則玩家一得 11 分、玩家二 -3 分、玩家三 -6 分、玩家四 -2 分。

# 第四章　實驗設置與數據

接著說明實驗的設置與數據

關於電腦的人工智慧，在此設置了兩種電腦，分為 Type A 和 Type B。

Type A，是簡單的規則化電腦，其出牌的規則就是有牌能出就出，不考慮保留手中的大牌或牌型。總之就是不求贏牌，只求把牌的數量減至最少。此種電腦將做為和另一種電腦相比的基準。

Type B，是使用$max^n$演算法的電腦。在此我們假設我們知道所有對手的牌，如此一來我們才能使用搜尋判斷未來的局勢走向而不是靠假設。既然是搜尋那當然也需要審局函數來判斷當前局勢的分數，大致上根據所剩手牌數目和大牌數目和特殊牌型來計算總分。此外需要一個最大深度的參數，其意義是指往下搜尋到第n次玩家出牌(pass也算出牌)，就停止搜尋回傳分數。

牌局使用一千場相同亂數產生的相同牌局。

表一是四個電腦皆使用 Type A 的比賽的結果

表格 1 四個 Type A 電腦的比賽結果。

| | 電腦一 | 電腦二 | 電腦三 | 電腦四 |
|---|---|---|---|---|
| 分數 | -344 | 22 | 246 | -124 |
| 勝場 | 241 | 250 | 260 | 249 |

由表一中可以看出四個電腦彼此勝率是差不多的,表示勝負應該完全是靠手牌的好壞而定。

表二是將第四家電腦使用Type B ,其餘電腦使用Type A的比賽的結果。(後面括號後的數字表示使用$max^n$ 時搜尋的最大深度。)

表格 2 三個 Type A 和一個 Type B(搜尋深度 6)電腦的比賽結果。

| | 電腦一 | 電腦二 | 電腦三 | 電腦四(6) |
|---|---|---|---|---|
| 分數 | -954 | -208 | -50 | 1212 |
| 勝場 | 213 | 228 | 251 | 308 |

由表中可以看出,在同樣的一千場中,電腦四的勝率跟分數皆大幅提升,可得知Type B的電腦應該比Type A的強度高許多。但若要因此判定$max^n$演算法就是比普通使用固定規則的電腦來得強,也言之過早,因為我們其實還可以在規則化電腦上多加入許多規則來使得它變強,只是會使程式更複雜繁瑣難以理解,在調整改良上也比$max^n$來得困難些。

另外做為實驗數據的比較,我們曾經把電腦四 Type B 的深度調高或調低,但發現結果沒有改變,應該是審局函數設置時僅考慮到保留大牌及牌型,沒有考慮到出牌時若是被其他玩家蓋過的後果,所以表示其他玩家的出牌不太會影響自己的出牌。如此一來,似乎也就失去了搜尋其他玩家出牌的意義。

# 第五章　結論

本次實驗中，我們發現只要簡單的設置審局函數對所有可以出的牌做分數的搜尋來決定策略，其效果即可比普通規則化的電腦的強度好許多。

至於如果要做更深層的搜尋，包括判斷對手會出些甚麼牌，來決定自己要出甚麼牌，則審局函數就會變得十分複雜，需要做更複雜的判斷。此外節點的縮減也將更加重要，因為搜尋得越深，節點的擴增將等比級數的增加，會使搜尋時間爆增。

未來的研究除了可以更深入的探討更高級複雜的規則化電腦AI和$max^n$演算法的電腦AI之外，也可以試著加入對手模組，藉由強化學習做出更好的判斷。Monte-Carlo的模擬決策搜尋也可以試著實做出來，尤其在不完全資訊的遊戲裡它有它獨特的優點。

# 參考文獻


[1] B. Bouzy and B. Helmstetter. **Monte-Carlo Go Developments**. Advances in Computer Games, Many Games, Many Challenges, 10th International Conference ACG2003 , Graz, Austria, 2003.

[2] David Carmel and Shaul Markovitch. **Learning and Using Opponent Models in Adversary Search**. 1996.

[3] Guillaume M. J. B. Chaslot, Mark H. M. Winands, and H. Jaap van den Herik. **Parallel Monte-Carlo Tree Search**. Computers and Games: 6th International Conference, CG 2008.

[4] Matthew L. Ginsberg. **GIB: Imperfect Information in a Computationally Challenging Game**. Journal of Artificial Intelligence Research, 14:303–358, 2001.

[5] D. E. Knuth and R. W. Moore. **An Analysis of Alpha-Beta Pruning**. Artificial Intelligence, 6:293–326, 1975.

[6] Richard E. Korf. **Multi-Player Alpha-Beta Pruning**. Artificial Intelligence, 49:99–111, 1991.

[7] Valavan Manohararajah. **Parallel Alpha-Beta Search on Shared Memory Multiprocessors**. Master's thesis, Graduate Department of Electrical and Computer Engineering, University of Toronto, Canada, 2001.

[8] C. E. Shannon. **Programming a Computer for Playing Chess**. Philosophical Magazine, 41(314):256–275, 1950.

[9] Nathan Sturtevant. **Current Challenges in Multi-Player Game Search**. Computers and Games: 4th International Conference, CG 2004.

[10] Matthew R. Wahab. **A Reinforcement Learning Agent for 1-Card Poker**. 2004.