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

結合單迫著與雙迫著搜尋之六子棋程式之研發

研究生：賴昱臣撰

中華民國九十九年六月

摘要

六子棋(Connnect6)，又名連六棋，為國立交通大學資訊工程系吳毅成教授於第十一屆國際電腦賽局研討會(11th Advances in Computer Games Conference，ACG11)中提出和發展。六子棋的提出是有鑑於五子棋(Go-Moku)和連珠棋(Renju)在幾經改善後仍無法完全消除先手方的優勢而提出的另一種較公平的棋類。

2007年，由劉思源及顏士淨博士設計的六子棋程式「X6」，主要使用雙迫著搜尋，在ICGA(International Computer Cames Association)六子棋競賽中取得金牌，顯示出迫著搜尋對棋力影響的強度；2007年及2009年，由Theo van der Storm先生所設計的六子棋程式「MeinStein」，使用淺層的alpha-beta search取得兩面銀牌，可見其審局函數之準確性。

不同於雙迫著搜尋的單迫著搜尋，在實作上難以兼顧成功率、準確性與誤判風險。本論文利用「MeinStein」開放原始碼中之其審局函數來提升單迫著搜尋在成功率、準確性、誤判風險之間的平衡。由此我們所研發出來的程式已能在實戰時較 X6 與 MeinStein 更強。

ABSTRACT

Connect6, also called Connective6, was proposed and developed by Professor I-Chen Wu in the 11th Advances in Computer Games Conference. Connect6 is similar to Go-Moku and Renju, and the reason why Connect6 is proposed is to eliminate the dominant position of the first player.

In 2007, a Connect6 program “X6” got the gold metal at a computer Connect6 competition which was conducted by ICGA(International Computer Games Association.) X6 used a technical search named “DTS(double threats search algorithm),” which demonstrates the strength in the competitions; Another Connect6 program “MeinStein,” was developed by Mr. Theo van der Storm, by using alpha-beta search. In 2007 and 2009, MeinStein won two silver metals respectively, so that it showed the accuracy of evaluate function.

STS(Single threat search algorithm) is different from DTS. The main challenge is that it’s hard to balance hit-rate, reliability and risk. This thesis will try to balance the three properties by combining the evaluation function of MeinStein with STS. Experimental results show that our program can beat both X6 and MeinStein.

誌謝

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

另外還要感謝實驗室的其他成員——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、黃信翰學長，同學詹傑淳、陳俊佑、謝政孝、李明臻、李啟峰，以及學弟妹勞永祥、陳志宏、蔡宗賢、唐心皓。

特別感謝指導教授林博士，及士傑學長、立德學長、雲青學長、永祥學弟，在平時提供實作技術上的建議與方向。

最後感謝我的父母栽培我，並適時地給予鼓勵與支持，讓我能順利完成學業，僅將此論文獻給我最敬愛的父母。

目錄

附表目錄

附圖目錄

第1章　緒論

1.1 研究背景

六子棋(Connnect6)，又名連六棋[6]，為國立交通大學資訊工程系吳毅成教授於第十一屆國際電腦賽局研討會(11th Advances in Computer Games Conference, ACG11)中提出和發展。六子棋的玩法和五子棋的玩法相似，除了目標是連六子而非五子，先手的一方(黑方)在第一手時下一子，之後雙方輪流各下兩子。六子棋的提出是有鑑於五子棋(Go-Moku)和連珠棋(Renju)在幾經改善後仍無法完全消除黑方(先手方)的優勢而提出的另一種較公平的棋類。

1.2 研究動機與目的

自 2006 年六子棋被列入國際電腦奧林匹亞(Computer Olympiad 2006)的棋類競賽項目中後，在 ICGA(International Computer Cames Association)的六子棋項目[1]，陸續出現不少頂級的電腦六子棋程式(表 1.1)，成為人工智慧領域中，一項新的研究議題。

表 1.1　ICGA Connect6 歷屆競賽排名

	金牌	銀牌
2006 年	NCTU6	X6
2007 年	X6	MeinStein
2008 年	NCTU6-Lite	Bitstronger
2009 年	Bit	MeinStein

2007 年，由東華大學劉思源及顏士淨博士所開發的六子棋程式「X6」使用雙迫著搜尋法取得金牌[2]。2008 年，由 Theo van der Storm 先生[8]所開發的六子棋程式——「MeinStein」開放了原始碼[5]，讓我們得以一窺其技術。「MeinStein」在 2007 年使用 alpha-beta 搜尋法對上 X6 時，恰巧連續地以單迫著及雙迫著交錯進攻取得勝利，意外地顯示出單迫著搜尋的重要性(如圖 1.1)。因此本論文構思利用其審局函數來實現單迫著搜尋的可行性。

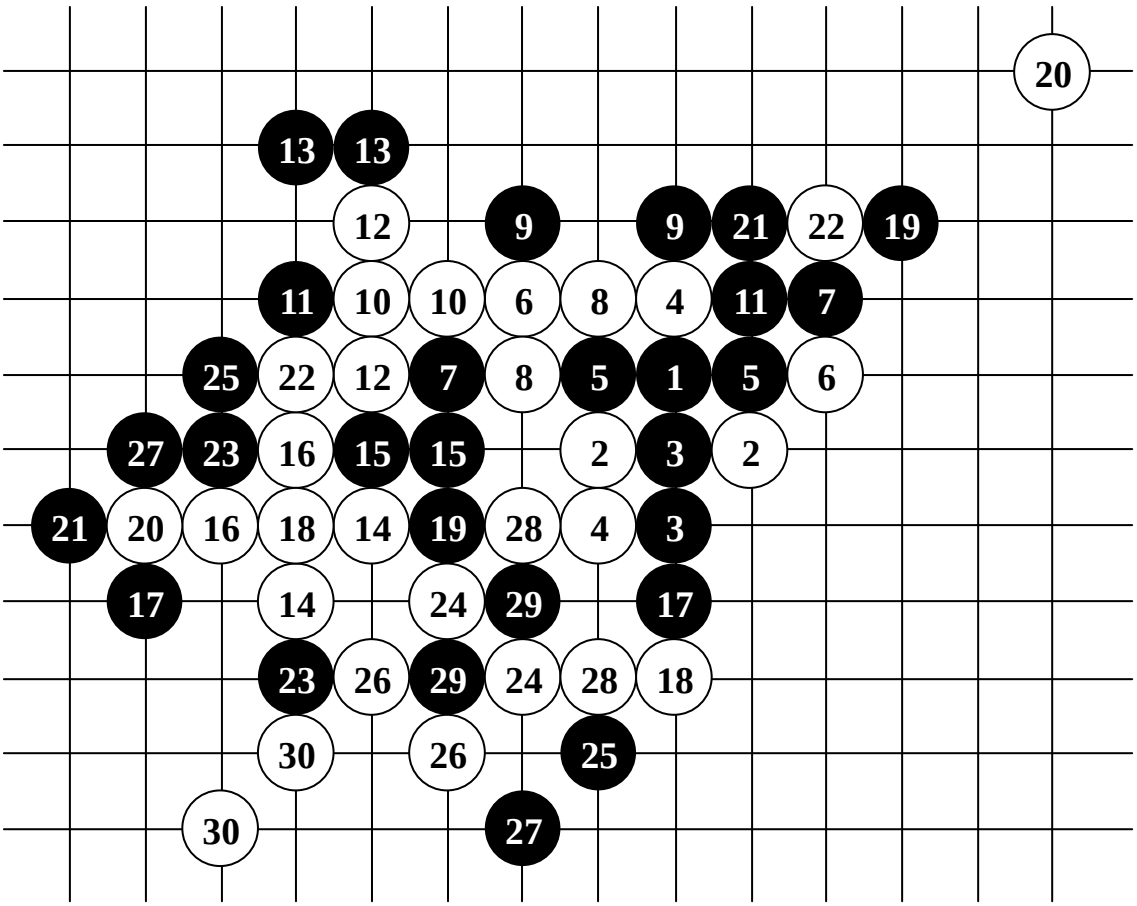


圖 1.1 MeinStein(白)(勝) vs. X6(黑)

1.3 論文概述

本論文共分為五章。首先第一章說明研究背景、動機與目的，第二章介紹六子棋的相關研究及基礎理論，接下來第三章說明本論文六子棋程式所使用的技術，第四章是相關的測試與分析，最後第五章是結論與未來的展望。

第2章　相關研究與基礎理論

2.1 六子棋的緣由與歷史

與發展歷史超過一百年的五子棋比較，六子棋發展至今不過四、五年，由於五子棋黑方子數佔優的不公平概念下，在五子棋和連珠棋的公平性的研究，引申出對一般的K子棋 Connect(m, n, k, p, q)研究，即是在 m × n 的棋盤上，先手的一方先下 q 子，爾後雙方輪流下 p 子，先連成 k 子者得勝，其公平性的一些研究成果，亦即各參數對公平性的影響參見表 2.1。在五子棋裡黑方(先手)永遠會比白方(後手)多一子，在白方下完子時，其在子數上最多和黑方一樣多。因此在 K 子棋 Connect(m, n, k, p, q)裡，第一手下的 p 子和之後輪流下的 q 子之間的關係，影響著 K 子棋的公平性[3]。

表 2.1　k 子棋的公平性

q(≦p)	k＝4 p＝1	k＝5 p＝2	k＝6 p＝3	k＝7 p＝4	k＝8 p＝5	k＝9 p＝6
1	B	B	W	W	W	W
2		B	W	W	W	W
3			B	FB	FB	FW
4				B	B	FW
5					B	B
6						B

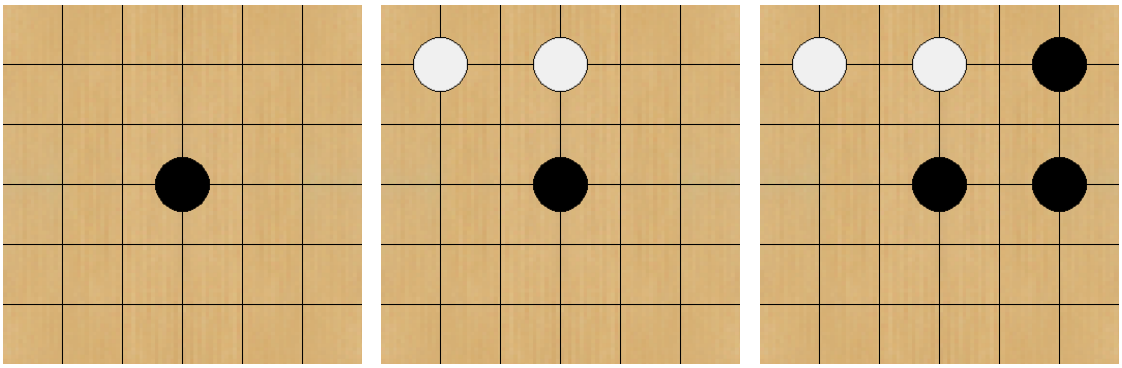
k 子棋中 p, q 和 k 值對公平性的影響。B 代表黑方(先手)必勝，W 代表白方(後手)必勝，FB 代表黑方有優勢，FW 代表白方有優勢。

2.2 遊戲規則

玩家：兩名玩家，各執黑子與白子。

玩法:除第一次黑方下一子外,之後黑白雙方輪流下兩子(圖 2.1)。在同一直線上先形成連續六子為己方棋子獲勝。

棋盤：在電腦六子棋競賽中，使用與圍棋相同的 19 × 19 棋盤；專業棋士的對局，可使用多個圍棋棋盤拼成 59 × 59 的棋盤。

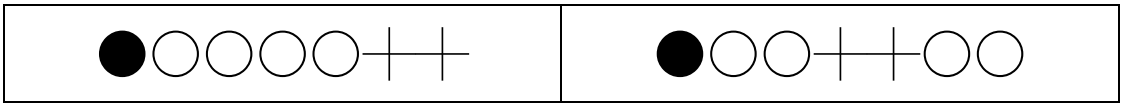


第一回合　　第二回合　　第三回合

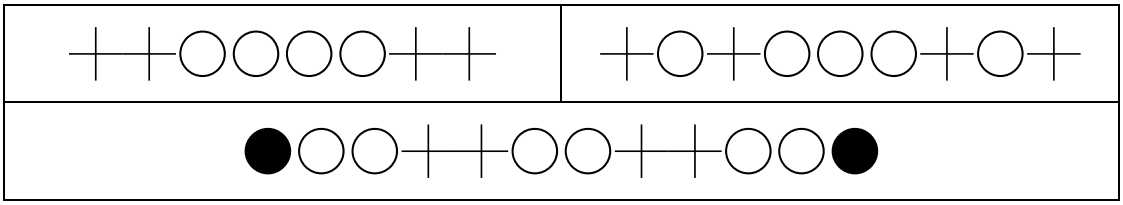
圖 2.1　棋局範例

2.3 遊戲策略

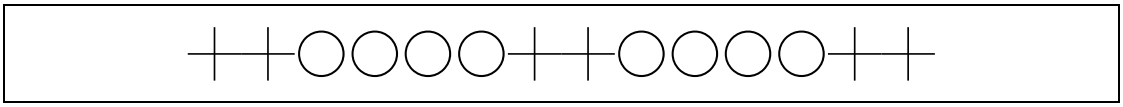
首先，迫著的定義如下：以六子棋為例，當對方形成連四時，我方必定要下一至二子防守，否則對方將會獲勝。而根據防守方必須抵擋下子個數的不同，可分為單迫著、雙迫著與三迫著。



單迫著



雙迫著



三迫著

圖 2.2　三種迫著棋型範例

如何連續地形成迫著(Threats)，使得對方忙於應對，是在取勝的過程中一個重要的原則。

第3章　電腦六子棋程式

3.1 程式架構

本論文的六子棋程式，其決定走步的流程如圖 3.1，首先作迫著搜尋(Threat-based search)[4]，成功便直接下子；失敗就利用審局函數得出若干走步，再分別檢查對方接下來迫著搜尋是否會成功，選擇一步對方會失敗的作為我方走步；若對方全數搜尋都成功，那麼就選擇我方審局分數最高分的一步。

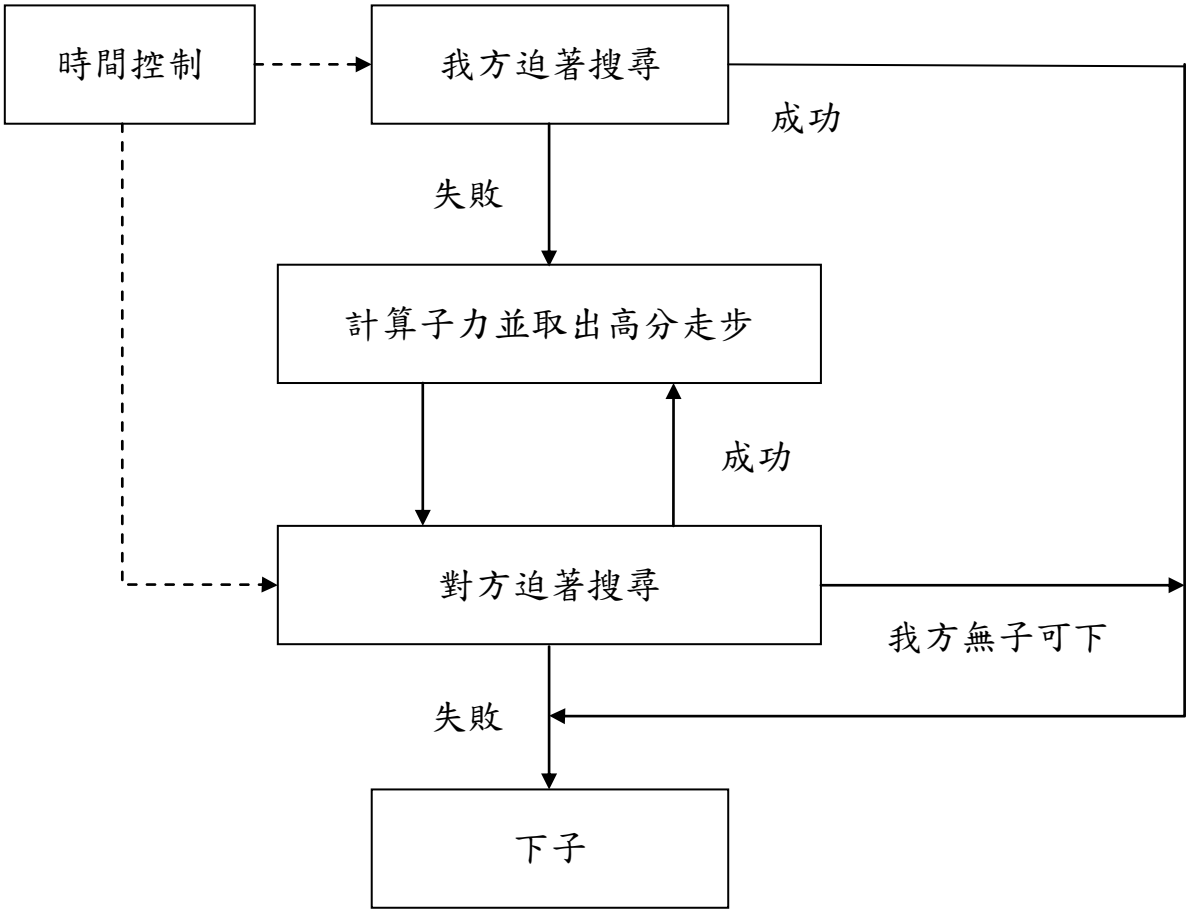


圖 3.1　六子棋程式流程圖

盤面資訊

在「MeinStein」程式中，我們僅見到它使用了 alpha-beta 遊戲樹的搜尋[12]，此種搜尋法對盤面資訊的依賴性相當高，而在比賽中其排名卻也不差，可見其盤面資訊之準確性。因此本論文程式使用「MeinStein」的棋型分數來作為盤面資訊之參考。「MeinStein」共有 10 種棋型(表 3.1)(圖 3.2)。

表 3.1　MeinStein 的棋型分數

棋型	棋型分	迫著分
死二	100	0
死三	300	0
活二	400	0
活二死三*	600	0
活三	1000	0
死四	2000	1
死五	2000	1
活四	4000	2
活四死五*	4500	2
活五	4500	2

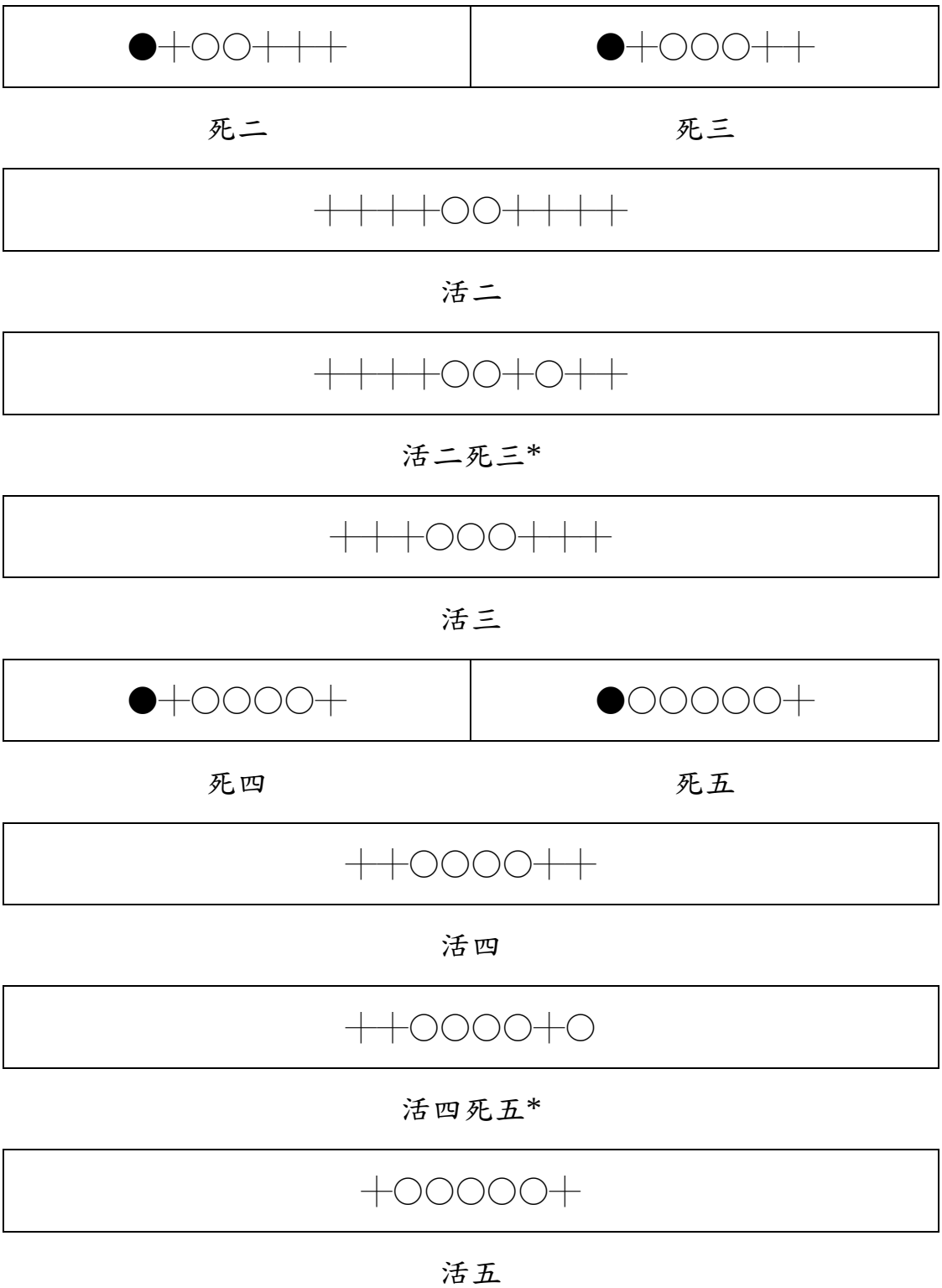


圖 3.2　MeinStein 棋型範例

在這些棋型中,可以見到其中較直覺的八種棋型——死二、死三、死四、死五、活二、活三、活四、活五。有趣的是其餘兩種——活二死三與活四死五。

由於六子棋棋型較五子棋為長，棋子排列組合的種類多，除了八種死型與活型，X6 還用了八種「眠型」與「斷型」。「MeinStein」也知曉此一狀況，因此使用了「半活型」來彌補死型與活型的不足。

3.2 資料結構

3.3.1 Slice

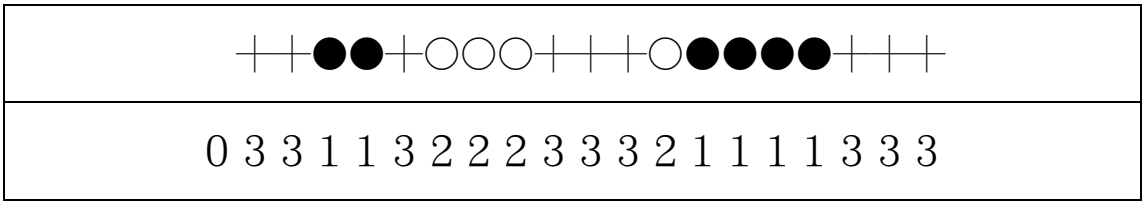
在六子棋程式「MeinStein」中，將棋盤上所有單一的一個直線稱為一個「Slice」。在一個 19 × 19 的棋盤上，共有 112 個 Slice。每一個 Slice，使用兩個整數，分別紀錄黑子與白子所在之位置，並各自視為 2 進位的數字，本論文將此數字稱作 2 進位棋型值，見圖 3.3。

	┼┼●●┼○○○┼┼┼○●●●●┼┼┼
黑	0 0 1 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0
白	0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0

圖 3.3　使用 2 進位棋型值之 Slice

3.3.2 Slice 改良

沿用 Slice 的概念，本論文將兩個 2 進位棋型值所組成的 Slice，整合為一個整數，為 4 進位棋型值 (圖 3.4)。



0(不可下子)，1(黑子)，2(白子)，3(無子)

圖 3.4 使用 4 進位棋型值之改良 Slice

由於每一個位置都會改變盤面上的 4 個 Slice，因此每一回合，每下一子，便對該子所影響到的 4 個 Slice 作更新。

3.3 迫著搜尋

3.4.1 迫著搜尋簡介

迫著搜尋意指我方(攻擊方)不斷形成迫著，逼迫對方(防守方)防守，我方是否能取得勝利。此種搜尋法在搜尋成功時，表示我方存在有一條必勝的路徑，此時不必在意對方的棋型，接連地按照順序形成迫著便可取勝；搜尋失敗時，代表我方不能以連續性的迫著取勝，該好好注意盤面局勢，應該壓制對方亦或發展我方的棋型。

依照形成最少迫著數的不同，分為兩種搜尋：

- 單迫著搜尋：我方形成迫著時，迫著數必定大於等於 1。
- 雙迫著搜尋：我方形成迫著時，迫著數必定大於等於 2。

迫著搜尋的程式流程如圖 3.5。

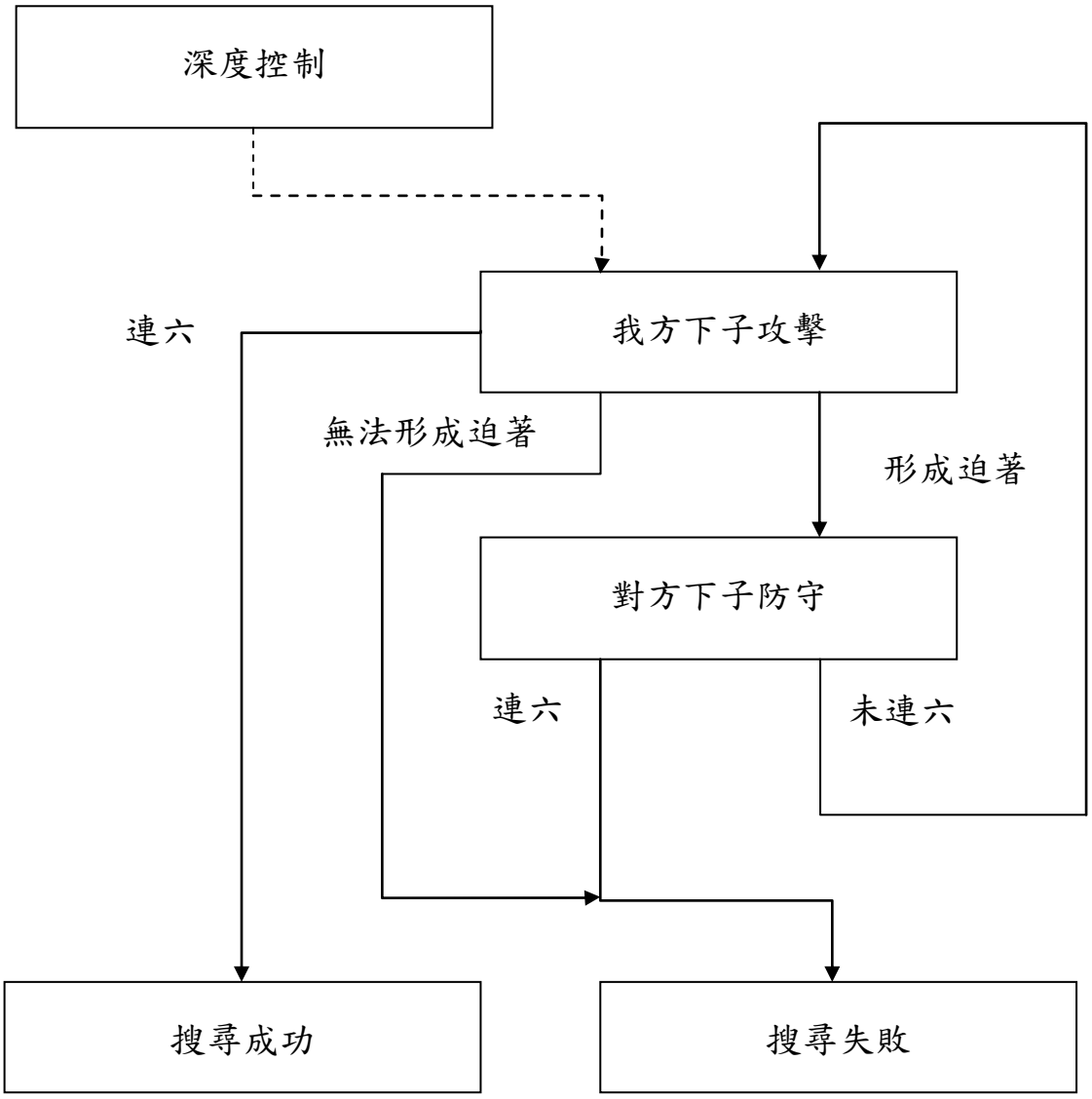


圖 3.5　迫著搜尋流程

攻擊方形成迫著，防守方如何防守，決定了迫著搜尋的準確性與成功率。單迫著搜尋與雙迫著搜尋時，防守方的策略將分開來討論。

3.4.2 雙迫著搜尋

攻擊方形成雙迫著之後，防守方的選擇不多，最多可能有三到四種防守法，如圖 3.6，攻擊方為白方，防守方為黑方，所下的黑子為以「・」表示

┼・○○○○・┼
・┼○○○○・┼
┼・○○○○┼・

(a) 當白方有活四時，黑方可能的防守組合有三種。

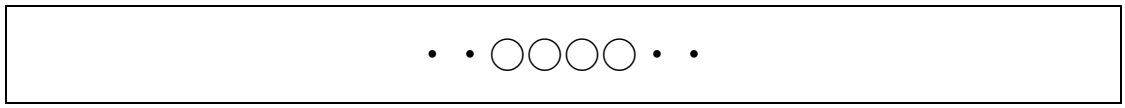
●○○○○┼・	●○○○○┼・
●○○○○┼・	●○○○○・┼
●○○○○・┼	●○○○○┼・
●○○○○・┼	●○○○○・┼

(b) 當白方有兩個死四時，黑方可能的防守組合有四種。

圖 3.6 雙迫著的防守組合(以・表示)

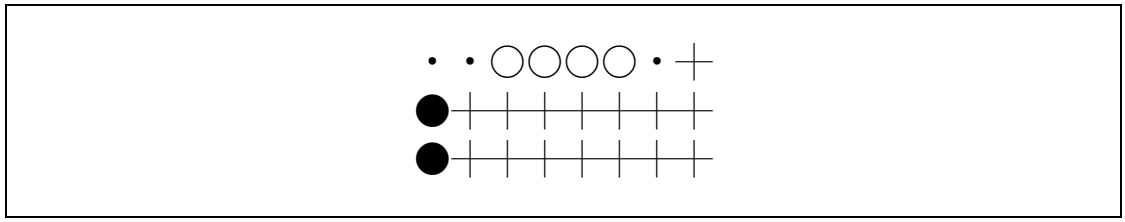
若要確保雙迫著搜尋的準確性，在作遊戲樹的展開時，必須要對防守方所有可能防守的方式都作展開，這使得遊戲樹的分支度增加，在有限時間的搜尋下，有廣度太寬、深度不足的問題。

為解決這個問題，X6 使用了保守型防禦與改良型防禦：無視一次走步恰巧下兩子的規則，將可能的組合作聯集(圖 3.7)。如此遊戲樹在防守方那一層只有一個分支 (圖 3.8)。



(a) 當白方有活四時，黑方的保守型防禦組合，一次下四子。

(b) 當白方有兩個死四時，黑方可能的保守型防禦組合，一步下四子。



(c) 當白方有活四時，黑方可能的改良型防禦組合，一步下三子。

圖 3.7　保守型防禦(以・表示)

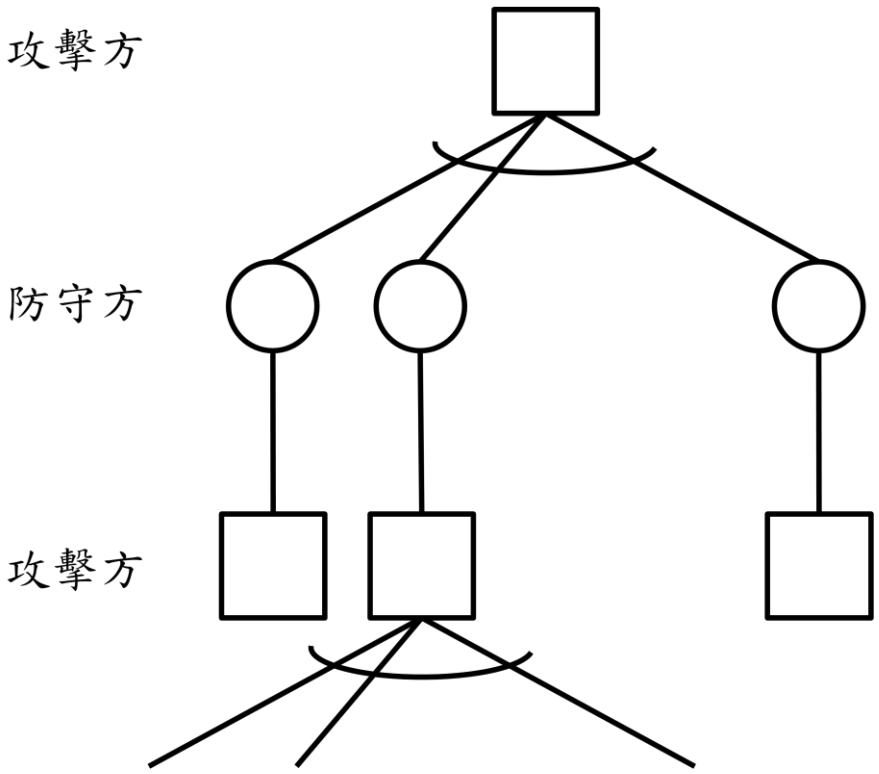


圖 3.8　雙迫著搜尋的遊戲樹

使用保守型防禦與改良型防禦可確保雙迫著搜尋的準確性，但犧牲了成功率。因為防守方最多下四子，防守方的子數增加得太快，可能將實際上會成功的路徑誤判為失敗。

為降低誤判率，本論文程式將防守方可下的最多的子數限制為三子，藉由審局函數，將分數最低的一子剔除。如此搜尋的成功率會較保守型防禦與改良型防禦更高；而在誤判率方面，我們的實驗中並未出現誤判的情況。

3.4.3 單迫著搜尋

相較於雙迫著搜尋，防守方可能只有一子必須防守，防守方另一子的走步將更難預測，一旦預測錯誤就會造成誤判。誤判的結果可能反而讓防守方存在一條必勝的路徑。為降低誤判的機率與風險，可能的方法如下：

I. 提升審局函數的準確率

- 必須透過大量的棋局測試，持續並有效地對棋型的分數作修改與維護，困難度較高，本論文並未使用。

II. 增加防守方的分支度

- 對防守方較有可能的走步皆作展開，但這會增加龐大的計算時間，因此本論文也不考慮。

III. 讓防守方的走步有明顯的傾向

- 讓防守方傾向於防守性強，以下稱為防守型防守方。
- 讓防守方傾向於攻擊性強，以下稱為攻擊型防守方。
- 此方法將在 4.2 詳細的探討。

3.4 子力的計算

要決定如何下子時，必須知道盤面上特定位置的優劣，在該位置下子的優劣稱為該位置的子力。而一個子會影響到 4 個 Slice，便用該子在 4 個 Slice 中的子力作組合。Slice 中某特定位置的子力，是針對該位置下了黑子或白子之後，對鄰近位置所造成的棋型變化所得到。因此審局函數主要步驟有三：

- 取出棋型。
- 得到(多個)棋型變化的分數。
- 將棋型變化的分數作權重的加總得到子力。

很明顯地，與欲評分的位置距離太遠的棋子，對子力並無太大影響，可以忽略。因此本論文程式是以該位置為中心，往左右各拓展 9 格，左右 9 格與本身 1 格共 19 格稱為該位置的視窗(window)，其中可能包含了白子、黑子、空格跟不可下子處。

但此視窗中的使用的是 4 進位棋型值，棋型值最大為 4^{19}，過於龐大，在硬體的限制之下，無法藉由 Hash Table 來直接得到子力。

我們退而求其次，使用四個 hash table：Left、Right、LLength、RLength 作組合來得到視窗內黑子與白子的 2 進位棋型值。今假設一棋型，如圖 3.9 之上方所示，欲評分的位置左邊九格的 4 進位棋型值為 $A=(322233321)_4=(240633)_{10}$，右邊九格的 4 進位棋型值為 $B=(311130000)_4=(218880)_{10}$，可得到棋型值與長度為：

$$\begin{cases} \text{棋型值} = A \times 4^{10} + B = (3222333210311130000)_4 \\ \text{棋型長度} = 19 \end{cases}$$

將 A 經過 Left 查表得其二進位棋型值 $A'=(1)_2=(1)_{10}$，經過 LLength 查表得其棋型長度 $L_{A'}$；將 B 經過 Right 查表得其二進位棋型值 $B'=(01110)_2=(14)_{10}$，經過 RLength 查表得其棋型長度 $L_{B'}$。可得到：

$$\begin{cases} \text{棋型值} = A' \times 2^{18-L_A-L_B} + B' = (1001110)_2 = (78)_{10} \\ \text{棋型長度} = L_{A'} + L_{B'} + 1 = 7 \end{cases}$$

然而為了效率，在實作上使用 shift 與 or 來代替次方與乘法運算，圖 3.10 為子力計算之虛擬碼。

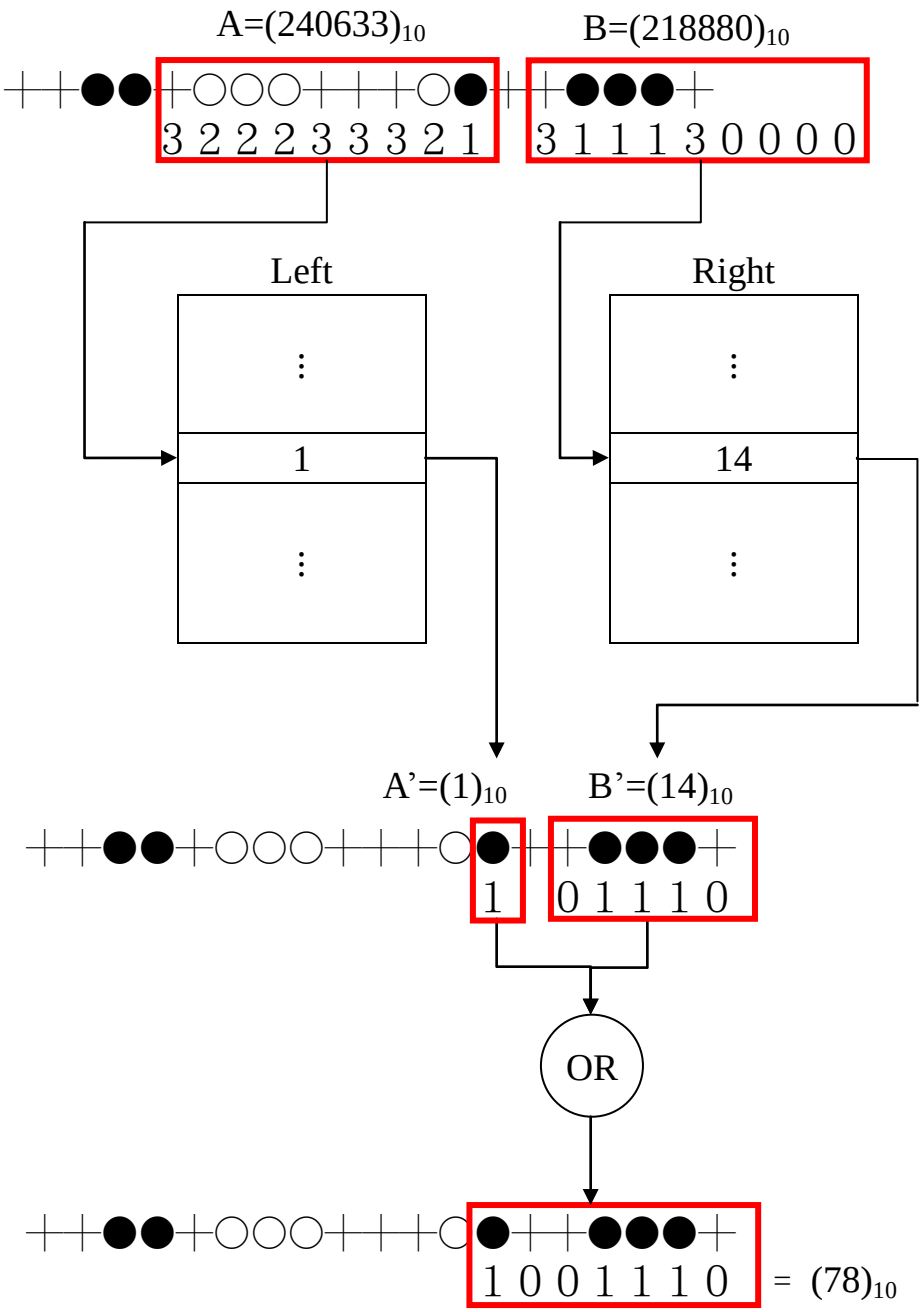


圖 3.9　由 4 進位棋型值得到 2 進位棋型值

```
     Input:
        Slice //棋型值
        Index //欲評分之位置，0~18
        //由 4 進位棋型值得到 2 進位棋型值
        Left[],Right[]
        //由 4 進位棋型值得到 2 進位棋型值長度
        LLength[], RLength[]
     Output:
        Key //2 進位棋型值
        Length //2 進位棋型值長度

01   Index = Index  * 2
02   A = (Slice >> Index) & 0x3FFFF
03   if( Index >= 18 )
04      B = (Slice >> (Index - 18)) & 0x3FFFF
05   else
06      B = (Slice << (18 - Index)) & 0x3FFFF
07   Key = (Left[A] << 10) | Right[B]
08   Key = Key >> (9 - RLength[B])
09   Length = LLength[A] + RLength[B] + 1
10   return Key, Length
```

圖 3.10　由 4 進位棋型值得到 2 進位棋型值虛擬碼

利用 2 進位棋型值，便可快速的得到棋型分數，進而計算棋型分數的變化。在「MeinStein」程式中，任一空格位置之子力由以下 6 個分數計算而得：

- 此空格下了我方子後，我方棋型的變化分(ΔP_1)。
- 此空格下了我方子後，對方棋型的變化分(ΔP_2)。
- 此空格下了對方子後，對方棋型的變化分(ΔP_3)。
- 此空格下了對方子後，我方棋型的變化分(ΔP_4)。
- 此空格下了我方子後，我方迫著的變化分(ΔT_1)。
- 此空格下了我方子後，對方迫著的變化分(ΔT_2)。

$$\begin{cases} 子力 = (\Delta P_1 - \Delta P_2) + \frac{2}{3}(\Delta P_3 - \Delta P_4) + w_1 \Delta T_1 + w_2 \Delta T_2 \\ w_1 = 10000 \\ w_2 = -100000 \end{cases}$$

假設今一棋型如圖 3.11，P 為黑方欲評分之特定位置。

圖 3.11　MeinStein 之審局函數計算子力範例

其中各項參數為：

$$\begin{cases} \Delta P_1 = 4500 - 2000 = 2500 \\ \Delta P_2 = 0 - 0 = 0 \\ \Delta P_3 = 0 - 0 = 0 \\ \Delta P_4 = (0 + 300) - 2000 = -1600 \\ \Delta T_1 = 1 - 1 = 0 \\ \Delta T_2 = 0 - 0 = 0 \end{cases}$$

最後所得該位置 P 之子力為：

$$\begin{aligned} \text{子力} &= (\Delta P_1 - \Delta P_2) + \frac{2}{3}(\Delta P_3 - \Delta P_4) + w_1 \Delta T_1 + w_2 \Delta T_2 \\ &= (2500 - 0) + \frac{2}{3}(0 - (-1600)) + w_1 \times 0 + w_2 \times 0 \\ &= 2500 + 1066 + 0 + 0 = 3566 \end{aligned}$$

3.5 審局函數

為了因應迫著搜尋的需要，我們的程式的子力求法便由此為基礎作變化。將有四個審局函數：

- 無迫著時使用的審局函數。由兩個棋型變化分所組成。
  - 下了我方子後，我方棋型的變化分(ΔP_1)。
  - 下了對方子後，對方棋型的變化分(ΔP_3)。

$$子力 = \Delta P_1 + \Delta P_3$$

舉例如圖 3.12，A 為黑方欲求子力之位置。

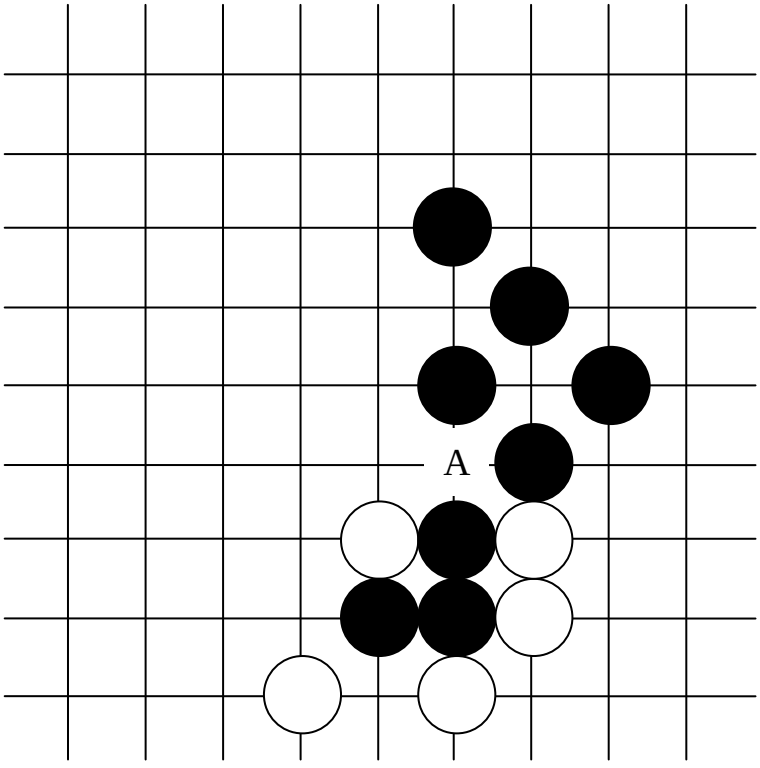


圖 3.12　無迫著審局函數範例

各項參數為：

$$\begin{cases} \Delta P_1 = (4500 - 2000) + (400 - 0) = 2900 \\ \Delta P_3 = (400 - 0) + (400 - 0) = 800 \end{cases}$$

A 下黑子後之子力為：

$$子力 = 2900 + 800 = 3700$$

- 迫著搜尋時，攻擊方之審局函數。由兩個棋型變化分所組成。
  - 下了我方子後，我方棋型的變化分(ΔP_1)。
  - 下了我方子後，我方迫著的變化分(ΔT_1)。

$$子力 = \Delta P_1 + 10000 \Delta T_1$$

舉例如圖 3.13，B 為黑方欲求子力之位置。

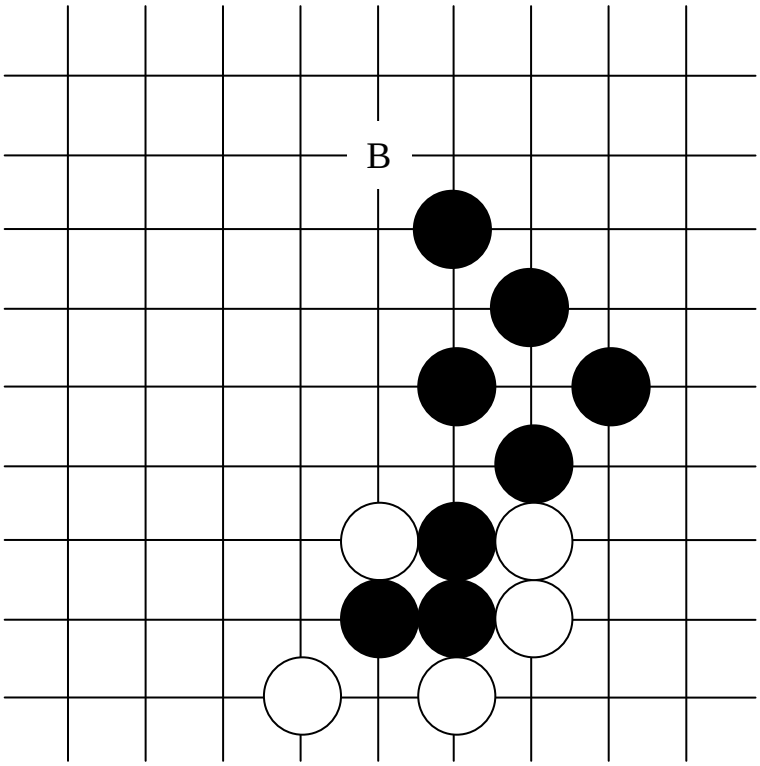


圖 3.13　有迫著之攻擊方審局函數範例

各項參數為：

$$\begin{cases} \Delta P_1 = (4000 - 1500) = 2500 \\ \Delta T_1 = (1 - 0) = 1 \end{cases}$$

B 下黑子後之子力為：

$$子力 = 2500 + 10000 \times 1 = 12500$$

- 迫著搜尋時，防守方之審局函數。由三個棋型變化分所組成。
  - 下了我方子後，我方迫著的變化分(ΔT_1)。
  - 下了對方子後，對方棋型的變化分(ΔP_3)。
  - 下了對方子後，對方迫著的變化分(ΔT_3)。

$$子力 = 2000\Delta T_1 + \Delta P_3 - 10000\Delta T_3$$

舉例如圖 3.14，C 為白方欲求子力之位置。

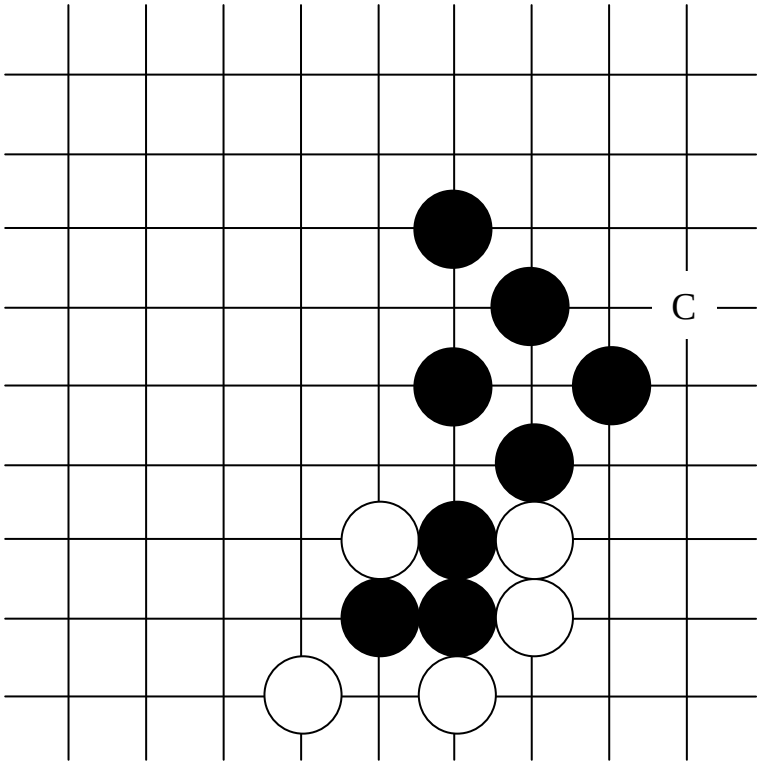


圖 3.14　有迫著之防守方審局函數範例

各項參數為：

$$\begin{cases} \Delta T_1 = (0-0) = 0 \\ \Delta P_3 = (2000-2000) = 0 \\ \Delta T_3 = 0-1 = -1 \end{cases}$$

C 下白子後之子力為：

$$子力 = 2000 \times 0 + 0 - 10000 \times 1 = 10000$$

- 迫著搜尋時，防守方可無視規則而多下的第三子之審局函數。由兩個棋型變化分所組成。
  - 下了我方子後，我方棋型的變化分(ΔP_1)。
  - 下了對方子後，對方棋型的變化分(ΔP_3)。

$$子力 = \Delta P_1 + \Delta P_3$$

舉例如圖 3.15，D 為白方欲求子力之位置。

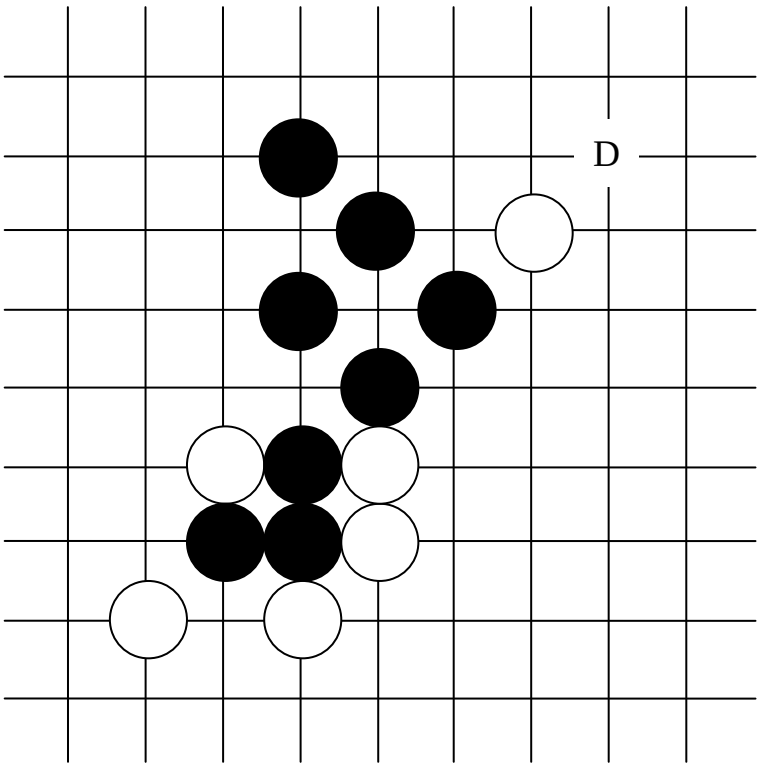


圖 3.15　有迫著時防守方第三子之審局函數範例

各項參數為：

$$\begin{cases} \Delta P_1 = (300 - 0) = 300 \\ \Delta P_3 = (0 - 0) = 0 \end{cases}$$

D 下白子後之子力為：

$$子力 = 300 + 0 = 300$$

第4章　測試與分析

4.1 測試平台

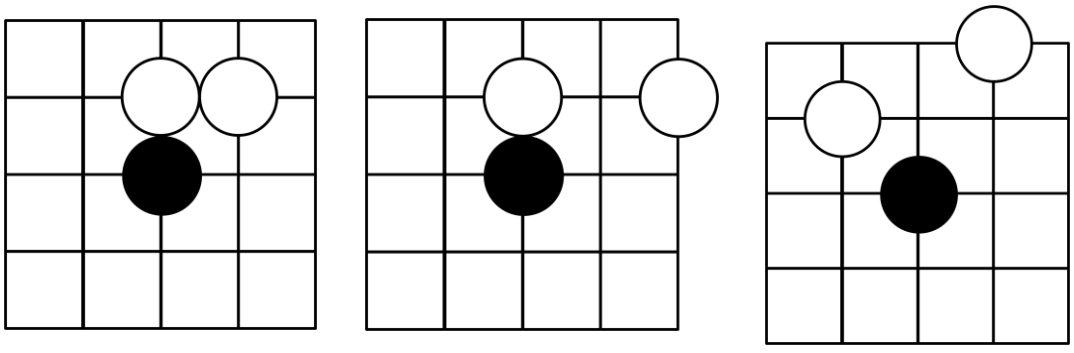
本論文程式分為兩部分，GUI部分使用UltraEdit撰寫JAVA程式，Compiler為JAVA 1.6u18；審局函數及迫著搜尋部分使用Dev C++為開發工具。兩者之間使用 JNI(JAVA Native Interface)形成DLL(Dynamic Linking Libary)。個人電腦為Intel P4 3.2GHz、2G RAM、Windows XP Professional。

4.2 測試盤面

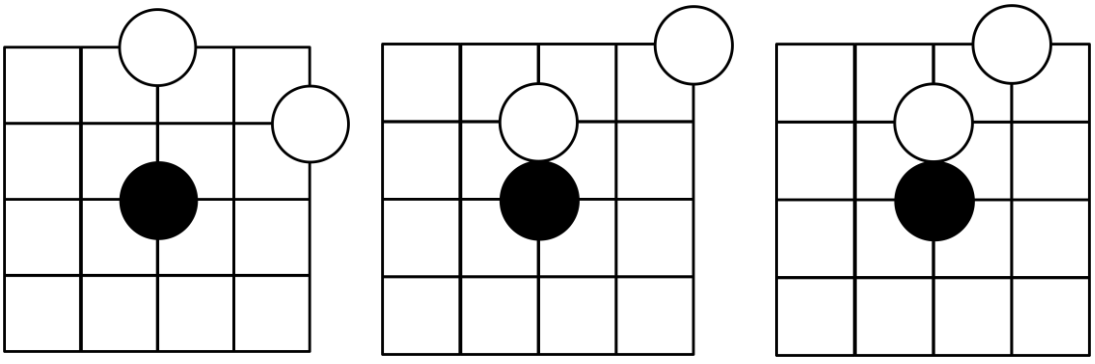
為測試使用防守型與攻擊型防守方的單迫著搜尋的效果，本論文將以以下幾點作討論：

- 搜尋的成功率。
- 搜尋成功時的可信度，即準確性。
- 誤判為成功時的風險。
  - 當一個盤面實際上並不存在於一條必勝的路徑時，迫著搜尋還是有可能搜尋成功。因為防守方的走步預測錯誤，這時貿然地攻擊，可能會得到反效果。

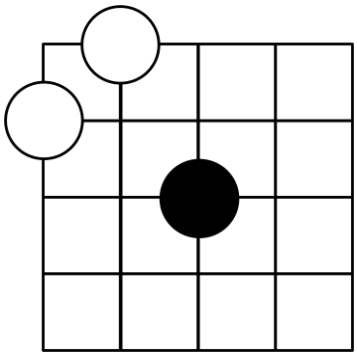
首先，本論文程式使用兩種版本(3.4.3 所述之防禦型及攻擊型)分別以圖 4.1 中的 7 個初始盤面與 X6(level 9)對戰。本論文程式執黑子，X6 執白子，結果如表 4.1。



(A)　　(B)　　(C)



(D)　　(E)　　(F)



(G)

圖 4.1　測試盤面

表 4.1　本論文程式(黑) vs. X6(白)

	A	B	C	D	E	F	G	勝率	敗率	和率
防守型	敗	勝	敗	敗	敗	敗	敗	14%	86%	0%
攻擊型	和	勝	勝	和	和	勝	和	43%	0%	57%

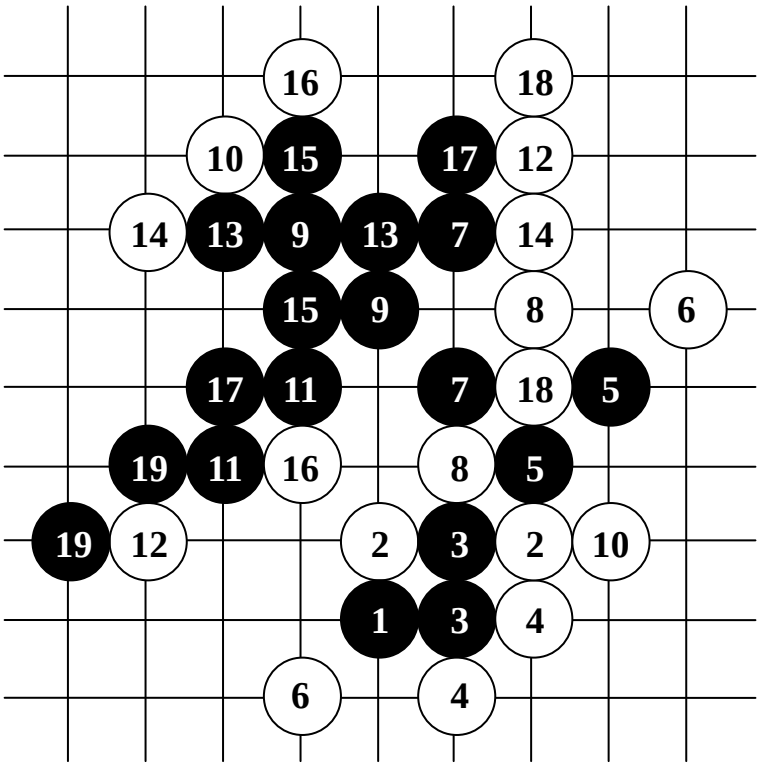


圖 4.2　盤面 B 本論文程式(黑)(勝) vs. X6(白)

4.3 策略分析

經過圖 4.1 中所有盤面的測試，分析如表 4.2、表 4.3。

表 4.2　單迫著搜尋防守方策略之分析

策略	評估		原因
防守型	成功率	高	迫著搜尋的過程中，並未考慮到防守方反擊的狀況。
	準確率	低	當搜尋成功時，只能確保在防守方嚴密的防守下，攻擊方之攻勢夠強；同成功率高的原因，並未考慮到防守方反擊的狀況。
	風險	高	同成功率高的原因，並未考慮到防守方反擊的狀況，而實際狀況可能只要受到一次防守方的反擊攻勢就會瞬間瓦解。而且防守方反擊時有極高的機率是已存在一條必勝的路徑。

表 4.3　單迫著搜尋防守方策略之分析

策略	評估		原因
攻擊型	成功率	中	考慮到防守方會反擊，而反擊的機會其實相當多，因此搜尋的成功率並不高。
	準確率	高	相較於防守型，此策略並無考慮防守方全力防守的狀況。因防守方反擊的機率高，搜尋過程中多數分支皆為失敗；搜尋成功的路徑中，即使防守方反擊，也以更強的攻擊壓制；換句話說，此策略較防守型更能確保攻擊方的攻勢夠強。
	風險	低	當實際上不存在必勝的路徑卻搜尋成功而貿然攻擊時，雖不會勝利，但在搜尋時已使得防守方的攻擊性強，因此不會造成防守方一反擊就一蹶不振的後果。

第5章　結論與未來展望

5.1 結論

本論文依照前面各章所提出之資料結構的改良、迫著搜尋與審局函數，所研發之電腦六子棋程式，在與 X6 的對局中有不錯的表現。尤其在迫著搜尋的成功率上，效果比起 X6 所使用之雙迫著搜尋更為突出。準確性方面，雖然單迫著搜尋的準確性不如雙迫著搜尋之精準，但藉由降低誤判風險，整體搜尋效果還是較純雙迫著搜尋要好。另外在純雙迫著搜尋方面，由於本論文程式將防守方所下的子數控制在二至三子，搜尋成功的路徑也較使用保守型防禦與改良型防禦的 X6 要短。

5.2　未來展望

本論文所深入研究之主題，主要偏重在單迫著搜尋的成功率、準確性、誤判風險之間的平衡。而電腦六子棋還有許多問題可以研究，例如使用改良 Monte-Carlo 演算法的 UCT 演算法[9][10][11]的遊戲樹搜尋；或是成功率更高、精確性更高、誤判風險更低的迫著搜尋演算法。本論文研究之實戰棋譜，以及實驗數據，對於電腦六子棋的後續研究，都可作為對照與參考的資料。

參考著作

[1] ICGA(International Computer Cames Association)六子棋競賽，http://www.grappa.univ-lille3.fr/icga/game.php?id=18

[2] 劉思源，電腦六子程式 X6 之設計與實作，國立東華大學碩士論文，2006 年。

[3] I-Chen Wu and Dei-Yen Huang, A New Family of k-in-a-row Games, the 11th Advances In Computer Games Conference (ACG'11), Taipei, Taiwan, September 2005.

[4] I-Chen Wu and Ping-Hung Lin, New Threat-based Proof Search for k-in-a-row Games, IEEE,

[5] MeinStein，http://www.csvn.nl。

[6] 六子棋首頁，http://www.connect6.org/。

[7] 劉雲青，六子棋中一個結合迫著搜尋的防禦性策略。國立臺灣師範大學碩士論文，2009 年。

[8] Theo van der Storm 先生簡介，http://chessprogramming.wikispaces.com/Theo+van+der+Storm。

[9] David Silver, Gerald Tesauro,“Monte-Carlo Simulation Balancing”, 2009.

[10]Remi Coulom, Computing Elo ratings of move patterns in the game of Go, Computer Games Workshop, 2007.

[11]方裕欽，UCT 算法的適用性及改進策略研究—以黑白棋為例，國立臺灣師範大學碩士論文，2008 年。

[12]Alpha-beta pruning http://en.wikipedia.org/wiki/Alpha-beta_pruning。

[13]吳身潤，人工智慧程式設計，維科圖書，2002 年 3 月。