

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

八層及九層三角殺棋的勝負問題之改進與研究

On the Study and Improvement of 8 Layer and 9

Layer Triangular Nim

研究生：陳俊佑 撰

中華民國九十九年六月

摘要

電腦棋類遊戲在人工智慧領域中是很重要的一環。三角殺棋的部份，於 1985 年由許舜欽教授研究出七層三角殺棋的結果後便一直沒有更高層數三角殺棋的相關文獻了。直至 2009 年才有白聖群以及林宏軒兩位研究生各自做了八層三角殺棋的破解研究。

在本論文中，我們使用 CPU 規格為 Intel Xeon E5520 2.27GHz(雙處理器)，記憶體總量為 36G Byte 的機器，證明了九層三角殺棋於取得最後一子為敗的規則下，是先手必勝的結果。另外我們也應用 Divide-and-Conquer 以及 Sprague-Grundy function 等方法，列出了九層三角殺棋於取得最後一子為勝的規則下，保證下了必敗的著手。

我們除了找出九層三角殺棋的結果，也對八層三角殺棋的解法做了分析與改良，提出可以大幅度節省破解所需空間及時間的辦法，更有效率的使用記憶體。雖然以目前的硬體設備只能應用在八層以下的三角殺棋，但是這個概念或許也可以應用在往後的更高層數三角殺棋求解上。

關鍵字：三角殺棋、人工智慧、回溯分析、倒推法

ABSTRACT

Computer chess game is a very important part in the field of artificial intelligence. There is no research on Triangular Nim in higher dimensions since Professor Shun-Chin Hsu solved the 7 Layer Triangular Nim in 1985. Then the 8 Layer Triangular Nim had been solved by two graduate students Bai and Lin independently until 2009.

In this thesis, a dedicated computer equipped with Intel Xeon E5520 2.27GHz(Dual Processor) CPU and 36G Bytes RAM is utilized to conduct our experiments. Thus, we get the result that in the 9 Layer Triangular Nim, the first player can win in misere play. Besides, we also list all the legal moves which can lead the first player lose the game in normal play, by using divide-and-conquer and Sprague-Grundy function.

In addition to finding the results of 9 Layer Triangular Nim, we also analyze and improve the program for solving the 8 Layer Triangular Nim. We can greatly save the time and space used. Although the current hardware can only be applied in solving the 8 Layer Triangular Nim, but this concept may be applied to solve Triangular Nim in higher dimensions in the future.

Keywords: Triangular Nim, Artificial Intelligence, Retrograde

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、賴昱臣、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，及聖群學長、雲青學長，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

摘要.....	I
ABSTRACT.....	II
誌謝.....	III
附圖目錄.....	VI
附表目錄.....	VIII
第一章 緒論	1
第一節 前言.....	1
第二節 研究動機.....	4
第三節 論文架構.....	4
第二章 相關文獻及基礎理論	5
第一節 相關研究成果.....	5
第二節 倒推法.....	7
第三節 產生可行著手.....	10
第三章 九層三角殺棋之破解	15
第一節 利用必敗盤面的非完全資訊嘗試破解.....	15
第二節 減少棋子數與重新編碼.....	19
第三節 Divide-and-Conquer.....	28
第四節 利用 Sprague-Grundy Function 解九層三角殺棋.....	31
第五節 淺層搜尋演算法.....	37
第四章 八層三角殺棋之加速與空間之節省	46
第一節 三角殺棋的等價性質與複雜度.....	46
第二節 重新編碼.....	46
第三節 於九層三角殺棋上之應用.....	50
第五章 結論及未來研究方向	53

第一節 結論.....	53
第二節 未來研究方向.....	54
附錄 A 九層三角殺棋已知必敗之著手列表.....	55
參考著作.....	70

附圖目錄

圖 1-1 六層三角殺棋	1
圖 1-2 非法著手	2
圖 1-3 所有棋子皆被取過的狀態	3
圖 2-1 分段式記憶體配置	6
圖 2-2 回溯分析法預先更新盤面的狀況	7
圖 2-3 倒推法歸納勝負的原理	8
圖 2-4 九層三角殺棋編碼圖	8
圖 2-5 倒推法演算法	9
圖 2-6 可行著手對應之盤面狀態示意圖	10
圖 2-7 邏輯運算的利用法及結果	11
圖 3-1 旋轉等價的盤面	16
圖 3-2 對稱等價的盤面	17
圖 3-3 非完全資訊破解流程	19
圖 3-4 預先選取的可行著手	21
圖 3-5 選取 9 子的著手	22
圖 3-6 九層三角殺棋預先取子後的重新編碼	23
圖 3-7 規則為取得最後一子為敗，九層三角殺棋第勝的第一著手	25
圖 3-8 預先取子與對應之必勝著手	27
圖 3-9 三層三角殺棋之必勝著手	28
圖 3-10 四層餐角殺棋之必勝著手	28
圖 3-11 五層三角殺棋之必勝著手	28
圖 3-12 六層三角殺棋之必勝著手	29
圖 3-13 七層三角殺棋之必勝著手	29
圖 3-14 八層三角殺棋之必勝著手	30

圖 3-15 Divide-and-Conquer 概念下的六層三角殺棋	30
圖 3-16 Sprague-Grundy Function	32
圖 3-17 二層三角殺棋各盤面狀態之 Grundy Number	33
圖 3-18 選定將盤面一分為二的四種可行著手	35
圖 3-19 Grundy Number 值相等的盤面狀態	36
圖 3-20 暴力展開遊戲樹之作法	37
圖 3-21 十二種盤面切割檢查法	39
圖 3-22 淺層搜尋演算法的虛擬碼	40
圖 3-23 N 層淺層搜尋之結構	41
圖 3-24 可行著手表之更新流程	42
圖 3-25 淺層搜尋找出之必敗著手	45
圖 4-1 八層三角殺棋的重新編碼	46
圖 4-2 八層三角殺棋重新編碼之區段圖	47
圖 4-3 旋轉盤面之虛擬碼	48
圖 4-4 可利用加速及空間節省概念找出勝負之著手	50
圖 4-5 重新編碼之九層三角殺棋	51
圖 4-6 找到之兩手必敗著手	52
圖 A-1 九層三角殺棋編碼表	55

附表目錄

表 2-1 一至七層三角殺棋勝負情形列表(取得最後一子為敗).....	5
表 2-2 旗子編碼與各位元對應表	9
表 2-3 位置關係連傑表	11
表 3-1 各層三角殺棋敗盤面總量分析表	15
表 3-2 棋子數與記憶體需求對照表	20
表 3-3 各預先選取盤面的可行著手數及可行著手所需記憶體空間	24
表 3-4 預先選取之著手勝負與時間花費	24
表 3-5 三角殺棋一至八層勝負表(取得最後一子為勝).....	31
表 3-6 盤面勝負關係組合	34
表 3-7 各層數淺層搜尋之結果與所需時間	43
表 4-1 第一區段之組合與相互關係	47
表 4-2 旋轉等價之 Hash Table 所需空間表	49
表 4-3 八層三角殺棋實作之空間與時間需求	50
表 5-1 已知的三角殺棋勝負情形	53
表 A-1 九層角殺棋必敗著手(單線表示先手之敗著，雙線表示後手對應之勝著)..	56

第一章 緒論

第一節 前言

三角殺棋相傳最早來自於中國，當時人們用小石頭堆砌成數堆進行遊戲，進而而有規律的發展出三角殺棋這麼一套有趣的遊戲。三角殺棋是 Nim[1][2]的一種變形，於坊間，有些網站[3]也有提供三角殺棋的遊戲服務。三角殺棋是兩人進行的一種益智遊戲，必須輪流取子，且不能跳過不取。遊戲棋盤為正三角形，三角形的邊長顆數即為此三角殺棋的層數。如圖 1-1 所示，該盤面為一個六層的三角殺棋。

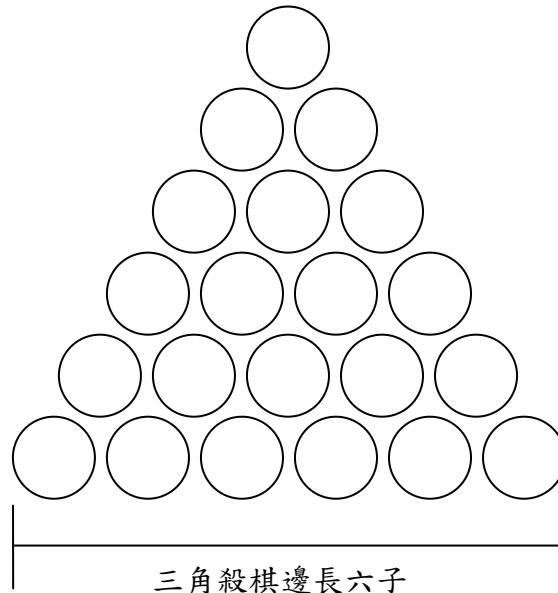
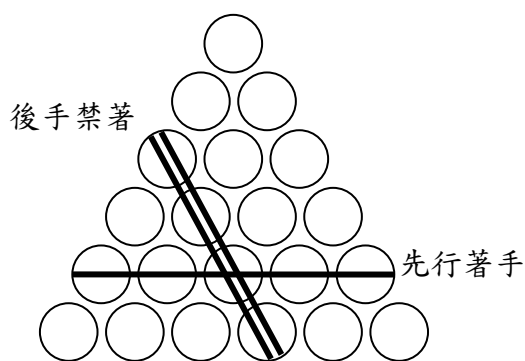


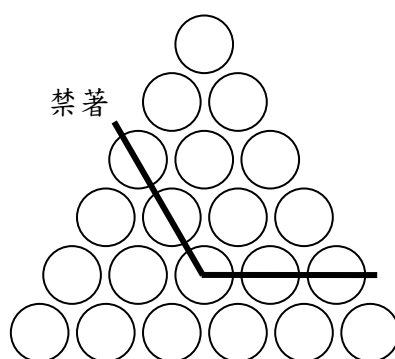
圖 1-1 六層三角殺棋

遊戲規則有兩種，一種是自盤面上取得最後一個子的玩家獲勝 (Normal Play)，一種是自盤面上取得最後一個子的玩家落敗 (Misere Play)。取子的方式為，必須在同一方向上取子，且取子數最少為一，至多為該方向上連續未被取過的子數。

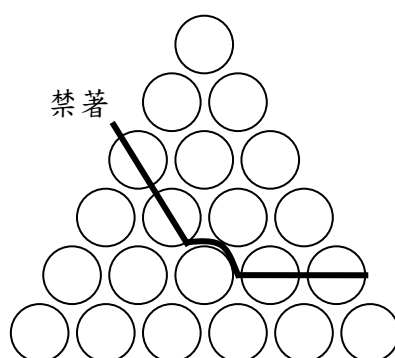
遊戲進行過程中，盤面上的棋子只會有取過以及未取過這兩種狀態，且取過的狀態將不能再被改變，也不能重複取子。常見的非合法著手如圖 1-2 所示。



選擇的著手重複選取到已取過的子



此著手所選取的子並未在同一方向上



此著手所選取的子並未在同一方向上且不連續

圖 1-2 非法著手

如圖 1-3 所示，當棋盤上的所有棋子皆為被取過的狀態，雙方皆再無合法著手時，遊戲便決定勝負了。由於此遊戲並無停著，因此三角殺棋的結局只有輸與贏兩種狀況，並無和局。

三角殺棋遊戲一旦破解，人類與電腦對弈是很難有機會獲勝的，因為電腦能夠分析所有盤面的狀態以及接下來的所有走步，得到最佳的應對策略。然而越高層數的三角殺棋盤面組合數越多，複雜度也越高。以八層三角殺棋來說，就有 2^{36} 種狀態需要計算與分析。

目前已知三角殺棋於取得最後一子為勝的規則下，僅二層三角殺棋為先手必敗，其餘一至八層皆為先手必勝；於取得最後一子為敗的規則下，二、四、六、七、八層三角殺棋為先手必勝。由這些資料來看，我們認為三角殺棋應是對先手較為有利的遊戲

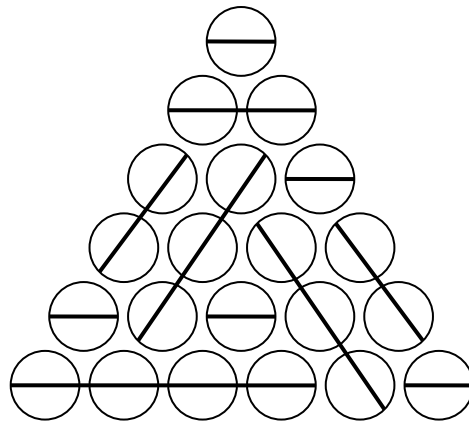


圖 1-3 所有棋子皆被取過的狀態

第二節 研究動機

七層三角殺棋於 1985 年被破解後，由於破解的複雜度極高，一直沒有其他的相關研究，直到 2009 年才同時有兩位研究生破解了八層的三角殺棋。雖然破解並不是根據以往的結果推估而來，但是以往的結論幾乎都偏向於先手必勝，也就是有機會能夠找到類似於三角殺棋的原型 Nim 遊戲一樣的規律性，讓三角殺棋可以藉由此規律性來讓所有層數的三角殺棋皆被破解。而我們希望所研究破解三角殺棋的演算法，其對效能增進以及空間節省的技术，也能為其他棋類遊戲所應用，在電腦人工智慧以及博奕領域中貢獻心力。

第三節 論文架構

本論文分為五章。第一章為緒論，包含前言、研究動機及論文架構。第二章介紹電腦三角殺棋相關的研究，簡述這些研究的基礎理論，如倒推法的程式設計、記憶體的控制等技巧。第三章為破解九層三角殺棋的一些概念與過程，以及演算法的設計。第四章為對八層三角殺棋的加速以及空間節省的改進策略。第五章為結論以及未來的研究方向，首度列出能夠於取得最後一子為敗之規則下，達成先手必勝的著手；於取得最後一子為勝之規則下，必定會落敗著手數，以及可以繼續研究的目標。

第二章 相關文獻及基礎理論

第一節 相關研究成果

以下介紹具代表性的電腦三角殺棋相關研究成果：

1. 以倒推法證明七層三角殺棋之勝負

在許舜欽教授的論文中[4][5]，因為該遊戲複雜度高，用估值函數或是 MIN-MAX 搜尋，抑或是配上 α - β 切捨等方式都難以分析解決，所以使用倒推的方式來達到完全破解。倒推法完全利用了電腦能夠大量運算以及大量記憶的特性，將七層三角殺棋共 2^{28} 種狀態完全計算出來，找出必勝著手以及必勝的對弈策略。

許舜欽教授利用倒推法解開了一至七層三角殺棋的勝負問題，遊戲規則為取得最後一子為敗。其勝負狀況如表 2-1 所示。

表 2-1 一至七層三角殺棋勝負情形列表(取得最後一子為敗)

三角殺棋層數	勝負狀況
一層	先手敗
二層	先手勝
三層	先手敗
四層	先手勝
五層	先手敗
六層	先手勝
七層	先手勝

許舜欽教授使用的機器為 VAX-11/750，記憶體為 32M Bytes，共花費三天半的時間求出七層三角殺棋的結果為先手勝。

2. 利用記憶體控管來加速破解八層三角殺棋

在白聖群研究生的論文中[6]，八層三角殺棋的狀況共有 2^{36} 種，若每個盤面狀況皆以 1bit 來儲存，也需要 8G Bytes 的記憶體。白聖群研究生使用的機器為 AMD Athlon64 X2 4000+ 2.1GHz 的 CPU，8G Bytes 的記憶體。就硬體限制而言，若程式直接使用 8G Bytes 的記憶體勢必會大幅度的降低速度，因為程式立刻會用滿記憶體來記錄所有狀況，一定要動用虛擬記憶體。為解決這種狀況使用了分段式的記憶體配置，如圖 2-1 所示。記憶體於計算最後的區段時才會動用到虛擬記憶體，因此能在有限的記憶體容量下，有效的降低破解八層三角殺棋的時間。

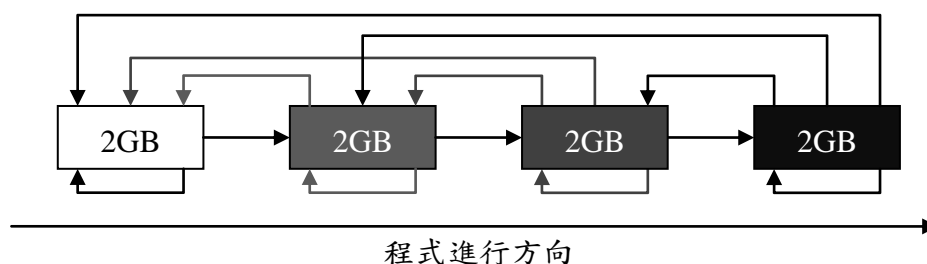


圖 2-1 分段式記憶體配置

白聖群研究生花費了 14.56 小時破解了八層三角殺棋，得到了取得最後一子為勝以及取得最後一子為敗的規則下，皆為先手必勝的結果。

3. 利用回溯分析法解八層三角殺棋

在林宏軒研究生的論文中[7]，使用回溯分析的方法來改良倒推法，加速破解了八層三角殺棋的勝負問題。回溯分析法充分利用了必敗盤面的特性，預先更新了未知盤面的勝負狀況，如圖 2-2 所示。大幅度的節省了計算量。林宏軒研究生使用的機器為 2.7GHz 的 CPU，2G Bytes 的記憶體，由於共需處理 8G Bytes 的資料量，所以將每 0.5G Bytes 的資料寫入硬碟，並視情況更新。共花費了 2.6 小時將八層三角殺棋破解。

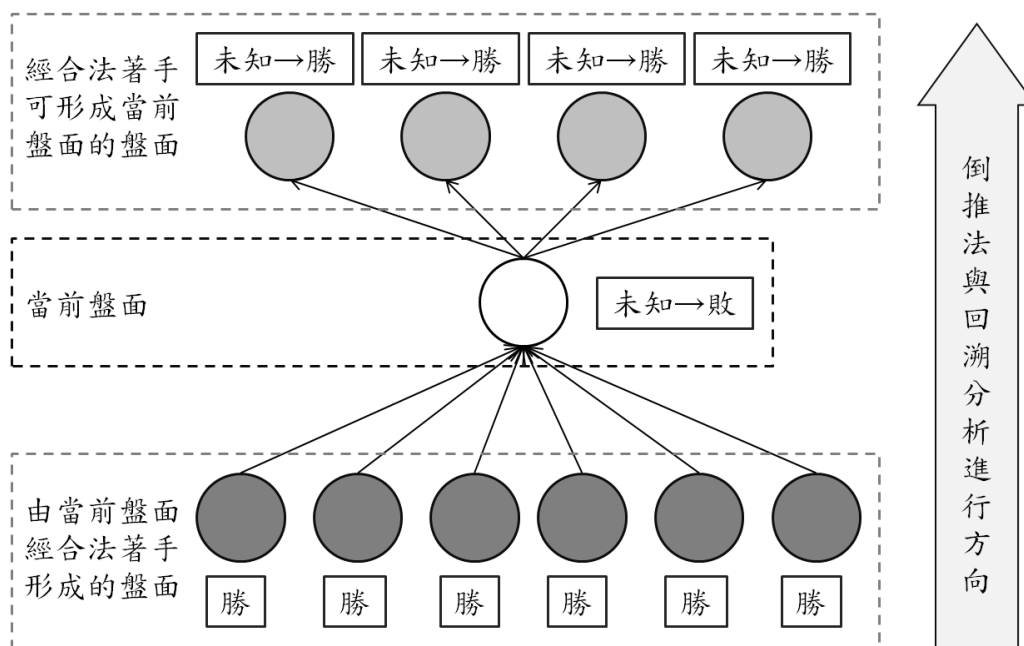


圖 2-2 回溯分析法預先更新盤面的狀況

第二節 倒推法

倒推法是許舜欽教授提出用來破解七層三角殺棋的演算法，該演算法的時間複雜度為 $O(2n^{(n+1)/2} \times n^2(n+1)/2)$ ，其中 n 為三角殺棋的層數。因此層數越高的三角殺棋，其時間複雜度以及所需的儲存空間也會以驚人的速度成長。單就九層三角殺棋來說，若完全以傳統的倒推法來處理，至少就需要 245 個位元來記錄所有狀態，也就是 4T Bytes 的記憶體或硬碟空間。

倒推法歸納勝負的原理如圖 2-3 所示，由全部被取過的盤面開始倒推至全部未取過的初始盤面，每次只計算一種盤面狀態。若當前的盤面所有經由一個合法著手形成的盤面狀態皆為先手必勝，則當前盤面的狀態便是先手必敗。換句話說，在目前盤面的狀態下，我方所能選擇著手皆會形成使對手勝的盤面，我方便會落敗。若當前盤面經由某一合法著手後，形成的盤面狀態為先手敗，則當前盤面的狀態便是先手勝。意即當前盤面的狀態下，我方必定有一種著手可以令對手落敗。

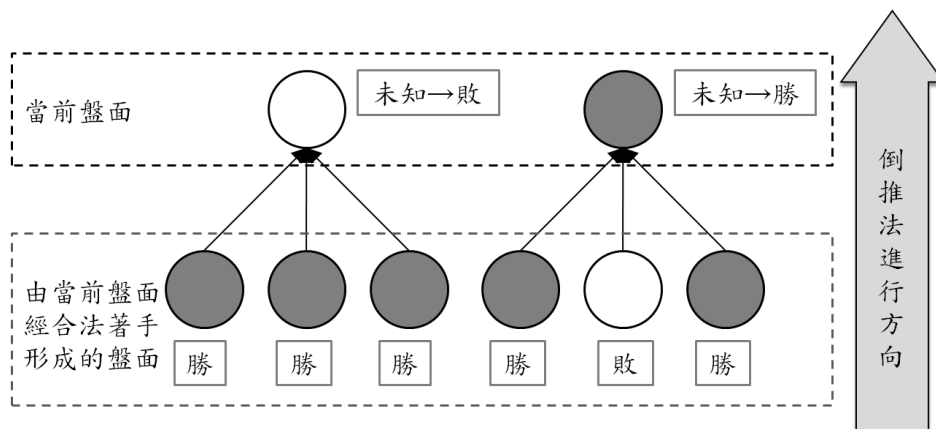


圖 2-3 倒推法歸納勝負的原理

由於三角殺棋在遊戲進行中，盤面上的棋子僅會有「取過」以及「未取過」兩種狀態，所以只需要一個位元便足以表示之。因此有 45 子的九層三角殺棋，我們可以用 45 位元來表示整個未取過的盤面到全部取過的盤面，共 2^{45} 種盤面狀態，所有狀態對應的整數值為 $0 \sim 2^{45} - 1$ 。各棋子對應之位元如圖 2-4 所示。

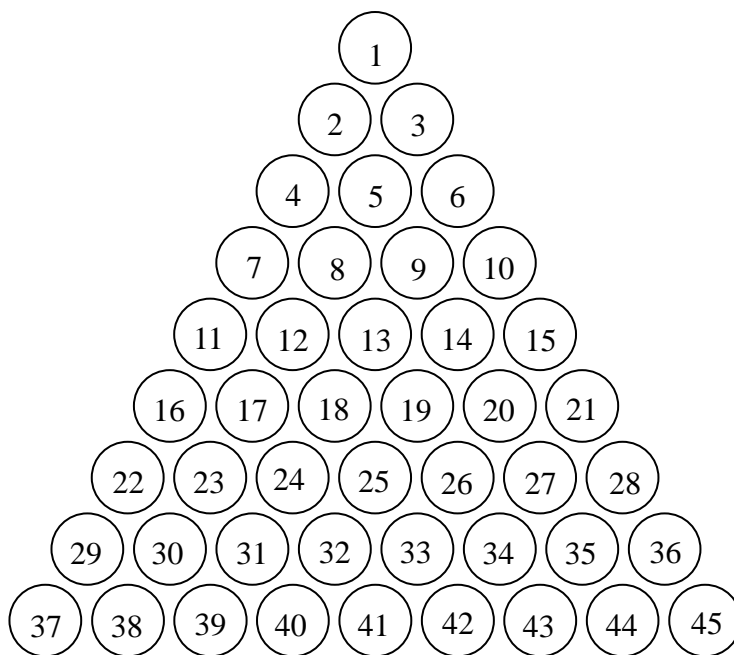


圖 2-4 九層三角殺棋編碼圖

編碼後任意盤面狀態皆可以一個 45 位元的二進位數表示之，其對應方式如表 2-2 所示。

表 2-2 旗子編碼與各位元對應表

棋子編號	45	44	43	...	5	4	3	2	1
對應位元	45	44	43	...	5	4	3	2	1

如此全被取過的盤面狀態便可由 $2^{45}-1$ 表示之，而全未被取過的初始盤面也可用 0 來表示。利用許舜欽教授提出的倒推法演算法[5]將盤面狀態由 $2^{45}-1$ 倒推至 0，便可以決定全部未被取過的初始盤面為必勝或必敗。倒推演算法如圖 2-5 所示。

```

S(  $2^{45}-1$  ) = 必勝;
for( i= $2^{45}-2$ ; i>=0; i-- ){
    S(i) = 必敗;
    for( j=1; j<=405; j++ ){
        if( 第 j 著手合法 ){
            求出 S(i) 下第 j 著手之後的盤面狀態;
            if( 下完 j 著手後的盤面狀態為必敗 ){
                S(i) = 必勝;
                break;
            }
        }
    }
}

```

圖 2-5 倒推法演算法

圖 2-5 表示的演算法是用來解取得最後一子為敗的規則，若要解取得最後一子為勝的規則，只需將初始第一行的 $S(2^{45}-1)$ 設為必敗即可。其原因以直觀的概念表達的話，若規則為取得最後一子為敗，則全部被取過的盤面我方必然取不到最後一子，所以該盤面狀態為必勝；若規則為取得最後一子為勝，則全部被取過的盤面我方亦取不到最後一子，所以該盤面狀態為必敗。

便可以完成。如要確認某一著手對當前的盤面來說是否合法，只需將該著手與盤面狀態作 AND 運算，若結果不為 0 則代表不合法，否則為合法著手。如要將當前盤面與一合法著手結合成為下一狀態的盤面，只需將該合法著手與當前盤面作 OR 運算即可得到新的盤面值。如需推知哪些盤面經由合法著手後可以形成當前的盤面，只需將當前盤面與該合法著手作 AND 運算，若其結果之值與該合法著手所代表的值相同，則將當前盤面與該合法著手作 XOR 運算，即可得到能形成當前盤面的新盤面。圖 2-7 整理了這些邏輯運算的用途與結果。

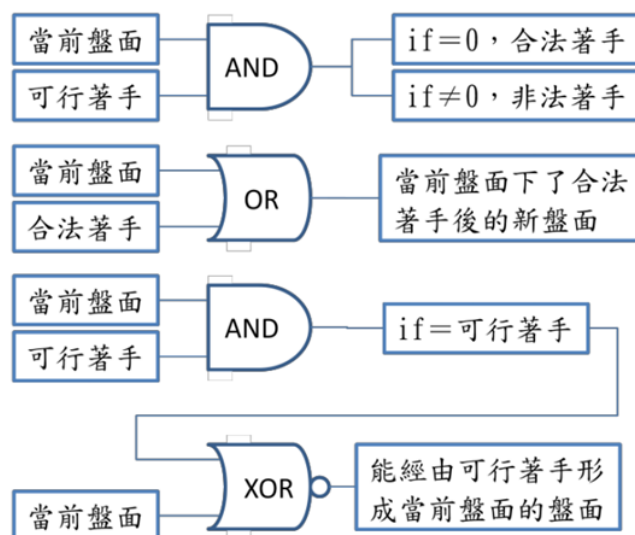


圖 2-7 邏輯運算的利用法及結果

確定了可行著手的表示法後，我們要建立一個位置關係連結表，記錄並列出每個棋子的右方、左下方以及右下方的棋子，因為三角殺棋僅能於這三個方向上進行取子的動作。位置關係連結表是為了檢查各個方向上是否仍有相鄰的棋子可取，依據此表來產生可行著手，並避免出現非法的著手。位置關係連結表如表 2-3 所示，其中 0 代表該方向上無相鄰的棋子。

表 2-3 位置關係連結表

棋子編號	右方	左下方	右下方
------	----	-----	-----

1	0	2	3
2	3	4	5
3	0	5	6
4	5	7	8
5	6	8	9
6	0	9	10
7	8	11	12
8	9	12	13
9	10	13	14
10	0	14	15
11	12	16	17
12	13	17	18
13	14	18	19
14	15	19	20
15	0	20	21
16	17	22	23
17	18	23	24
18	19	24	25
19	20	25	26
20	21	26	27
21	0	27	28
22	23	29	30
23	24	30	31
24	25	31	21
25	26	32	33

26	27	33	34
27	28	34	35
28	0	35	36
29	30	37	38
30	31	38	39
31	32	39	40
32	33	40	41
33	34	41	42
34	35	42	43
35	36	43	44
36	0	44	45
37	38	0	0
38	39	0	0
39	40	0	0
40	41	0	0
41	42	0	0
42	43	0	0
43	44	0	0
44	45	0	0
45	0	0	0

完成位置關係連結表後，我們可以根據許舜欽教授提出的演算法，依序將所有的可行著手產生，也就是將所有可行著手對應的值求出，如圖 2-8 所示。

n: 可行著手之值

Link: 位置關係連結表

```
for( i=1; i<=45; i++){
    n = 2(i-1);
    Output n;
    for( Direction = Right, Lower Left, Lower Right ){
        n = 2(i-1);
        j = i;
        for( k=1; k<=9; k++){
            if( Link( j, Direction ) == 0 ){
                break;
            }
            else{
                j = Link( j, Direction );
                n = n + 2(j-1);
                Output n;
            }
        }
    }
}
```

圖 2-8 可行著手產生演算法

第三章 九層三角殺棋之破解

第一節 利用必敗盤面的非完全資訊嘗試破解

由於利用第二章所提出的倒推演算法來硬解九層三角殺棋必須要使用 4T Bytes 的記憶體空間或是硬碟空間，幾乎是不切實際的做法。為了找尋新的三角殺棋解法，我們研究分析了三層三角殺棋至八層三角殺棋的解法以及結果，得到了必敗盤面量遠少於必勝盤面總量的結論，這一點於林宏軒研究生的論文中也有提及[7]。表 3-1 為我們統整出盤面勝負比例的分析結果。

表 3-1 各層三角殺棋敗盤面總量分析表

層數	盤面狀態總數	必敗盤面總數 (剔除旋轉對稱等價盤面)	必敗盤面 百分比(%)
3	$2^6 = 64$	6	9.375
4	$2^{10} = 1024$	36	3.516
5	$2^{15} = 32768$	683	2.084
6	$2^{21} = 2097152$	31414	1.498
7	$2^{28} = 268435456$	3300859	1.230
8	$2^{36} = 68719476736$	687194767	1

表 3-1 中我們統計了必敗盤面的總數，因為一個必敗盤面可以衍生出更多的必勝盤面，有較高的利用價值，而我們亦只需要記錄數量極少的必敗盤面。然而必敗盤面的總量依然不算少數，所以我們整理了三角殺棋的另一種特性，旋轉等價以及對稱等價的性質，來減少所需記錄的數量。

旋轉等價即是三角殺棋的任何盤面狀態，皆會與該盤面旋轉 120 度、240 度後形成的盤面狀態相同。也就是說，若盤面 A 之狀態為必敗，則盤面 A 旋轉 120 度形成的盤面 B，與旋轉 240 度形成的盤面 C 亦皆為必敗，如圖 3-1 所示。

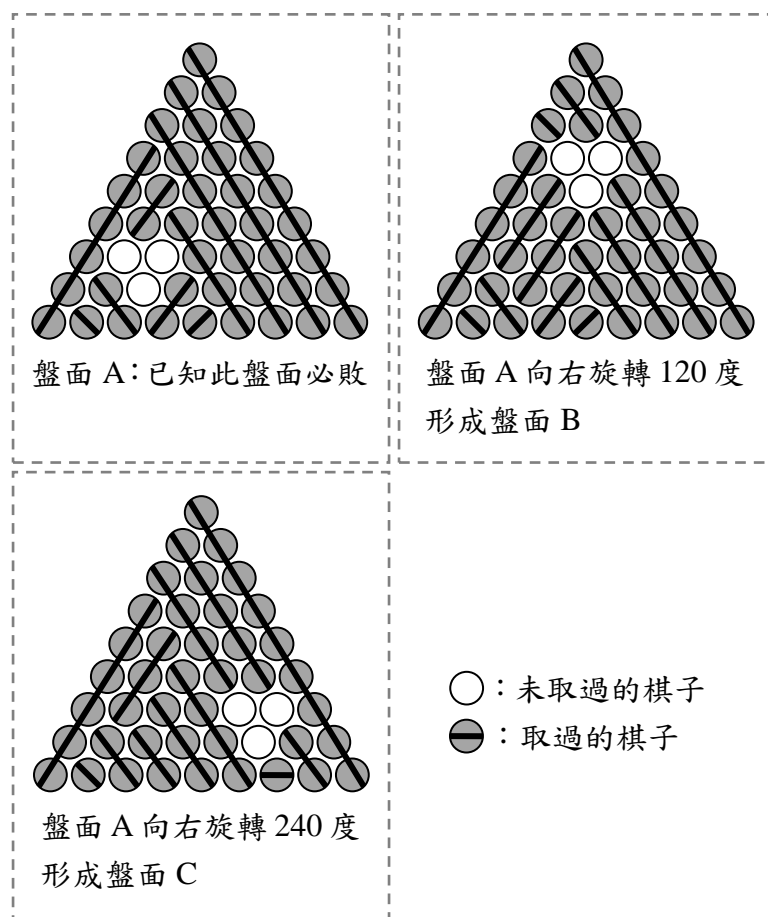


圖 3-1 旋轉等價的盤面

對稱等價則是指任意三角殺棋盤面狀態，會與其水平翻轉後形成的盤面之狀態相同。也就是若盤面 A 之狀態為必敗，則盤面 A 水平翻轉後形成的盤面 B 之狀態亦為必敗，如圖 3-2 所示。

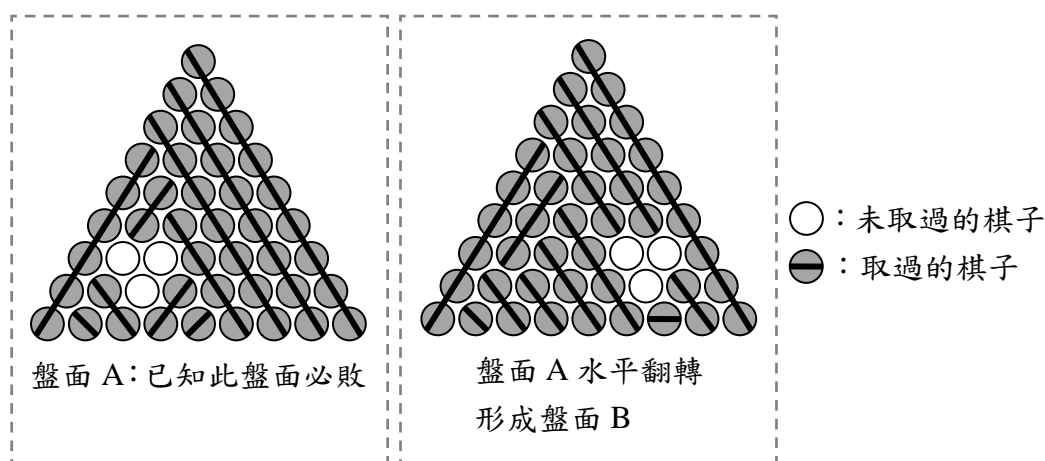


圖 3-2 對稱等價的盤面

經由這兩個特性，我們需要記錄的敗盤面數量可以達到表 3-1 所記錄的比例，而我們推估九層三角殺棋所需記錄的敗盤面數量，約佔盤面狀態總數的 1% 以內。但是以總數 $2^{45}-1$ 而言，即使是 1% 的量依然極大，且僅儲存必敗盤面勢必要將代表該盤面的值完整儲存，而不能僅以 1 個 Bit 來表示。粗略估計之，大約需要 $2^{45} \times 0.01 \times 64$ bits，也就是大約 2.8T Bytes 的記憶體空間，以現有的硬體實作依然不堪負荷，所以僅記錄必敗盤面依然是不可行的作法，但是旋轉與對稱等價的概念，仍然十分有用。

針對必敗盤面的利用，我們想出了較為可行的辦法，即是利用倒推法結合必敗盤面能夠衍生更多資訊的特性，將盤面狀態僅以「暫存」的方式記錄，進行九層三角殺棋的非完全資訊的破解，其流程如圖 3-3 所示。所謂的非完全資訊，即是我們所有的盤面勝負皆為暫時儲存，運算過後便不做任何記錄的。所以當我們找出必勝著手或是確認整個九層三角殺棋為先手必敗時，我們僅能知道先手必勝或是必敗的結論，而不能知道其必勝的走法以及整個盤面所有狀態下的勝負資訊。

這個方法的好處是，我們不需要檢查便能知道必敗盤面為何。我們先以人工的方式預設一個盤面為必敗，以取得最後一子為勝的規則為例，第一個必敗盤面便是全部被取過的盤面。將所有能經過一個合法著手後，能變成此盤面狀態的所有盤面存入暫存表中，表示這些盤面之狀態為必勝。而暫存表內的盤面必須由大

到小排序，且不能重複。當倒推到的盤面與暫存表中的盤面相同時，便將暫存表中相同的盤面移除。而當倒推到的盤面與暫存表內的所有盤面皆不相同時，即代表該盤面之狀態為必敗，需再次將所有能轉變成此當前盤面的所有盤面放入暫存表中。當倒推到的盤面狀態恰為一可行著手之狀態時，便能知道此可行著手是否為必勝著手。

由於必勝盤面的比例相當高，所以暫存表裡的資訊將會越來越龐大，且暫存表內的資訊必須由大到小排序，亦不能重複。所以暫存表的資料結構不能是單純的 linked list，否則檢查欲放入暫存表的盤面是否重複將會耗費極大量的時間。為了解決這個問題，我們採用的資料結構為 Skip list，如此可以將搜尋的時間由 $O(n)$ 降低至 $O(\log n)$ 。

然而此方法於 CPU 為 AMD Athlon64 3800+ 2.40GHz，記憶體大小為 1.93G Bytes 的電腦上，用於破解七層三角殺棋時，便已耗費數小時的時間，因為當倒推的流程到中段時，暫存表內的資訊會成長到一個極值，很容易超出記憶體容量的上限，動用到虛擬記憶體。所以此方法雖然不需要完全儲存所有資訊，但過程中仍不可避免用到大量的記憶體，因此需要再作改良，才有機會用以破解更高層數的三角殺棋。

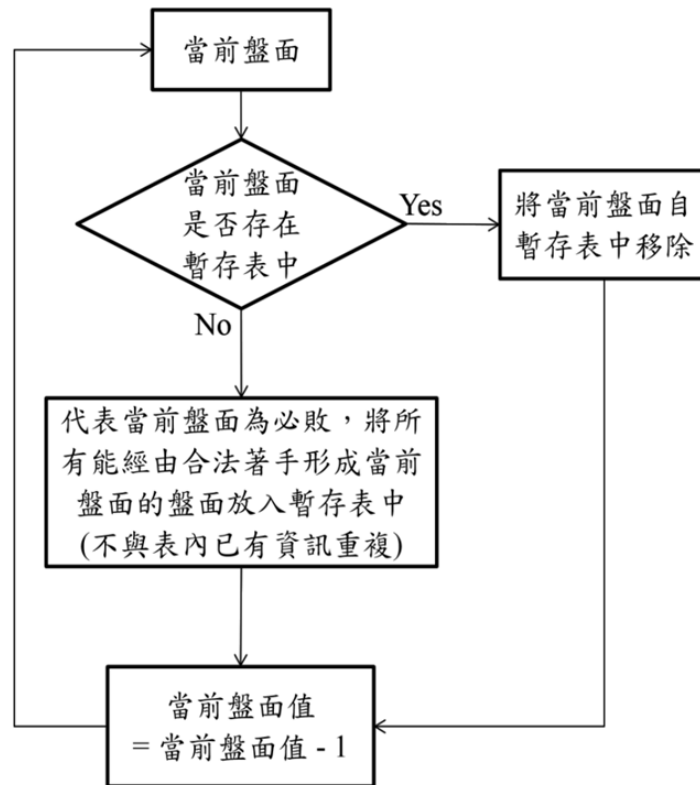


圖 3-3 非完全資訊破解流程

第二節 減少棋子數與重新編碼

本研究基本上延續了許舜欽教授於「利用電腦研究七層三角殺棋的勝負問題」論文中所提到的倒推法演算法[5]，以及林宏軒研究生於「八層三角殺棋之解法」中提到的回溯分析法[7]來加速運算，這是我們認為最有效率的，也最能提供完整資訊的做法。

但是僅以一般的倒推法，我們至少需要儲存 2^{45} 個位元的資訊，也就是 4T Bytes 的記憶體，若以現有的記憶體來實作，其動用到的虛擬記憶體將非常驚人，即使將資訊寫入檔案暫存，也會耗費相當大量的時間。

由於本研究使用的機器能提供 36G Bytes 的記憶體空間，為了能讓整個計算過程都不要接觸到虛擬記憶體，我們分析盤面棋子數多於 36 子後，其所需記憶體空間的總量，如表 3-2 所示。由此可知我們能夠處理的最多棋子數為 38 子，也就是共有 2^{38} 種盤面狀態，共需 32G Bytes 的記憶體空間。

表 3-2 棋子數與記憶體需求對照表

棋子數	36	37	38	39	40	41	42	43	44	45
記憶體需求 (G Bytes)	8	16	32	64	128	256	512	1024	2048	4096

換言之，我們可以從完整的九層三角殺棋棋盤中，預先取掉至少 7 子，至多 9 子來降低我們的空間需求。但是預先取子的作法僅能從可行著手的集合中尋找，而不能隨意取子，因為預先取掉一可行著手，意味著我們要先幫先手之玩家選擇一個著手，然後才去計算下了該著手的盤面狀態之勝負，若盤面狀態為必勝則可知該著手會導致先手必敗；若盤面狀態為必敗則可知該著手會令先手必勝。相對的，若隨機取子則獲得的盤面狀態將較不具利用價值。能夠選取的著手如圖 3-4 所示。

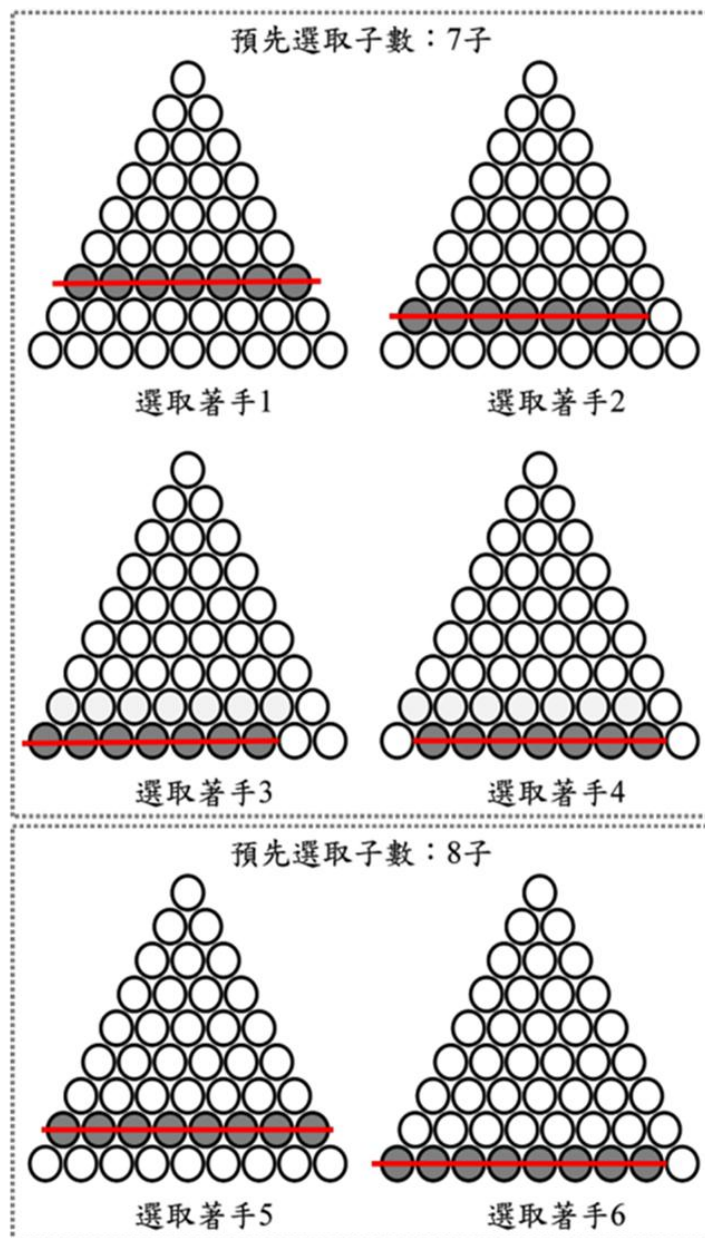


圖 3-4 預先選取的可行著手

我們不考慮預先選取 9 子的著手，因為一次取掉 9 子的著手只有取最後一層這種取法，勢必會形成八層三角殺棋，如圖 3-5 所示。而八層三角殺棋於取得最後一子為勝以及取得最後一子為敗兩種規則下，皆已被證明為先手必勝，所以選取 9 子的著手必定為必敗著手。

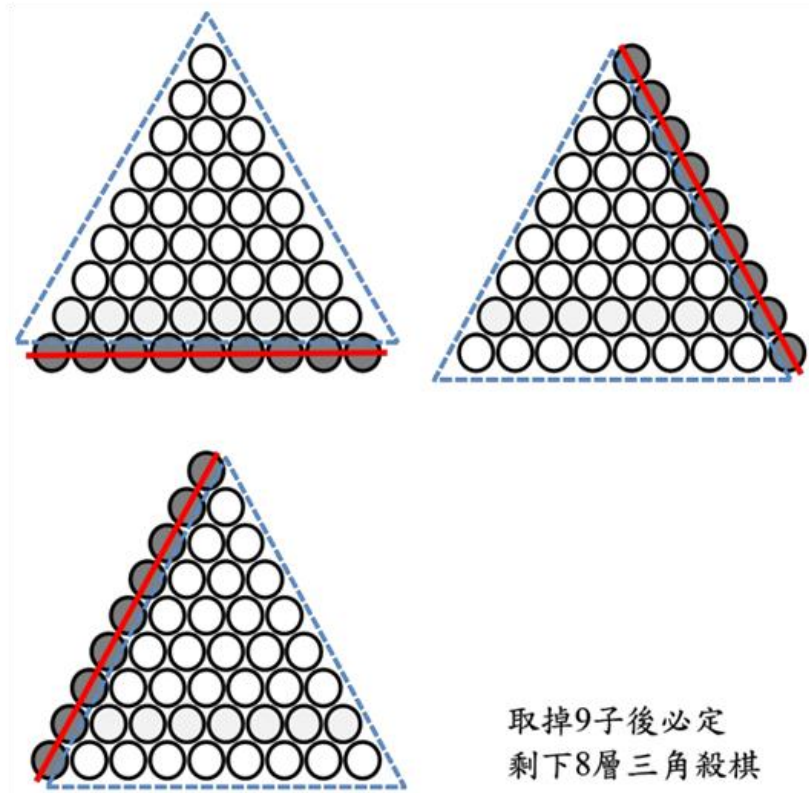


圖 3-5 選取 9 子的著手

經過預先取子之後，我們只需要花費至少 16G Bytes，至 32G Bytes 的記憶體空間，便能得知這六種著手於兩種規則下，為必勝著手或是必敗著手。但是經過預先取子後我們必須將盤面的編碼重置，因為預先選取 7 子時，剩餘盤面的狀態為 0 至 $2^{38}-1$ ；而預先選取 8 子時，剩餘盤面的狀態為 0 至 $2^{37}-1$ ，必須重新將各個棋子編碼，其編碼狀況如圖 3-6 所示。而重新編碼後的盤面，其位置關係連結表也必須重新製作，才能產生正確的可行著手。

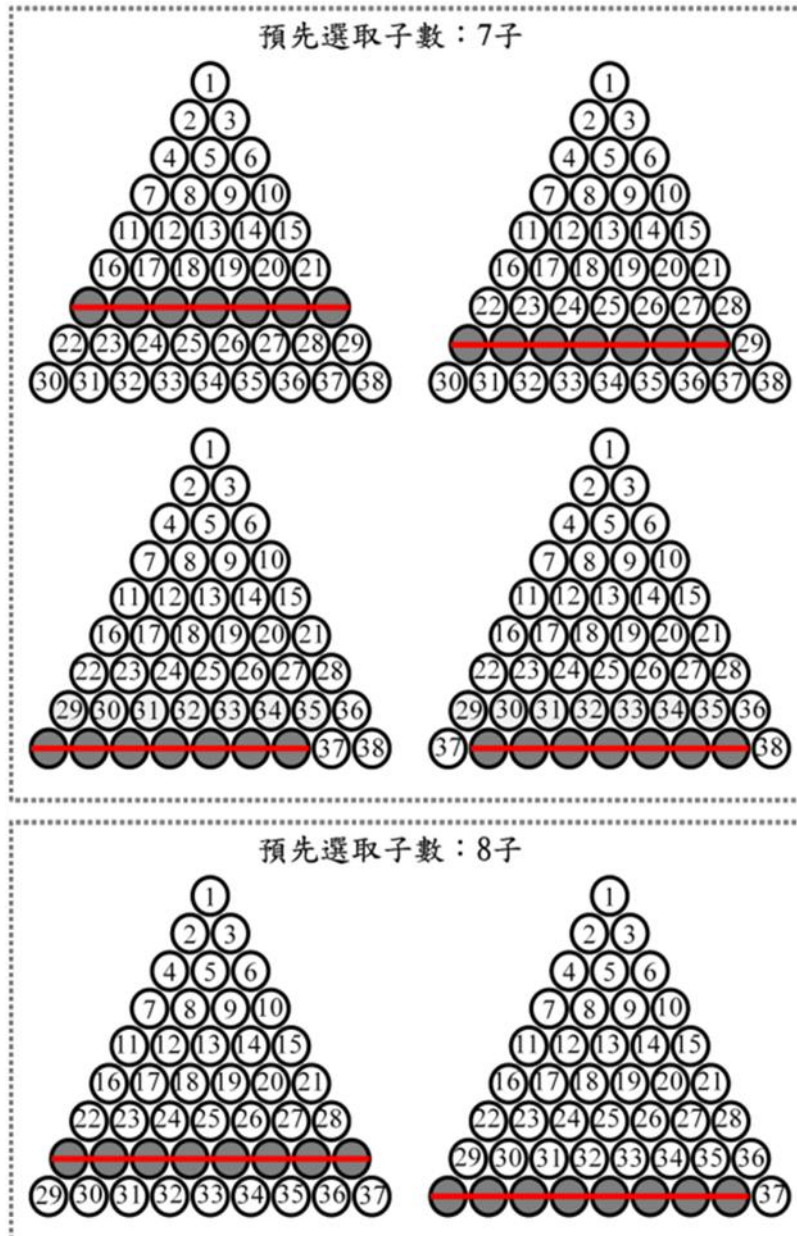


圖 3-6 九層三角殺棋預先取子後的重新編碼

我們必須針對這六種不同的盤面產生新的可行著手集合，各種盤面的可行著手數以及所需的記憶體如表 3-3 所示。

表 3-3 各預先選取盤面的可行著手數及可行著手所需記憶體空間

	可行著手數	可行著手集合所需記憶體空間
選取著手 1	223 種	1784 Bytes
選取著手 2	258 種	2064 Bytes
選取著手 3	307 種	2456 Bytes
選取著手 4	306 種	2448 Bytes
選取著手 5	241 種	1928 Bytes
選取著手 6	297 種	2376 Bytes

我們將這六種盤面以倒推法實作，而只需要找到一種著手可以令盤面狀況變為必敗，則可以直接宣布該預先取子的盤面為必勝，即預先選取的著手為必敗著手，無須將所有狀態全部倒推完。

這種作法僅能知道這六種著手的勝負狀態，倘若全部都得到為必敗的結果，便僅能宣告這些著手為必敗著手，而無法說明九層三角殺棋為先手必勝或先手必敗。表 3-4 列出了取得最後一子為勝以及取得最後一子為敗兩種規則下，六種預先選取著手的勝負情形以及找出勝負結果所耗費的時間。

表 3-4 預先選取之著手勝負與時間花費

	取得最後一子為勝	費時(秒)	取得最後一子為敗	費時(秒)
選取著手 1	必敗	72946.4	必敗	68831
選取著手 2	必敗	124238	必敗	119752
選取著手 3	必敗	97146.3	未實驗	未實驗
選取著手 4	必敗	122055	必勝	119152
選取著手 5	必敗	1029.23	必敗	1002.43
選取著手 6	必敗	32237.5	必敗	0 (直接推知)

於先選取著手 6 的盤面狀況我們不需要計算便可直接知道答案，因為預先選取著手 5 的盤面狀況經由倒推法計算的結果，找到對手能夠對應的著手恰巧為著手 6，也就是當第一位玩家選取著手 5 為第一著手時，第二位玩家只要選取著手 6 為應著即可獲勝；反之第一位玩家若選取著手 6 為第一著手，第二位玩家依然能選取著手 5 為應著來取勝。

由表 3-4 可知，我們利用這個方法已經順利的找到在規則為最後一子為敗的三角殺棋勝負結果。該規則下的九層三角殺棋為先手必勝，使用了 32G Bytes 的記憶體空間，耗費 119152 秒（約 33.1 小時），確認選取著手 4 為必勝著手，如圖 3-7。由於已知必勝著手，表 3-4 中「選取著手 3」的部份我們就沒有再做實驗了。

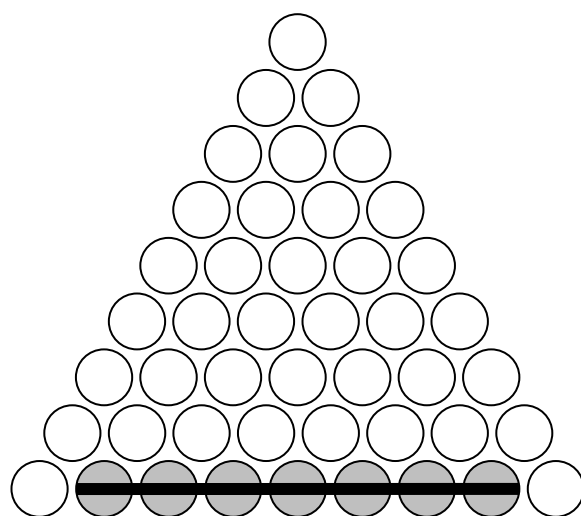
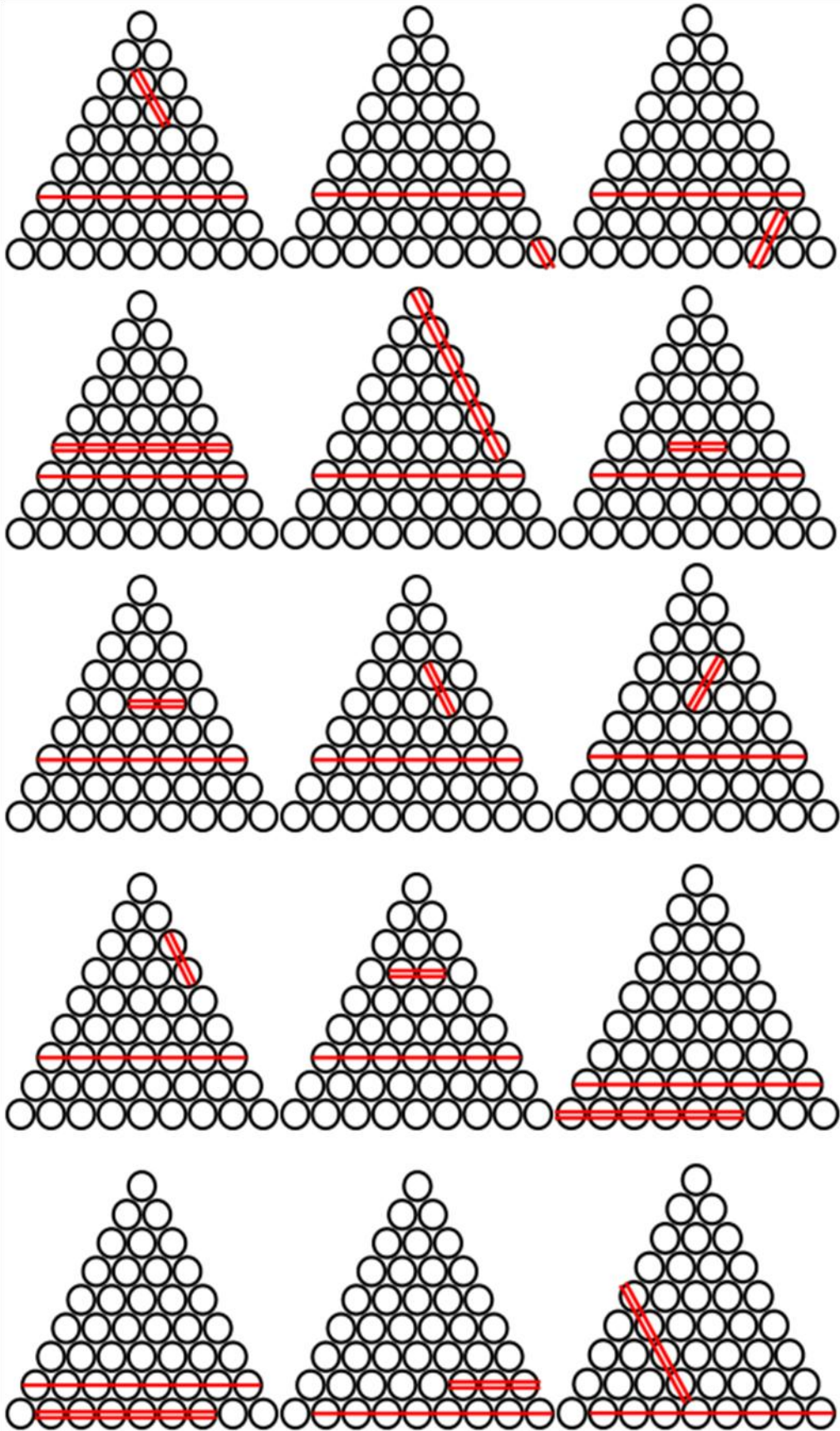


圖 3-7 規則為取得最後一子為敗，九層三角殺棋第勝的第一著手

而在取得最後一子為勝的規則下，雖然沒有在這六種預先選取的著手中找到必勝的答案，但是我們將這六種預先取子的盤面完全倒推，將對手所有可以應對取勝的著手找出，如圖 3-8 所示。如此便可找到一部份的必敗著手，縮小必勝著手的搜尋範圍。



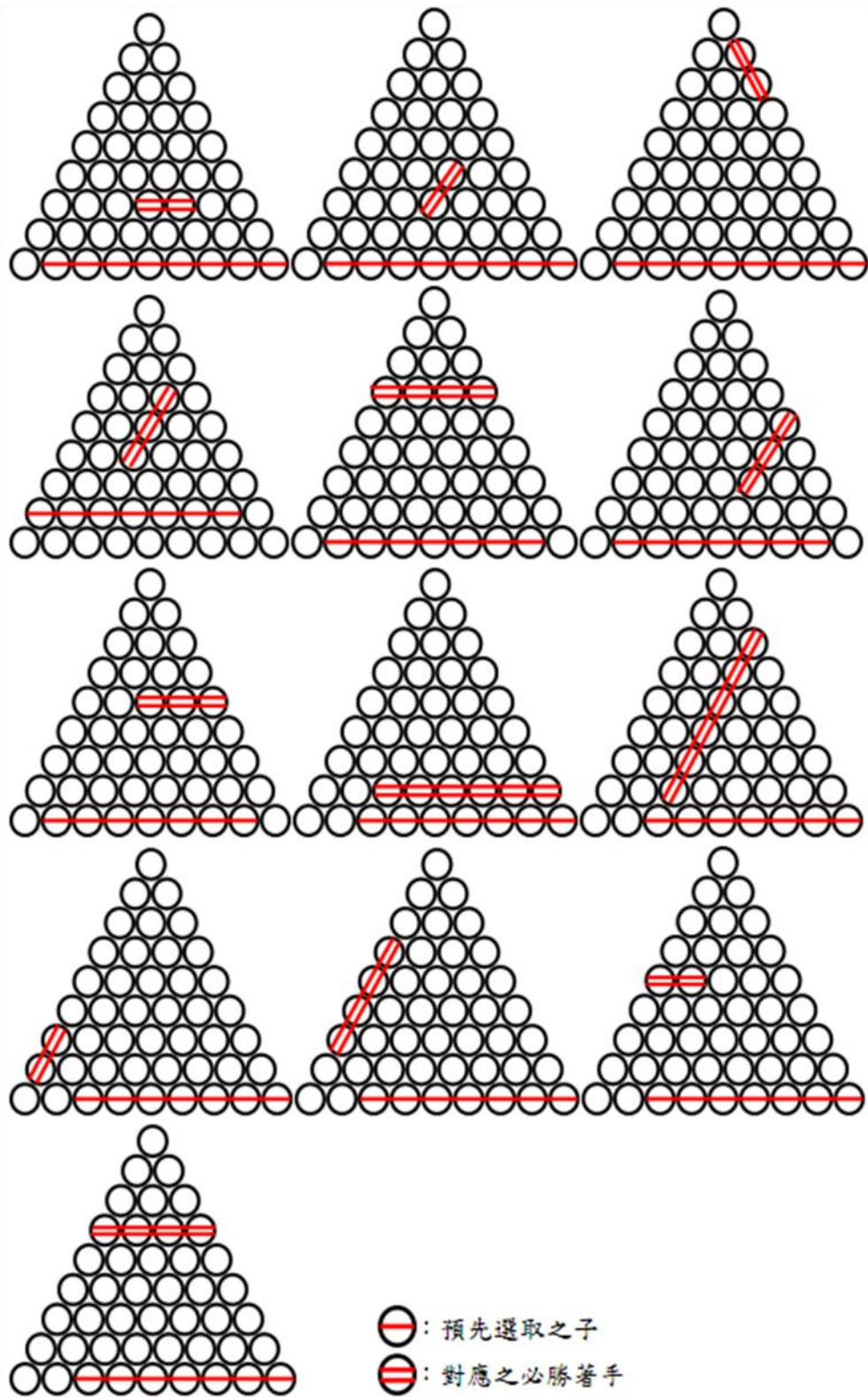


圖 3-8 預先取子與對應之必勝著手

第三節 Divide-and-Conquer

為了找出取得最後一子為勝的規則下，是否存在必勝的著手。參考白聖群研究生於「八層三角殺棋的勝負問題之研究」一文中的作法[6]，我們對現有三角殺棋之解法加以觀察分析，確認各個先手必勝的三角殺棋，其必勝的著手為何，期望藉此找出規律性，進而找到九層三角殺棋於取得最後一子為勝規則下的必勝著手。

三層三角殺棋之必勝著手，若排除旋轉等價及對稱等價的著手，共有 2 個勝著，如圖 3-9 所示。

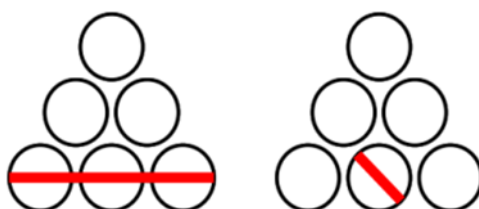


圖 3-9 三層三角殺棋之必勝著手

四層三角殺棋之必勝著手，若排除旋轉等價及對稱等價的著手，僅有 1 個勝著，如圖 3-10 所示。



圖 3-10 四層三角殺棋之必勝著手

五層三角殺棋之必勝著手，若排除旋轉等價及對稱等價的著手，共有 2 個勝著，如圖 3-11 所示。

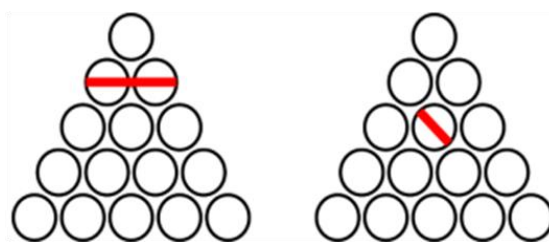


圖 3-11 五層三角殺棋之必勝著手

六層三角殺棋之必勝著手，若排除旋轉等價及對稱等價的著手，僅有 1 個勝著，如圖 3-12 所示。

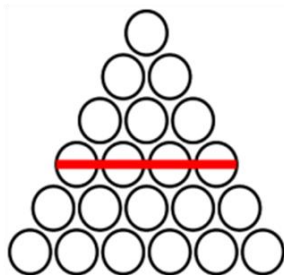


圖 3-12 六層三角殺棋之必勝著手

七層三角殺棋之必勝著手，若排除旋轉等價及對稱等價的著手，共有 4 個勝著，如圖 3-13 所示。

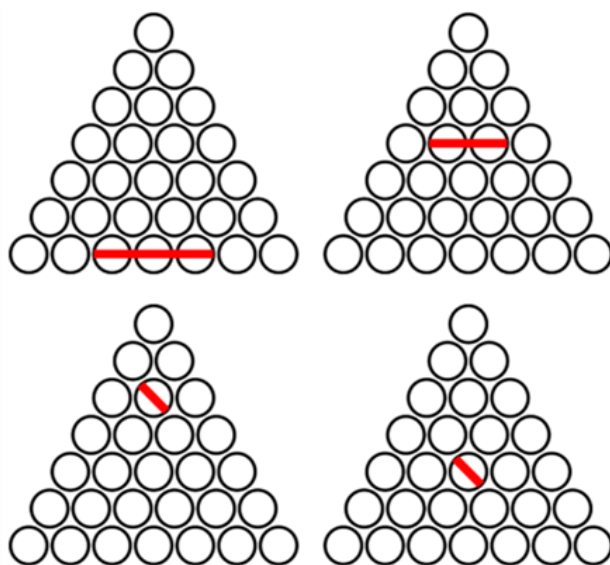


圖 3-13 七層三角殺棋之必勝著手

八層三角殺棋之必勝著手，若排除旋轉等價及對稱等價的著手，僅有 1 個勝著，如圖 3-14 所示。

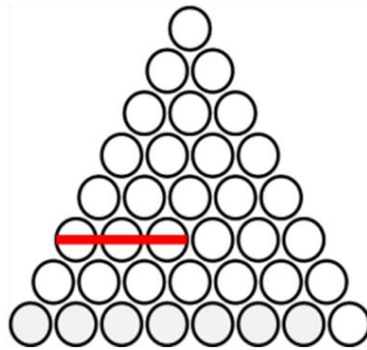


圖 3-14 八層三角殺棋之必勝著手

觀察各層三角殺棋的必勝著手後，發現大部分三角殺棋的必勝著手所取的棋子較少，若以預先取子的觀念來看，我們是無法處理殘餘子數過多的狀況的。然而六層三角殺棋的必勝著手仍值得我們去分析，該著手將盤面徹底的一分為二，讓一個大的遊戲盤面分割成兩個小型的遊戲，如圖 3-15。我們認為探究兩組遊戲的組合方式或許能讓我們求得答案，因此我們決定應用 Divide-and-Conquer 的概念來求解九層三角殺棋。

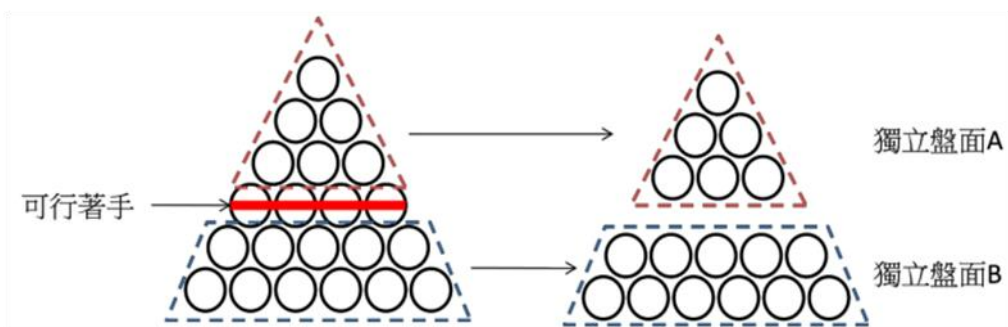


圖 3-15 Divide-and-Conquer 概念下的六層三角殺棋

白聖群研究生雖然於論文中提到[6]，必勝盤面與必敗盤面的組合仍為必勝，必敗盤面與必敗盤面的組合仍為必敗，但如表 3-5 所示，可以利用的必敗盤面僅有二層三角殺棋。即使砍除第三層，九層三角殺棋的下半部分仍有 39 子，我們無法對下半部的梯形盤面進行勝負分析。且該論文對於必勝盤面與必勝盤面的組

合卻沒有答案，所以我們考慮以別的方式來尋找兩個獨立的遊戲組合後，所產生的新遊戲之結果為何。

表 3-5 三角殺棋一至八層勝負表(取得最後一子為勝)

三角殺棋層數	勝負情形
一層	先手勝
二層	先手敗
三層	先手勝
四層	先手勝
五層	先手勝
六層	先手勝
七層	先手勝
八層	先手勝

第四節 利用 Sprague-Grundy Function 解九層三角殺棋

為了能有效利用 Divide-and-Conquer 的概念，我們參考了許多文獻以及網路上的許多方法[8][9][10]，其中 Sprague-Grundy Function 於 Nim 遊戲上分析的結果，能讓我們完全找出兩個獨立盤面之勝負以及兩盤面組合後形成的新盤面之勝負，這兩者之間的關係。

Grundy Function 的概念很簡單，如圖 3-16 所示。我們將一個 Nim 遊戲表示為 G ，而 G' 則代表了所有遊戲 G 經由一合法著手所能形成的盤面的集合，集合中的每一個盤面都是一個獨立的遊戲，將這些遊戲表示為 $G_1、G_2、G_3、\dots、G_n$ 。而每一個遊戲都有一個固定的數值，代表了這個 Nim 遊戲的勝負狀況，稱為 Grundy Number。我們將 $*G$ 表示為遊戲 G 的 Grundy Number，接著自 G' 集合中，

檢查所有遊戲的 Grundy Number，找到一個大於等於 0，且不存在於該集中的最小整數，該數字即為 $*G$ ，為遊戲 G 的 Grundy Number。若 $*G$ 恰等於 0，則代表遊戲 G 是一個先手必敗的遊戲，反之若 $*G$ 大於 0，則代表 G 是先手必勝。巧妙的是，若遊戲 G 恰可由彼此獨立的 G_A 以及 G_B 組成，則 $*G = *G_A \oplus *G_B$ 。

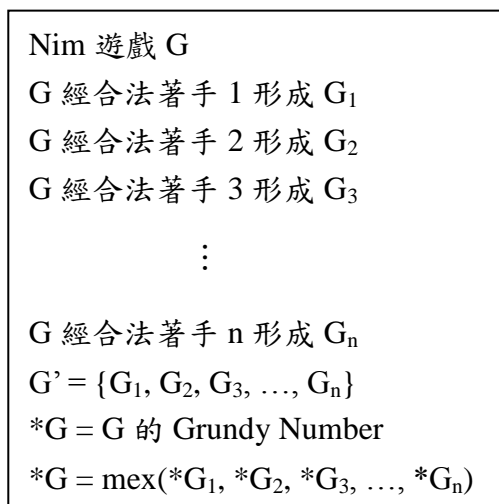


圖 3-16 Sprague-Grundy Function

我們嘗試將二層三角殺棋作測試，以倒推的方式產生各個盤面的 Grundy Number，如圖 3-17 所示，得到了正確的結果。

另外我們由 Grundy Function 中找到了組合的規則。若一盤面為兩個獨立子盤面的組合，則該盤面的值即為兩獨立子盤面之值作 XOR 運算的結果。也就是說，若兩個獨立子盤面皆為先手勝，且兩盤面之 Grundy Number 值恰相同，則組合後產生的新盤面必為先手敗，否則為先手勝。盤面勝負關係組合如表 3-6 所示。

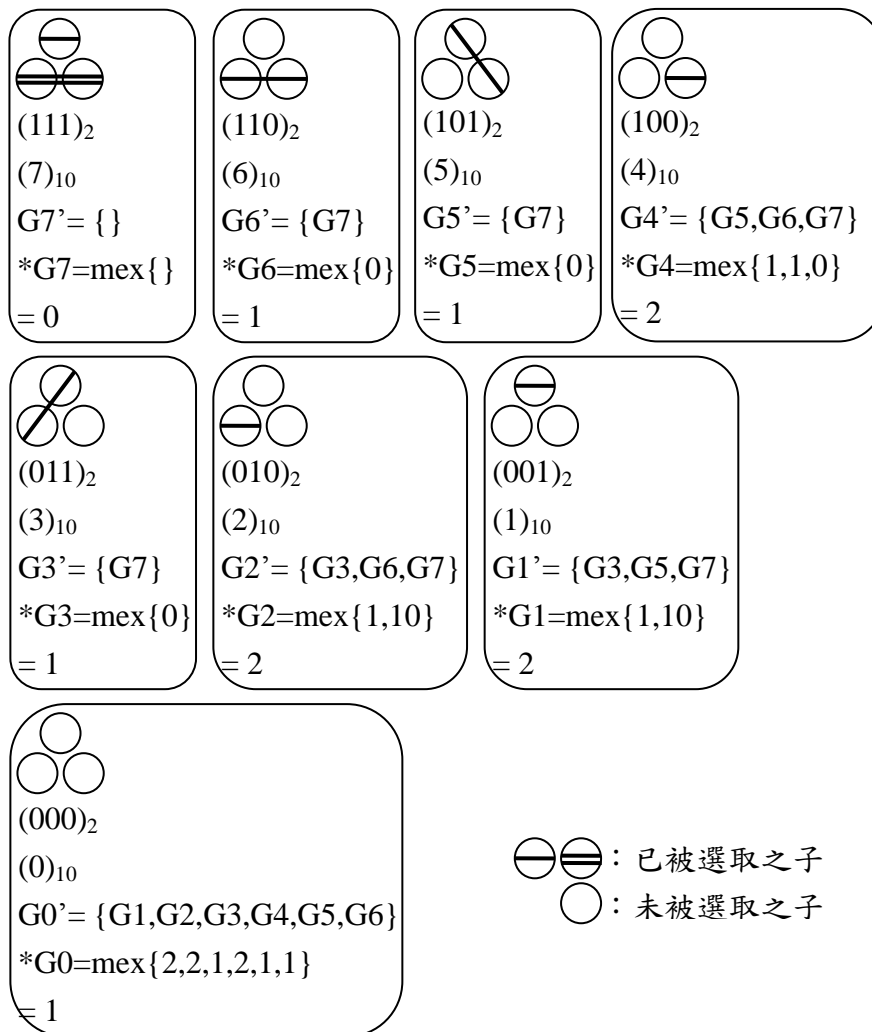


圖 3-17 二層三角殺棋各盤面狀態之 Grundy Number

表 3-6 盤面勝負關係組合

盤面 G_A 之 Grundy Number A	盤面 G_A 之勝負狀態	盤面 G_B 之 Grundy Number B	盤面 G_B 之勝負狀態	組合成之盤面 G 勝負狀態
$A > 0$	先手必勝	$B = 0$	先手必敗	$A \oplus B \neq 0$ 先手必勝
$A = 0$	先手必敗	$B = 0$	先手必敗	$A \oplus B = 0$ 先手必敗
$A > 0$	先手必勝	$B > 0, B \neq A$	先手必勝	$A \oplus B \neq 0$ 先手必勝
$A > 0$	先手必勝	$B > 0, B = A$	先手必勝	$A \oplus B = 0$ 先手必敗

歸納出盤面組合的結果後，我們便於硬體容許的範圍內，測試幾種可行著手，為確認該些著手是否為必勝的一步。由於將三角殺棋盤面分割後，會形成一個較少層的三角殺棋以及一個梯形的殺棋盤面，若一著手劃分出的梯形盤面過大，則必須要使用大量的記憶體，反之，若劃分出的三角殺棋盤面過大，亦會有此問題產生。因此我們限制最少取子數為五子，如此可以形成一個四層三角殺棋及一個殘餘 30 子的梯形殺棋盤面。餘下 30 子的盤面共有 230 種狀態，且每個狀態我們需要以一個 2 Bytes 的整數來記錄其 Grundy Number，估計需要 2G Bytes 的記憶體空間。而至多選擇八子，如此可形成一個七層三角殺棋與一個殘餘 9 子的一直線盤面。七層三角殺棋盤面需要 2^{28} 個 2 Bytes 整數，即 512M Bytes 的記憶體。

我們共選擇四種可將盤面一分為二，且互相不具旋轉及對稱等價關係的可行著手，如圖 3-18 所示。

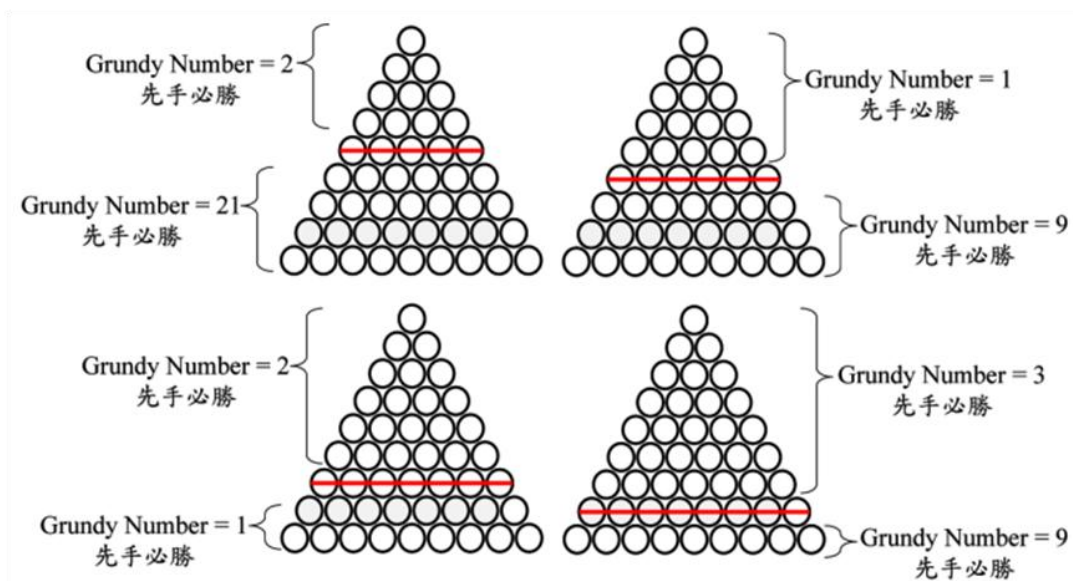


圖 3-18 選定將盤面一分為二的四種可行著手

然而這四種著手經過測試後，確認皆為必敗著手。雖然沒有成功，但是我們得到了不少 Grundy Number 的資料，且實際上我們發現，只需要儲存一組七層三角殺棋的 Grundy Number Table，以及一組上底為六層、下底為九層的梯形殺棋盤面之 Grundy Number Table，便可以完全分析這四種狀態。假設我們欲得知完全未被取過子的五層三角殺棋之 Grundy Number，我們僅需將七層三角殺棋的狀態視為第六層及第七層之子皆被取過，再將該盤面狀態對應到 Grundy Number Table 即可取得正確的值，如圖 3-19 所示。

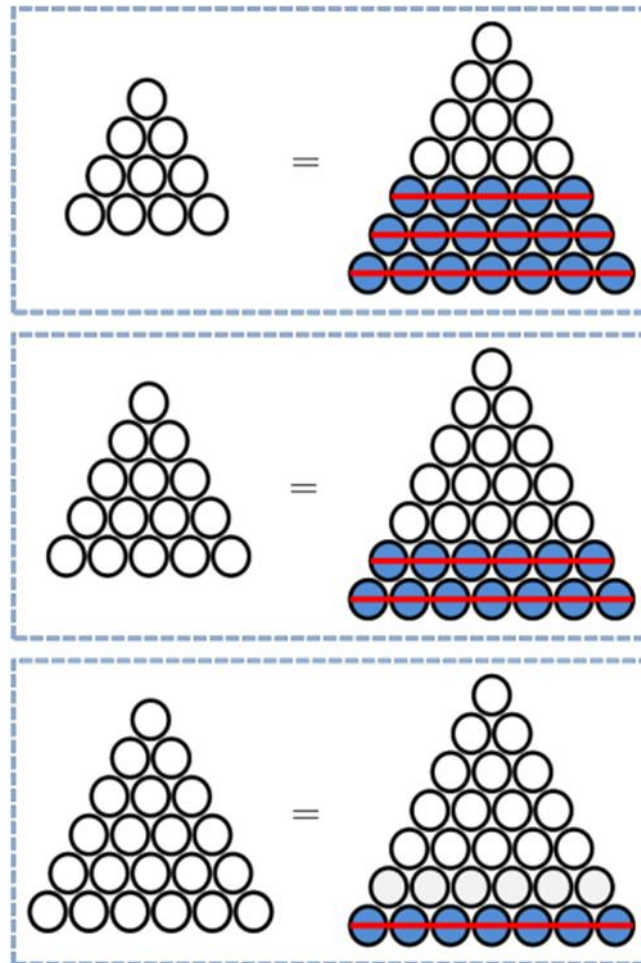


圖 3-19 Grundy Number 值相等的盤面狀態

我們蒐集以上的資訊，進一步尋找能夠妥善利用這些資訊及技術的演算法，以期能夠找尋到九層三角殺棋於取得最後一子為勝規則下的必勝著手。

第五節 淺層搜尋演算法

我們認為 Divide-and-Conquer 以及 Sprague-Grundy Function 仍然有其利用價值，因此我們設法製作了一個不同於倒推法的作法來求解九層三角殺棋。

若不使用倒推法，而是以傳統的方式暴力展開，則需自完全未被取過的盤面開始，挑選一個合法的可行著手並產生一個新的盤面狀態，如此往下伸展遊戲樹直至盤面為全部被取過的狀態，並回傳盤面的勝負值。其作法如圖 3-20 所展示的，盤面 A 可經由合法可行著手產生子盤面 B、子盤面 C 與子盤面 D。以深度優先的方式先選定子盤面 B，並再找尋合法可行著手，產生子盤面 E、子盤面 F 以及子盤面 G。若子盤面 E 經由合法可行著手後形成了全部被取過的全空盤面 H，則將子盤面 E 之狀態更新為必勝，並繼續做子盤面 F。若子盤面 E、F 及 G 之狀態皆為必勝，則盤面 B 之狀態為必敗。

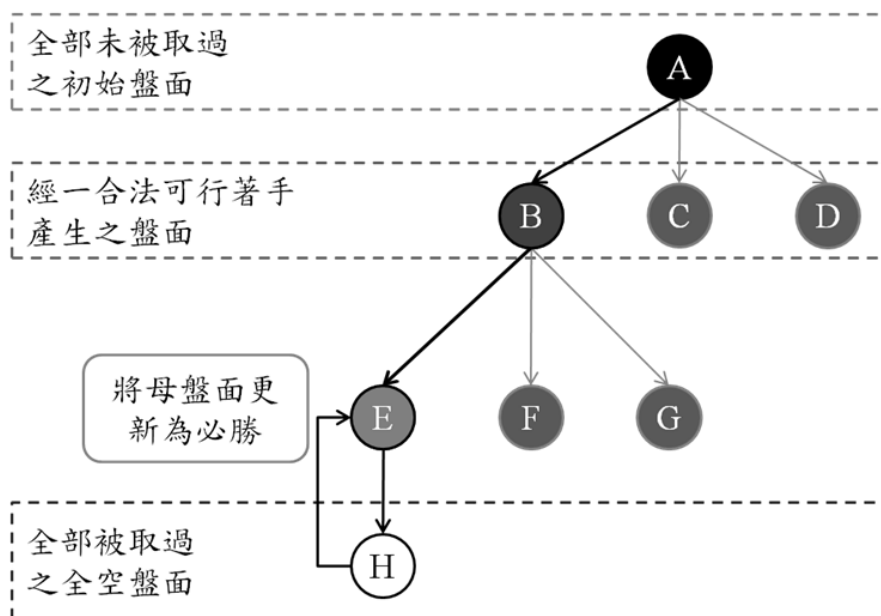


圖 3-20 暴力展開遊戲樹之作法

由暴力展開的流程可知，若單純的用這個方法求解，則總盤面狀態數為 2^{45} 種，可行著手數為 405 種的九層三角殺棋，其遊戲樹的高度在最差的狀況為雙方

輪流各取一子，共達 45 層。而遊戲樹寬度的成長也將非常驚人，若不改良此方法，將會耗費不可預期的時間。

要提早結束每一個分枝，就必須提早決定盤面的勝負值。我們需要在一固定的深度時，即判斷盤面的勝負情形，而不需要展至最底層才一一回傳盤面之勝負狀態。我們認為這麼做有機會讓我們提早得到三角殺棋的結果，這便是淺層搜尋。而要達成這個目的，便需要利用 Divide-and-Conquer 以及 Sprague-Grundy Function 的概念。

由於在遊戲進行過程中，我們可經由某些合法的可行著手，將遊戲盤面一分为二，所以我們可以較為主動的讓盤面被分割為兩個部份。而在允許的切割範圍下，我們可以利 Grundy Number 知道各個分割後盤面的勝負狀況，再經由 XOR 運算取得整體盤面的勝負狀況。

我們將預先存取的兩組 Grundy Number Table 拿來利用，也就是說，我們能夠分析的盤面即是第五層、第六層、第七層或第八層的所有棋子皆被取過的盤面，而其餘的棋子狀態則是可以被取過或是未取過，我們可以立即知道這些類型盤面的勝負狀態。我們要做的便是檢查某一盤面是否可以經由一個合法著手，將這四種層數的其中一層完全選取掉，從而獲得該盤面的勝負。由於可行著手有三個取子方向，因此我們可以檢查總共十二種取掉一整層的三角殺棋盤面，如圖 3-21 所示。

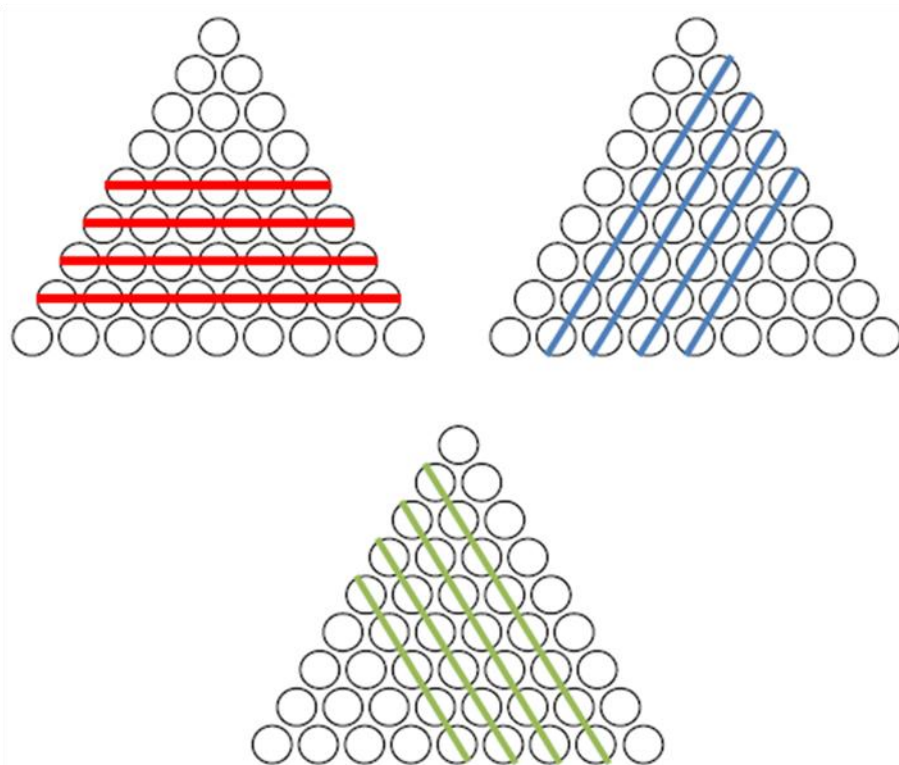


圖 3-21 十二種盤面切割檢查法

根據上述的方法，我們制訂了淺層搜尋的流程，並使整個搜尋結束於第 N 層，我們稱為 N 層淺層搜尋， N 為奇數。也就是最終檢查是否可以將盤面一分为二的動作，將由第一位玩家來執行。因為我們要控制第一位玩家為將盤面切割的人，才能提早確認該玩家最初所下的著手是否為必勝著手。整個淺層搜尋演算法的虛擬碼如圖 3-22 所示。


```

LowSearch( currentBoard, depth ){
  if( depth > 1 ){
    if( DivideCheck( currentBoard ) == LOSE)
      return WIN;//若當前盤面分割後為敗，則當前盤面狀態為勝
    else{
      for( i = 0; i < 405; i++ ){
        if( “move i” is legal ){
          nextBoard = ( currentBoard | move i );
          if( LowSearch( (nextBoard | move i), depth-1 ) == LOSE ){
            return WIN;
          }//若著手 i 合法，則將盤面與著手 i 結合並置入迴圈
        }
      }
      If all nextBoard’s state are “WIN”, return LOSE;
    }
  }
  else
    return ( DivdeCheck(currentBoard) );//若達最後一層，直接分割求勝負
}

DivideCheck( currentBoard ){
  for( i = 0; i<405; i++ ){
    if “move i” is legal, set nextBoard = ( move i | currentBoard );
    Check the 12 kinds of divideMoves:
    if( ( nextBoard & divideMoves1 ) == divideMoves1 ){
      find the Grundy Number of each divided board;
      if the Grundy Numbers of two divided boards are equal
        return LOSE;
    }
    else if( ( nextBoard & divideMoves2 ) == divideMoves2){ ... }
    else if( ( nextBoard & divideMoves3 ) == divideMoves3){ ... }
    ...
    else if( ( nextBoard & divideMoves12 ) == divideMoves12){ ... }

    return WIN;
  }
}

```

圖 3-22 淺層搜尋演算法的虛擬碼

我們先扮演第一個玩家的角色，選擇一個可行著手，代表了一個初始的盤面，並給予一個偶數的深度值。若該盤面所附加的深度大於 1 時，我們會先檢查這個盤面是否能夠經由一個合法著手產生切割的效果，且切割的方式必須與我們預設的 12 種切割法之一吻合。若可行，則將切割後的兩個子盤面擷取出，並對照預先產生好的 Grundy Number Table，找出兩個子盤面的 Grundy Number。若取得的兩個 Grundy Number 值相同，則表示我們切割這一刀會導致盤面狀態變為必敗，也就是這個著手為勝著。若我們找不到任何一種合法著手可以讓盤面被分割，或是分割後的盤面狀態皆不為必敗，我們就從所有的合法可行著手中選一個繼續往下搜尋，並將深度值減 1。當盤面附加的深度值恰為 1 時，代表我們將要執行最後一次的選取動作。也就是我們於這一層只需要檢查是否可以分割盤面，而分割之後產生的盤面狀態若為必敗，則可將分割前的盤面更新為必勝。若皆無法分割，或是分割後皆找不到必敗的狀況，則視分割前的盤面為必勝。

由於 N 層的淺層搜尋要證明某一著手為必勝著手，必須於該著手下的分支中，所有位於倒數第二層的盤面狀態皆為先手必勝。也就是說，這些盤面必須皆能經由一合法著手分割，而且分割後皆能找到至少一個必敗的結果，如圖 3-23 所示。這需要相當的搜尋層數才有機會讓不同盤面經切割後形成的盤面重疊，進而破解，得到一個必勝的著手。

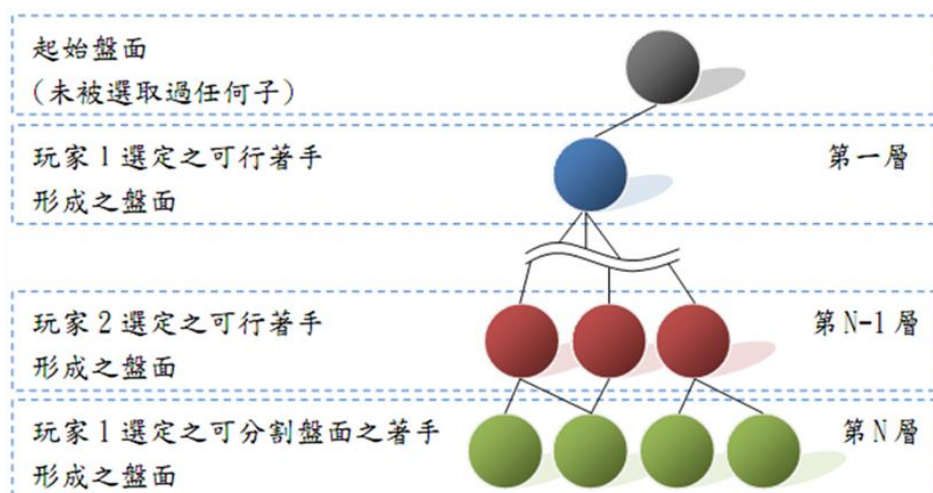


圖 3-23 N 層淺層搜尋之結構

無論淺層搜尋法或是暴力法皆會遇到可行著手重複搜尋的問題。若依照以往的方法，我們每一次搜尋一個狀態下的合法著手便需要將所有的可行著手分析一次，才能確認合法著手為何。然而當盤面上被選取過的子越來越多時，不合法的著手也就越來越多。以九層三角殺棋來說，可行著手共有 405 手，當盤面越往下分析到深層時，將會平白浪費許多時間在計算必不可行的著手上。因此我們特別建立了一組陣列，於每次選定一著手後，便將可行著手與該選定之著手做 AND 運算，若運算結果為 0 則放入指定的陣列中。當全部的可行著手都被處理好之後，我們將該陣列內的著手做為新的可行著手列表，交給下一層的盤面選取。如此我們不僅能減少運算時間，更不需要再做選定著手是否合法的檢查，因為新的可行著手列表內所提供之著手對當前狀態而言，必定皆為合法著手。其流程如圖 3-24 所示。

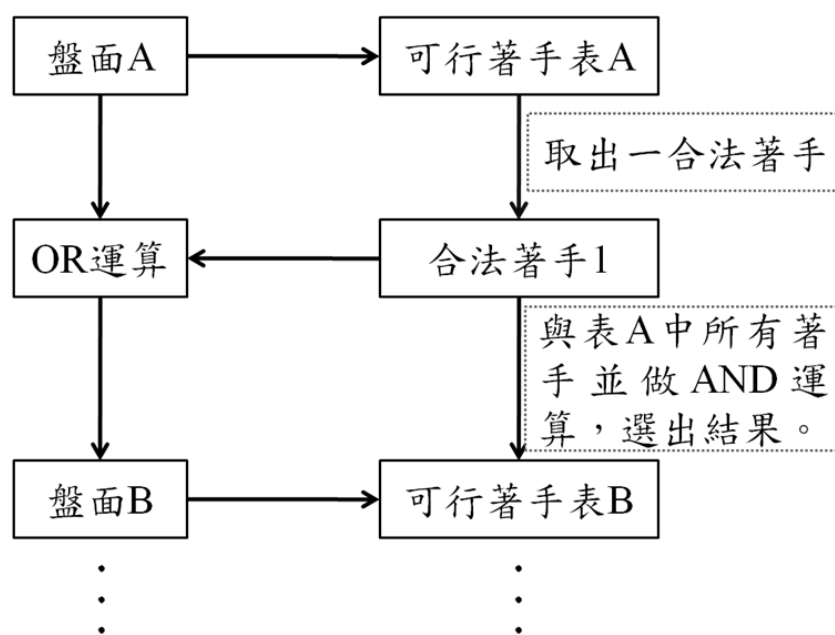


圖 3-24 可行著手表之更新流程

用了更新可行著手表的作法後，我們再將可以選擇的第一個著手總量進行壓縮。我們將第一個著手可選擇的可行著手表特別處理，剔除所有旋轉等價以及對稱等價的著手，如此第一步的可行著手將由 405 手縮減至 80 手。而選定著手後仍須將該著手與原先的 405 手做 AND 運算，才能正確的挑出所有合法可行著手

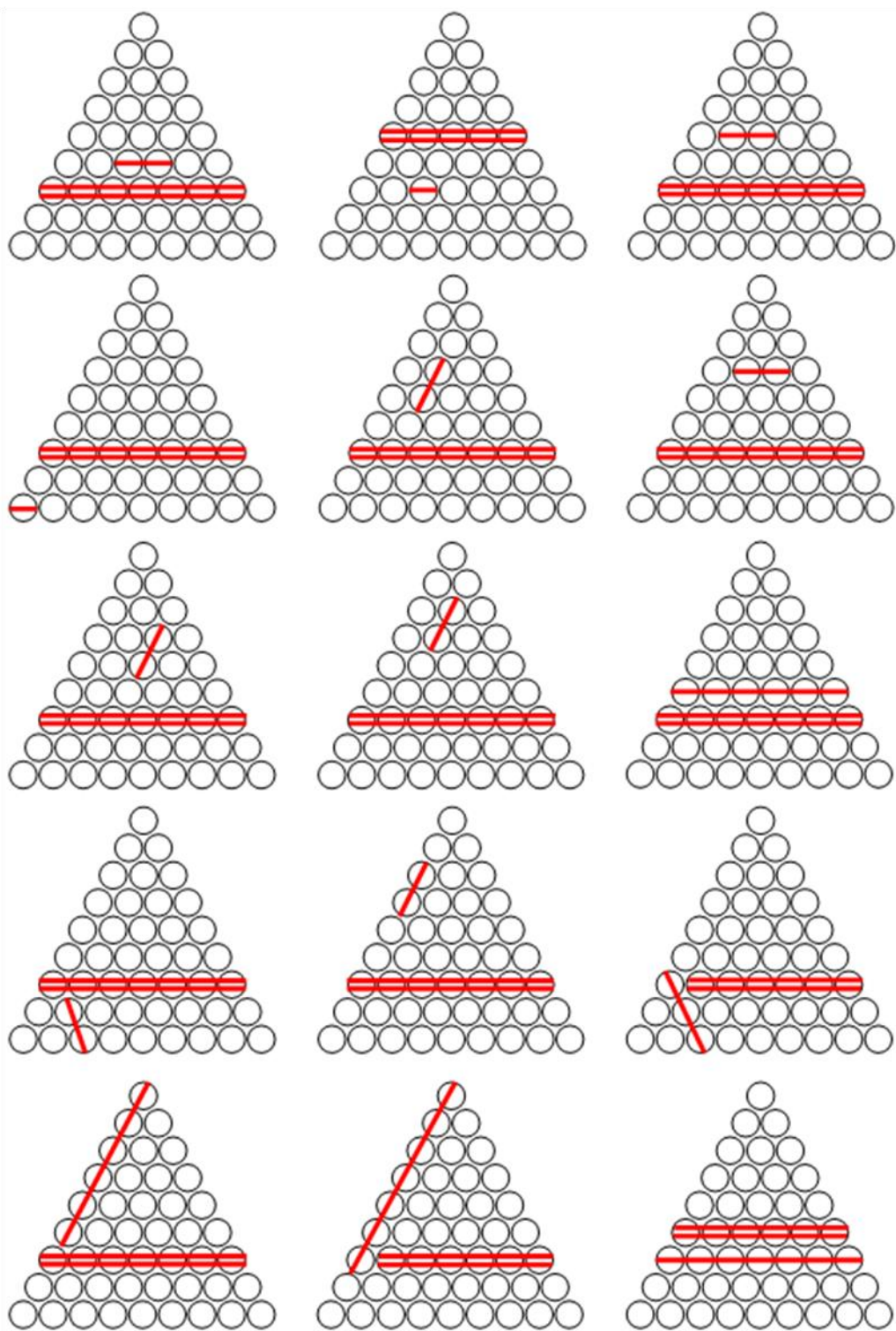
至新的可行著手列表。實驗淺層搜尋的所需時間大約減少了 50%，效果顯著的提升。而各層數淺層搜尋之結果以及耗費時間如表 3-7 所示。



表 3-7 各層數淺層搜尋之結果與所需時間

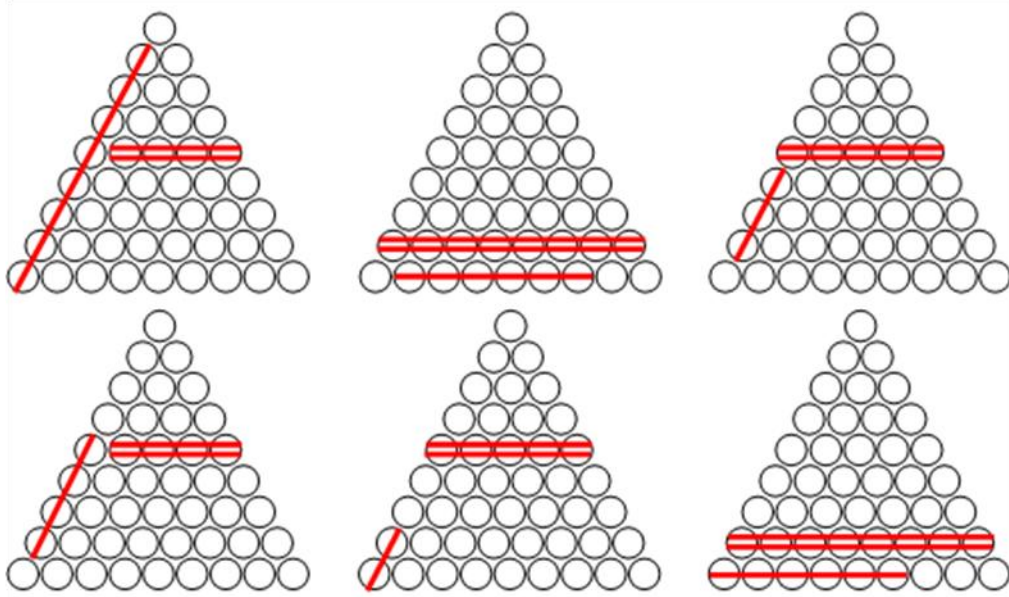
搜尋層數	搜尋結果	耗費時間
3	未找到必勝著手	少於 0.01 秒
5	未找到必勝著手	少於 0.01 秒
7	未找到必勝著手	5.68 秒
9	未找到必勝著手	332.82 秒
11	未找到必勝著手	9471.88 秒
13	未找到必勝著手	191597 秒

雖然我們將速度提升，但是較可惜的是仍未找到必勝著手，應是深度大於 13 層之後，各個盤面的分割後情況才有較高的重疊率，較能提升找到分割後盤面狀態為必敗的機率。

淺層搜尋雖然仍未提供九層三角殺棋於取得最後一子為勝的規則下先手必勝的結果，但是根據這個做法，我們可以提供更多的必敗著手，來更加縮小破解九層三角殺棋所需分析的範圍。因為三角殺棋的先手方之著手 A 若為必敗，則表示後手方必有一著手 B 甚至更多著手可將盤面之狀態變更為必敗；反之先手方若轉而取對手用以應對的著手 B 來當先行著手，後手方也只需將著手 A 作為對應著手即可。所以著手 A 以及著手 B 皆為必敗之著手。我們只要將淺層搜尋控制在第一層即可找到許多相應的著手。圖 3-25 為利用淺層搜尋的方法下，我們找到的所有必敗著手。



 : 必敗之可行著手
 : 淺層搜尋找出之對應著手



⊖ : 必敗之可行著手
 ⊕ : 淺層搜尋找出之對應著手

圖 3-25 淺層搜尋找出之必敗著手

第四章 八層三角殺棋之加速與空間之節省

第一節 三角殺棋的等價性質與複雜度

我們考慮先將八層三角殺棋的作法改良，在空間及效能上都有所改進後，才有可能再應用於更高層數的三角殺棋，進而對九層的三角殺棋破解產生幫助。我們於第三章有提到，三角殺棋具有旋轉等價以及對稱等價的性質，而這些等價性質很明顯的提醒了我們，在完全計算所有的三角殺棋盤面狀態時，其實做了很多重複的動作。也就是說，若能完全把等價的盤面移除，就有機會如同縮減九層三角殺棋之可行著手集合一般，將總量 405 手縮減至 80 手互相無等價關係的著手。其總量僅為原本的 2 成左右，整體計算上的複雜度便能大幅度的降低，能節省的時間以及所需的儲存空間必定相當可觀。

第二節 重新編碼

因為倒推法依然是目前為止最有效果的演算法，所以我們決定從倒推法著手改良。首先要變更的即是盤面編碼，我們利用編碼來避開三角殺棋旋轉等價的特性，將整體需計算的總量降低至一半。新的八層三角殺棋編碼圖如圖 4-1 所示。

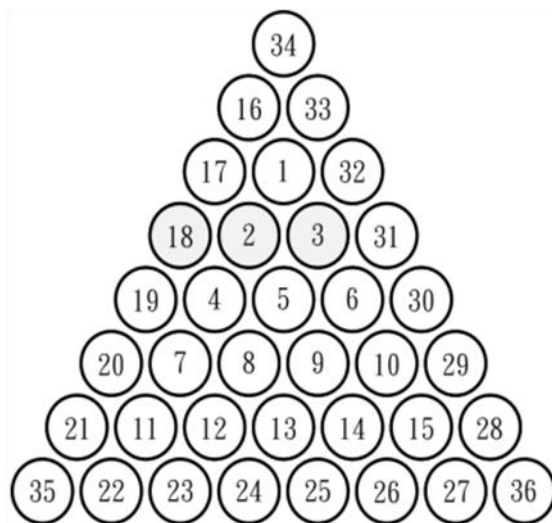


圖 4-1 八層三角殺棋的重新編碼

依照新的編碼表，我們可以將三角殺棋分為三個區段，分別為三個頂點、頂點除外的外圍棋子，以及最內層的五層三角殺棋。各區段的用途皆不同，我們以圖 4-2 來顯示各個區段。

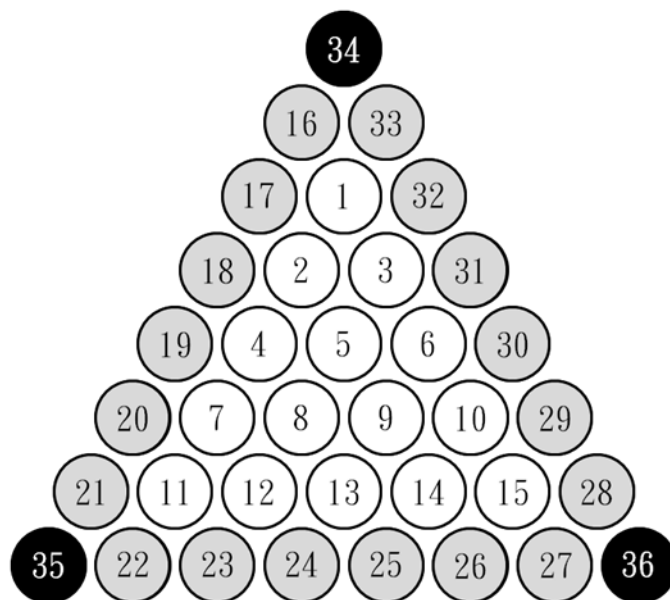


圖 4-2 八層三角殺棋重新編碼之區段圖

其中第一個區段，也就是三個頂點的部份，我們以人工的方式將其預先固定，如表 4-1 所示。這個區段內的狀態剔除旋轉與對稱後，其實以二進位數表示就只有 111、110、100 以及 000 這四組變化。

表 4-1 第一區段之組合與相互關係

狀態編號	第一區段之值	等價之區段值	經合法著手後可能形成之狀態編號	經回溯分析後可能形成之狀態編號
1	111	-	1	1、2、3
2	110	101、011	1、2	2、3、4
3	100	010、001	1、2、3	3、4
4	000	-	2、3、4	4

由表 4-1 可知，第一區段值為 111 的盤面，經過任何可行著手皆無法形成第一區段值為 000 的盤面，因為不存在任何可行著手能夠一次將三個頂點同時選取。而第一區段值為 110 的盤面，經由旋轉等價的概念可以知道，這盤面可經由旋轉的方式形成第一區段值為 101 及 011 的盤面。同樣地，第一區段值為 100 的盤面，可經由旋轉的方式形成第一區段值為 010 及 001 的盤面。根據以上的結果，我們將只需要做四段記憶體配置，作四個陣列來儲存、計算所有最高位元 111、110、100 及 000 的盤面即可。而因固定了三個位元後，需要計算的棋子數僅存 33 子，共 2^{33} 種狀態，所以每段僅需 2^{33} Bits(1G Bytes)的記憶體空間。

在倒推的過程中，我們必定會遇到最高的三個位元狀態不屬於狀態編號 1 至 4 的盤面，所以我們必須將整個盤面都旋轉成相對應的盤面，才能在正確的陣列中找到欲計算盤面的勝負狀態。旋轉的虛擬碼如圖 4-3 所示。

```

N：盤面上之棋子數
Board：盤面之值
transferMap[ N ] = { A | A：位元 A 經旋轉後之位元位置 }

for( i=0; i<N; i++){
    Result = ( Result | ( ( ( Board shift right i ) & 1 ) shift left transferMap[i] ) );
}

Output Result

```

圖 4-3 旋轉盤面之虛擬碼

有鑑於將盤面旋轉需要大量的時間，在盤面上有 33 個子的狀態下，每一次旋轉都需要做 66 次位元的左右 shift 動作。為了減少負擔，我們必須將所有欲計算的盤面之旋轉狀況存起來，然而總數有 2^{33} 種狀況，若每種狀況皆要將其旋轉等價的盤面儲存，總記憶體需求量便為 64G Bytes，如此便本末倒置了。為了處理這個問題，我們特別將整個編碼再分為第二區段以及第三區段，第二區段共有 18 子，第三區段共有 15 子。我們只要建立兩個 Hash Table，分別記錄第二區段

所有狀態旋轉後對應的等價狀態，以及第三區段的所有狀態旋轉後所對應的等價狀態。其所需的記憶體空間如表 4-2 所示。

表 4-2 旋轉等價之 Hash Table 所需空間表

編碼區段	總共子數	旋轉等價 Hash Table 所需記憶體空間
第二區段	18 子	1M Bytes
第三區段	15 子	128K Bytes

當我們的棋盤需要向右旋轉 120 度時，只需將盤面之第二區段值取出，向右移 15 個 bits 轉換成較小的整數，再由對應之 Hash Table 找到旋轉後的值，將此值左移 15 個 Bits，得到旋轉後的第二區段之值。而第三區段的部分，只需將盤面的第三區段擷取出，經由對應的 Hash Table 找到旋轉後的值，最後將對應之新的第一區段、第二區段與第三區段做 OR 運算，便可得到旋轉後的值。若要尋取向右旋轉 240 度的等價盤面，只需將一次旋轉後的第二區段以及第三區段的值再一次參照 Hash Table，即可獲得。

另外由表 4-1 所示，第一區段狀態編號為 4 的盤面，絕對不會參考到狀態編號為 1 的盤面。所以在不考慮回溯分析法的情況下，我們只需要使用總量為 3G Bytes 的陣列即可將八層三角殺棋完全破解。而若考慮回溯分析法，我們依然能於計算第一區段編號為 4 的盤面時，將第一區段編號為 1 的所有狀態釋放，可以大幅度的降低破解八層三角殺棋的記憶體需求量。表 4-3 記錄了我們實作八層三角殺棋於各種作法下的空間及時間需求。

表 4-3 八層三角殺棋實作之空間與時間需求

實作方法	最大記憶體需求	需求時間
倒推法	8G Bytes	15 小時
倒推法搭配回溯分析法	8G Bytes	7.9 小時
重新編碼與倒推法	3G Bytes	10 小時
重新編碼、倒推法 及回溯分析法	4G Bytes	4.4 小時

利用此種做法我們可以於短時間、少量的記憶體限制下，找出八層三角殺棋所有的必勝著手，證明這個做法是有用的。

第三節 於九層三角殺棋上之應用

考慮到九層三角殺棋於取得最後一子為勝的規則下，我們仍然沒有找到必勝著手，所以我們利用這個概念，並考量現有硬體的極限來用以計算九層三角殺棋的可行著手。我們分析了已知的所有必敗著手後，便找出了唯一仍不知其勝負狀態且可以利用此方法求解的可行著手，此著手一刀刪去 6 個子，如圖 4-4 所示。

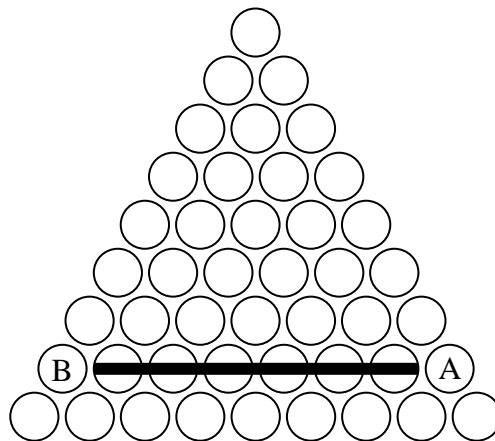


圖 4-4 可利用加速及空間節省概念找出勝負之著手

由於圖 4-4 中，標示為 A 與 B 的兩個棋子必不會由一可行著手同時將兩子取走，所以我們將這兩個棋子的位元設為最高位元，列為第一區段。如此第一區

段的值便只 11、10 與 00 三種狀態，也就是只需要考慮對稱等價的問題。而剩下的棋子只要特別做重新編碼，便可以方便我們找尋對稱等價的盤面即可。重新編碼的盤面如圖 4-5 所示。

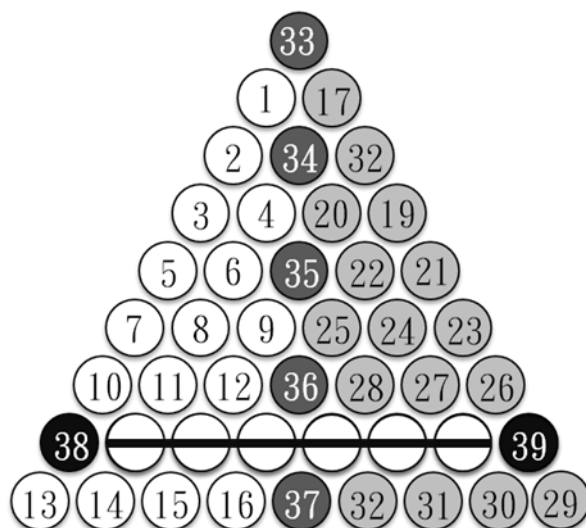
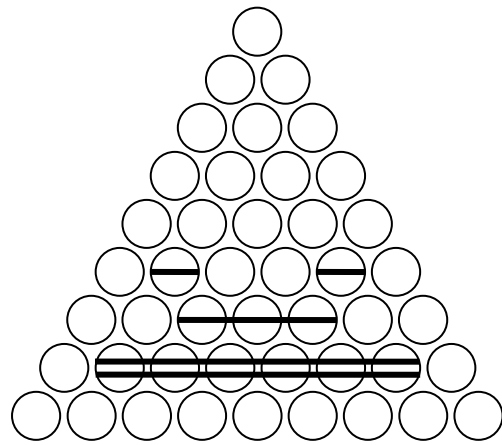


圖 4-5 重新編碼之九層三角殺棋

我們將第 38 與 39 位元設為第一區段，第 33 至 37 位元設為第二區段，第 17 至 32 位元設為第三區段，第 1 至 16 位元設為第四區段。當第一區段值為 01 時，我們可將整個盤面做對稱等價，令第一區段值轉變回 10，第二區段之值保持不變，而將第三區段之值右移 16 個位元，第四區段之值左移 16 個位元，再將所有區段重新組合，便可得到正確的對稱等價盤面。如此我們只需要三段記憶體，每一段記憶體需要存放 1 至 37 子的所有狀態，所以每一段需要 16G Bytes 的記憶體。而第一段記憶體(第一區段值為 11)當倒推至第二段記憶體做完(第一區段值為 10)後，便可以釋放，再繼續做第三段記憶體(第一區段值為 00)，總共需要 32G Bytes 的記憶體空間。我們利用這個做法，費時 535791 秒，證明此一著手仍為必敗著手。也藉由此方法，找出了對應的三手後手方的必勝著手，其中有兩手為我們先前的所有作法中皆沒有找到的，如圖 4-6 所示。至此我們共找出了 40 種九層三角殺棋於取得最後一子為勝規則下的必敗著手。



- ⊖ : 對應預先選取著手之必勝著手
- ⊗ : 必敗之可行著手

圖 4-6 找到之兩手必敗著手

第五章 結論及未來研究方向

第一節 結論

本研究首度證明了九層三角殺棋於「取得最後一子為敗」之規則下之 weakly solve，證明其結果為先手必勝，並將其中一個必勝著手找出。表 5-1 為現有已知的三角殺棋勝負情形。

表 5-1 已知的三角殺棋勝負情形

層數	取得最後一子為敗	取得最後一子為勝
一層	先手必敗	先手必勝
二層	先手必勝	先手必敗
三層	先手必敗	先手必勝
四層	先手必勝	先手必勝
五層	先手必敗	先手必勝
六層	先手必勝	先手必勝
七層	先手必勝	先手必勝
八層	先手必勝	先手必勝
九層	先手必勝	未確定

雖然於取得最後一子為勝的規則下，我們沒有找出必勝的結果，亦無法確認其為必敗，但我們於剔除旋轉對稱等價的著手後的 80 手可行著手中，證明了其中 40 手著手為必敗著手，如附錄 A 所示。而於研究的過程中，我們分析了許多三角殺棋的盤面特性，並加以列舉及應用，雖然沒有獲得預期的成果，但是也分析了許多不同的技術與做法，可以說是獲得了許多經驗，讓之後的研究者可以選擇較有效的方法來進行，而不會重複選擇可能無法獲得答案的方法。另外我們也

將八層三角殺棋的解法再度精進，如此便可以利用比原始的做法還要少約 50% 的時間與空間，希望這些方法能對未來的三角殺棋演算法相關研究有所幫助。

第二節 未來研究方向

我們已知三角殺棋存在許多旋轉等價及對稱等價的盤面，搭配盤面重新編碼後，我們可以用這類特殊的方式去避開無謂的重複運算與記錄。若能夠在保持其旋轉對稱特性不變，且盤面資訊不遺漏的情況，將預先固定分析的棋子數增加，減少必須窮舉的盤面狀態總量，將有可能破解九層以上的高層數三角殺棋，雖然需要分析大量的組合狀況以及消耗大量計算時間，但這仍是未來值得加以研究的方向。

附錄 A 九層三角殺棋已知必敗之著手列表 (取得最後一子為勝，剔除等價盤面)

表 A-1 所示的必敗著手皆對照圖 A-1 的編碼圖，各位元皆對應到盤面上所編寫之號碼。

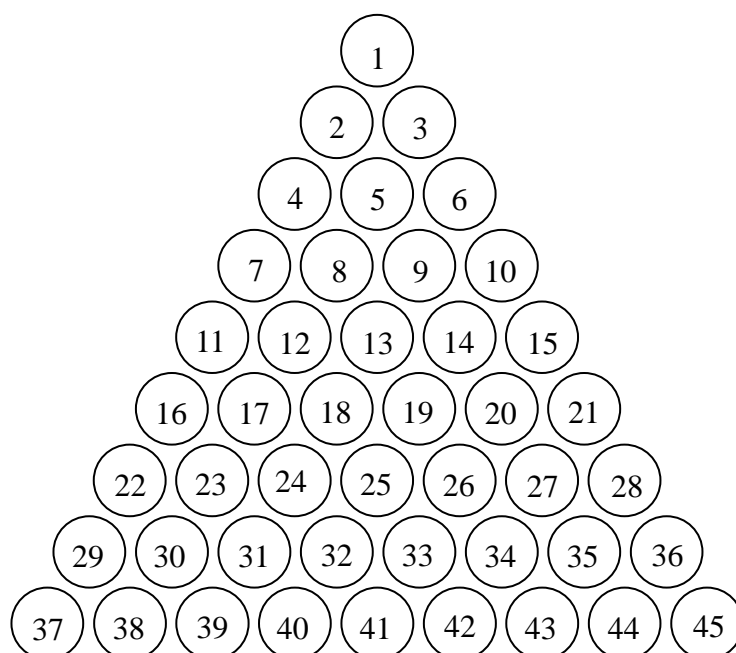
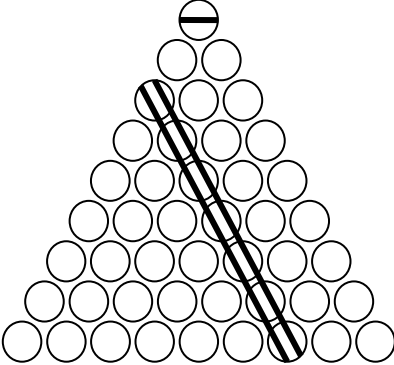
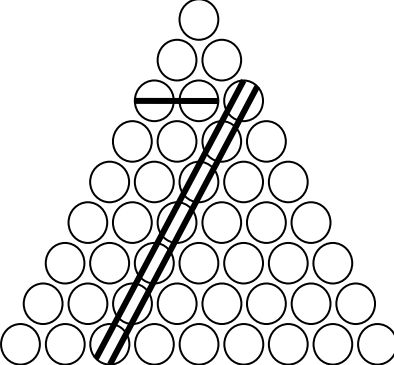
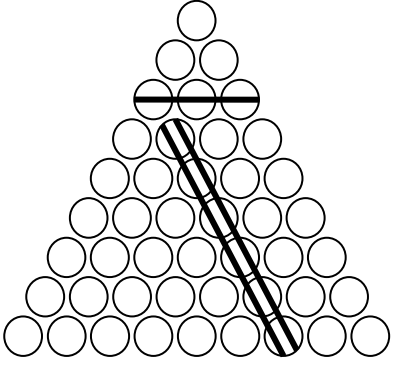
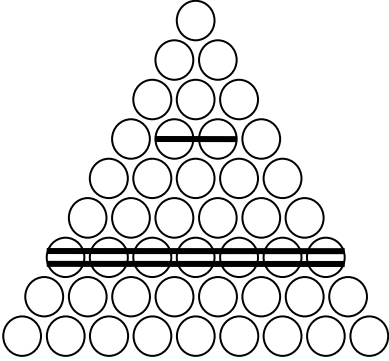
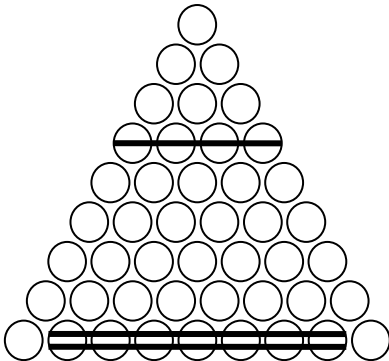
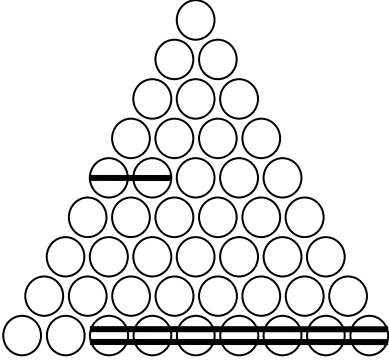
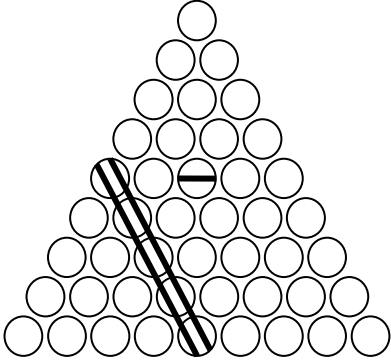
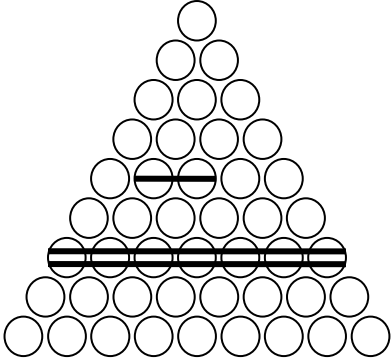
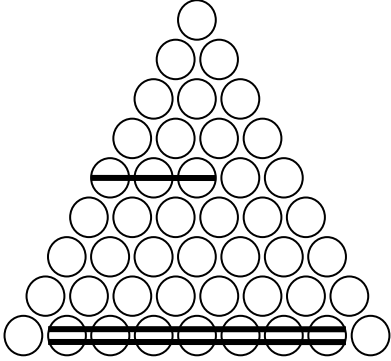


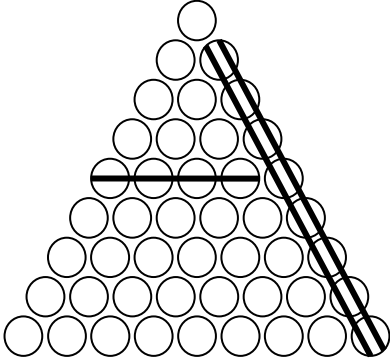
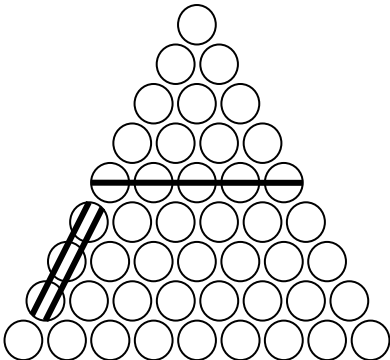
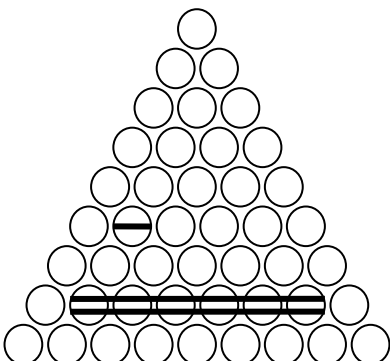
圖 A-1 九層三角殺棋編碼表

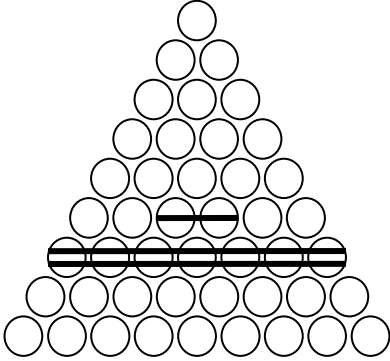
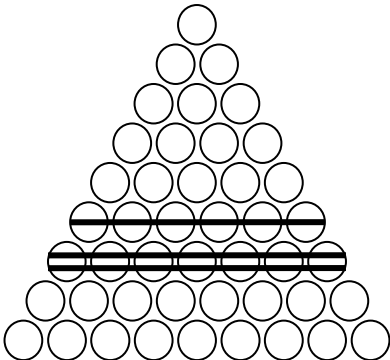
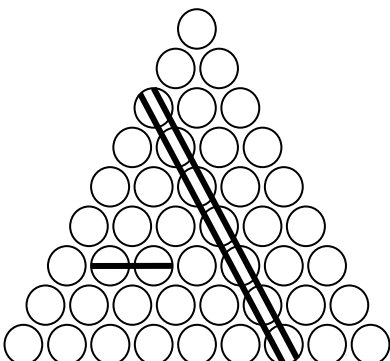
表 A-1 九層角殺棋必敗著手(單線表示先手之敗著，雙線表示後手對應之勝著)

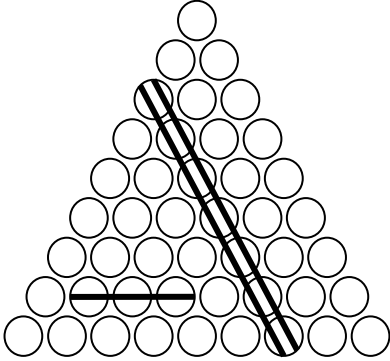
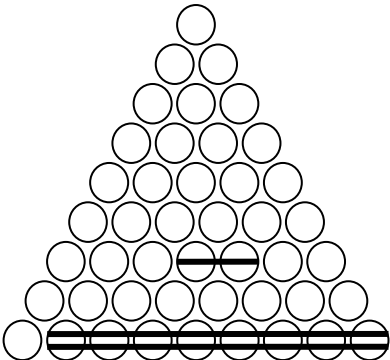
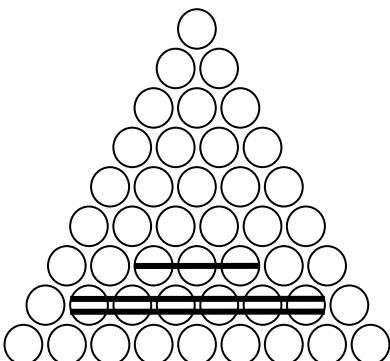
編號	必敗著手圖形	
1		
	二進位表示	十進位表示
	1	1
2		
	二進位表示	十進位表示
	11000	24
3		
	二進位表示	十進位表示
	111000	56

4		
	二進位表示	十進位表示
	110000000	384
5		
	二進位表示	十進位表示
	1111000000	960
6		
	二進位表示	十進位表示
	110000000000	3072

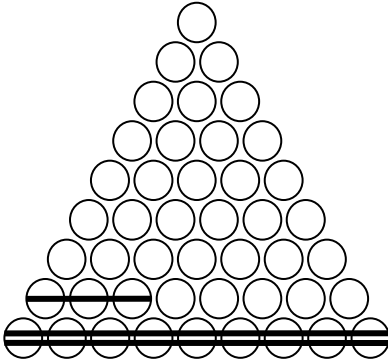
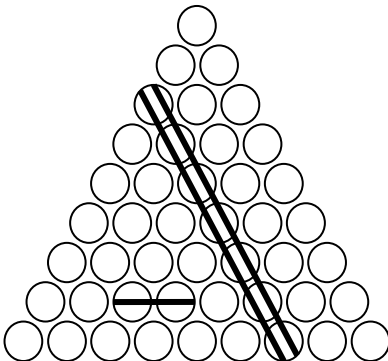
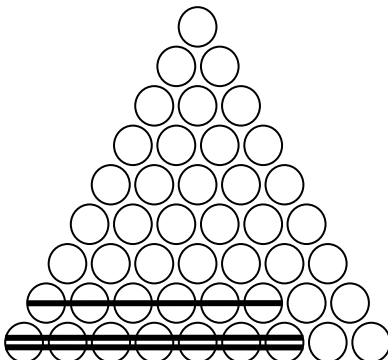
7		
	二進位表示	十進位表示
	1000000000000	4096
8		
	二進位表示	十進位表示
	1100000000000	6144
9		
	二進位表示	十進位表示
	1110000000000	7168

10		
	二進位表示	十進位表示
	11110000000000	15360
11		
	二進位表示	十進位表示
	1111100000000000	31744
12		
	二進位表示	十進位表示
	1000000000000000	65536

13		
	二進位表示	十進位表示
	11000000000000000000	393216
14		
	二進位表示	十進位表示
	1111110000000000000000	2064384
15		
	二進位表示	十進位表示
	110000000000000000000000	12582912

16		
	二進位表示	十進位表示
	111000000000000000000000	29360128
17		
	二進位表示	十進位表示
	110000000000000000000000	50331648
18		
	二進位表示	十進位表示
	111000000000000000000000	58720256

19		
	二進位表示	十進位表示
	11111100000000000000000000000000	132120576
20		
	二進位表示	十進位表示
	11111110000000000000000000000000	266338304
21		
	二進位表示	十進位表示
	11000000000000000000000000000000	1610612736

22		
	二進位表示	十進位表示
	11100000000000000000000000000000	1879048192
23		
	二進位表示	十進位表示
	11000000000000000000000000000000	3221225472
24		
	二進位表示	十進位表示
	11111100000000000000000000000000	16911433728

25		
	二進位表示	十進位表示
	11111100000000000000000000000000	33822867456
26		
	二進位表示	十進位表示
	11111110000000000000000000000000	34091302912
27		
	二進位表示	十進位表示
	11111111000000000000000000000000	68451041280

40		
	二進位表示	十進位表示
	111000000000000000	229376

表 A-1 中，第 40 個必敗著手為根據八層三角殺棋之必勝著手而來。因八層三角殺棋取該著手後，盤面狀態便為必敗，也就是說當九層三角殺棋之先手玩家取此著手時，後手玩家僅需將第九層之子完全取走，即會形成如八層三角殺棋取走該必勝著手的狀態，致使後手玩家獲勝。

參考著作

- [1] Charles L. Bouton, "Nim, A Game with a Complete Mathematical Theory," The Annals of Mathematics, 2nd Ser., Vol. 3, No. 1/4. (1901-1902), pp. 35-39.
- [2] "Wikipedia/Nim," <http://en.wikipedia.org/wiki/Nim>.
- [3] 群想網路科技, 「CYC 遊戲大聯盟」,
<http://cyc9.cycgame.com/cyc/cgi-bin/manual.php?i=manG&game=Nim>。
- [4] 許舜欽, "三角殺棋的電腦解法及其實現", 電腦季刊, 第 16 卷, 第 4 期, pp.15-23, Dec. 1982.
- [5] 許舜欽, "利用電腦探討七層角殺棋的勝負問題", Proc. Of 1985 NCS, pp.798-802, Dec. 1985.
- [6] 白聖群, "八層三角殺棋的勝負問題之研究", 2009 National Computer Symposium (NCS 2009), Workshop on Artificial Intelligence, Fuzzy, and U-Learning (AFU), Taipei, Taiwan, 2009.
- [7] 林宏軒, "八層三角殺棋之解法", 2009 National Computer Symposium (NCS 2009), Workshop on Algorithms and Bioinformatics (AB), Taiwan, 2009.
- [8] Grundy, P. M. (1939). "Mathematics and games". Eureka 2: 6–8. Reprinted, 1964, 27: 9–11.
- [9] "Wikipedia/Sprague-Grundy Theorem,"
http://en.wikipedia.org/wiki/Sprague%E2%80%93Grundy_theorem.
- [10] "Wikipedia/Nim," <http://en.wikipedia.org/wiki/Nim>.
- [11] S. Russell, P. Norving, Artificial Intelligence: A Modern Approach, 2/E, PEARSON, 2003.
- [12] "Wikipedia/ 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。