# 八層三角殺棋的勝負問題之研究

# On the Study of 8 Layer Triangular Nim

白聖群
國立台灣師範大學資工系
Email: g96470667@csie.ntnu.edu.tw

林順喜
國立台灣師範大學資工系
Email: linss@csie.ntnu.edu.tw

***摘要*—**電腦棋類遊戲在人工智慧領域中是很重要的一個課題。而三角殺棋部份，自從許舜欽教授在 1985 年全國計算機會議（NCS 1985）發表七層三角殺棋結果後。三角殺棋更多層數的結果，就尚未有人得出解答。

在本論文中，我們除了找到八層三角殺棋的結果，也提出了一些解三角殺棋勝負時，可以加快搜尋勝負速度的方法。雖然研究過程中花費許多時間在倒推法上，但我們也研究出來所有先前求出的盤面是可以運用到之後要求解的三角殺棋，並且提出了一個管理記憶體的方式，使得在求解三角殺棋的過程中，盤面資訊狀態可以儲存，這樣就可以利用較少量記憶體來解八層三角殺棋，而不必動用到虛擬記憶體。我們使用 CPU 為 AMD Athlon64 X2 4000+ 2.1GHz，記憶體為 8G Byte 的個人電腦，花費約十四個小時半的時間，證明了兩種規則的八層三角殺棋皆為先手勝。

***關鍵詞*—人工智慧、三角殺棋**

***ABSTRACT*—**Computer chess games are very important in the field of artificial intelligence. There is no research results on Triangular Nim in higher dimensions Since Professor Shun-Chin Hsu solved 7 layer triangular Nim in NCS 1985.

In this paper, we find some skills for improving the performance of our programs. We can save the sub-problem results and reuse these results for solving 8 layer Triangular Nim. A memory management scheme is also proposed to reduce the memory requirements which can avoid the virtual memory swapping. A personal computer equipped with AMD Athlon64 X2 4000+ 2.1GHz CPU and 8 GBytes RAM is utilized to conduct our experiments. Thus, it spends 14.5 hours and gets the results that two kinds of 8 layer Triangular Nim are a first-player win.

***Keywords* —** artificial intelligence, Triangular Nim

## 第一章　諸論

三角殺棋相傳最古早的淵源來自於中國，當時的人們利用小石頭的堆砌，進而有規律的發展出這麼一套有趣遊戲。輾轉流傳到現代，其遊戲方式算是 Nim 遊戲[1][2]的一種變形。Alan Tucker 在「Applied Combinatorics」[3]一書也曾經提到三層和四層的三角殺棋。在坊間，三角殺棋亦有一些遊戲網站[4]提供線上服務供人玩此遊戲。

三角殺棋是兩人進行的益智遊戲，一開始的遊戲棋盤是排成正三角形，棋盤高度即為此三角殺棋之層數。如圖 1 的盤面即為八層三角殺棋。

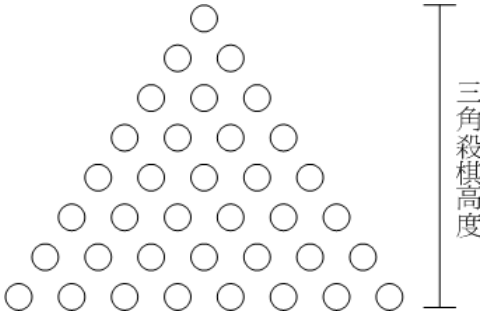


圖 1 八層三角殺棋

遊戲規則主要是有兩種，一種是從棋盤取得最後一顆子獲勝，另一種是從棋盤取得最後一顆子為失敗。每次玩家選取的棋子，在取子的時候不限定遊戲者取子數目，但必須相連（不可斷掉）且成同方向的一直線，而且必須至少取一子。

決定好玩家取子先後次序，雙方再不斷地輪流取子，總有一個時刻棋子會被取光而決定勝負。而此遊戲只有贏或輸，並沒有和局這種狀況。

一般來說在三角殺棋的遊戲中，人類與電腦程式對弈是不可能會獲勝的，這是因為電腦能比人類往前看更多的三角殺棋走法變化，從一開始下棋，便先設想到許多步棋以後的棋局變化，在第一步棋就能選擇最有優勢的候選步，規劃出無懈可擊的策略。然而以八層三角殺棋來說，可能的盤面組合實在是太多了，所有的盤面組合的數目是 $2^{36}$，計算複雜度也相對提高。

# 第二章　相關文獻及基礎理論

## 第一節　相關研究成果

在許舜欽教授的論文中[5][6]，因為該遊戲難以用傳統的分析方式解決，故直接使用倒推法來解該問題。也就是說，並不使用估值函數及 MIN-MAX 搜尋，或配上 $\alpha$-$\beta$ 切捨。倒推法主要利用了電腦快速運算及大量記憶的特性，將遊戲的所有可能狀況全部計算出來，找到最好的應手。

而許舜欽教授利用倒推法解開了一到七層三角殺棋的勝負問題，其三角殺棋規則為取到最後一子為敗。勝負情形如表 1。

表 1 三角殺棋勝負情形(取得最後一子為敗)

| 三角殺棋層數 | 勝負情形 |
|---|---|
| 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為先手勝的結果。

## 第二節　倒推法

以八層三角殺棋來說，倒推法就必須利用 $2^{36}$ 個位元(8G bytes)來記錄每一盤面狀態的勝負情況。若某狀態走一步後的下一個狀態均是先手必勝，則此狀態為先手必敗。因為以此狀態來說，不論接下來考慮的著手為何，由此狀態到下一狀態皆是對方會獲勝，皆找不到己方的致勝路徑。所以此狀態會被設為先手敗。反之，只要下一個狀態有一個是先手敗，則可以推導出目前狀態為先手勝。

由於三角殺棋的盤面狀態只有「未劃過」或「劃過」兩種狀態，只要一個 bit 即可明確表示。因此，解八層三角殺棋，我們可以利用 36 個位元來表示 $2^{36}$ 個種盤面狀態。而八層三角殺棋所有狀態可對應到整數 0 ~ $2^{36}$-1。

首先是將棋子排成如圖 2 的三角形，並且將八層三角殺棋的棋子編號如圖 2。

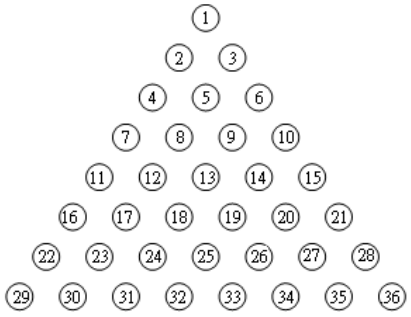


圖 2 三角殺棋編碼盤面

編碼完的三角殺棋對應到 36 bit 整數的方式就如圖 3，每一個 bit 為 0 或 1。

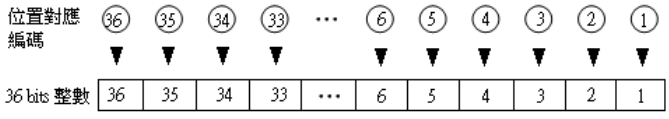


圖 3 位置編碼示意圖

編號 $2^{36}$-1 代表棋盤上全部棋子皆被取走，然後由此狀態的勝負倒推到編號 0，也就是倒推到棋盤上所有棋子皆未被取走的起始狀態，便可以決定三角殺棋的勝負。這個演算法最早是許舜欽教授提出[7]，如圖 4。

```
S(2^36-1)= 必勝 ;

for(i=2^36-2;i>=0;i--)
{
   S(i) 爲必敗 ;
   for(j=1;j<=288;j++)
   {
      if( 第j著手爲合法 )
      {
         求出S(i) 下第j著手之後的盤面狀態 ;
         if( 下完後的盤面狀態爲必敗 )
         {
            S(i) 爲必勝 ;
            break;
         }
      }
   }
}
```

圖 4 倒推法演算法

圖 4 是取走最後一個棋子為負的倒推法。若要將規則改為取走最後一個棋子為勝，我們只需將一開始時 $S(2^{36}-1)$的狀態設為必敗即可。因為根據演算法的設計原則，目前狀態是由以前狀態推導出來。所以當 $S(2^{36}-1)$設為必敗時，也就是所有棋子被取走的前一個狀態皆為勝，也就符合了取走最後一個棋子為勝的規則。

### 第三節　可行著手集合的產生

在使用倒推法之前，必須先建立八層三角殺棋的可行著手集合。可行著手集合的建立，也是延續許舜欽教授在「利用電腦研究七層三角殺棋的勝負問題」中所提到的建立方法。

可行著手集合的建立方法，是根據我們在倒推法一開始建立好的編碼盤面，如圖 2。然後根據資料結構的狀態產生對應編碼，如圖 5，一個可行著手取走了棋盤上的 7、8、9、10 四顆棋子，再根據圖 3，我們可以知道 bit7、bit8、bit9、bit10 為 1，其他 bit 皆為 0，這樣的著手編碼後為 00000000000000000000000001111000000 的二進位值，轉成整數值為 960。而利用這樣的編碼方式可將所有可行著手的值給尋找出來。

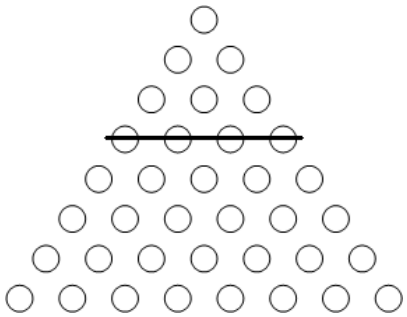

圖 5 可行著手一例

然後我們建立一個棋盤上每個棋子之間的位置關係圖，針對棋子的右邊、左下、右下建立一個連結表格。因為三角殺棋有這三個方向的可行著手，我們建立這個表格的目的，就是為了檢查各方向是否還有相鄰的棋子。建立的連結表格如表 2。其中 0 表示該方向並沒有相鄰的棋子。

有了位置連結表格，我們就可以根據圖 6 的演算法(也是由許舜欽教授提出)，依序由每一個點開始，產生經過該點的右邊、左下及右下方向的所有可行著手的值求出來，以便和三角殺棋的倒推法合併使用，而演算法中的 Link 即為位置連結表格。

## 第三章　如何解八層三角殺棋？

### 第一節　Divide-and-Conquer

在本研究過程中所使用的演算法與資料結構，是延續許舜欽教授在「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[6]。在初期的研究中，是預期使用 Divide-and-Conquer 的概念下去尋找三角殺棋勝負結果，將大問題切割成一些小問題，先去計算小問題的解，並儲存結果以供稍後的計算使

用，然後建構成整個問題的解。在這裡我們是切割為兩個子問題。

表 2 八層三角殺棋位置連結表格

| 位置 | 右邊 | 左下 | 右下 |
|---|---|---|---|
| 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 | 29 | 29 | 30 |
| 23 | 24 | 30 | 31 |
| 24 | 25 | 31 | 32 |
| 25 | 26 | 32 | 33 |
| 26 | 27 | 33 | 34 |
| 27 | 28 | 34 | 35 |
| 28 | 0 | 35 | 36 |
| 29 | 30 | 0 | 0 |
| 30 | 31 | 0 | 0 |
| 31 | 32 | 0 | 0 |
| 32 | 33 | 0 | 0 |
| 33 | 34 | 0 | 0 |
| 34 | 35 | 0 | 0 |
| 35 | 36 | 0 | 0 |
| 36 | 0 | 0 | 0 |

```
n 值爲可行著手的值，Link 爲連結表

for(i=1;i<=36;i++)
{
    n=2^(i-1);
    輸出 n 值 ;
    for( 方向 = 右邊，左下，右下 )
    {
        n=2^(i-1);
        j=i;
        for(k=1;k<=7;k++)
        {
            if(Link(j, 方向) 等於 0)
            {
                此方向結束
            }
            else
            {
                j=Link(j, 方向);
                n=n+2^(j-1);
                輸出 n 值 ;
            }
        }
    }
}
```

圖 6 輸出所有可行著手的演算法

為了尋找八層三角殺棋勝負的方便性，我們首先考慮將遊戲規則變更為取到最後一個棋子為勝，也就是三角殺棋在前言提到的另一個規則。這點與許舜欽教授的論文有些許不同，該論文中是以取到最後一個棋子為負。所以，一層到七層的結論必須重新尋找，但有鑒於搜尋空間遠小於八層，此變更方式仍然是值得。而且也把另外一種遊戲規則的三角殺棋結果給找出來，對學術上研究也是很有幫助的。

在這裡之所以要變更遊戲規則的原因是，為了要運用 Divide-and-Conquer 的概念，故假設一開始第一個玩家就將整個三角殺棋盤面切割

為兩個小區塊。如圖 7，一個小區塊是三角形，另一個小區塊是梯形。如此一來，只要能找到三角殺棋上面三角形盤面為先手必敗，下面梯型為先手必敗，則可證明出取到最後一子為勝的八層三角殺棋為先手必勝。

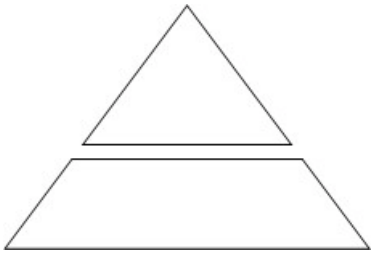

圖 7 Divide-and-Conquer 示意圖

當然會有這樣的推測，主要是因為許舜欽教授在取到最後一子為負的三角殺棋中，一、三、五層的結論是為先手必敗。而第六、第七層卻是先手必勝，這樣似乎可以歸納出越多層，對先手越有利。

若是不變更遊戲規則的情況下，也就是取得最後一子為敗，我們下完一個著手將三角殺棋分割為兩個子盤面時，在兩個三角殺棋子盤面中，我們對於兩個子盤面，皆是後手的狀態。在這樣的情況下，會產生三個 Case。

Case 1:若切割完的兩個子盤面皆為後手勝。

Case 2：若切割完的子盤面皆為先手勝的情況。

Case 3：若切割完的子盤面一個為先手勝、一個為後手勝。

因為在下棋過程中，雙方皆可自由地在兩個子盤面進行著手，所以無法有效保證各子盤面的勝負狀況。而且又因為每次取子的策略不固定，在這樣的規則之下，雙方有時會在盤面搶最後一顆子，有時又會避免取到最後一顆子。這樣增加了分析上的困難。

在變更遊戲規則的情況下(也就是取得最後一子為勝)，我們下完一個著手將三角殺棋分割為兩個子盤面時，在兩個三角殺棋子盤面中，我們對於兩個子盤面，皆是後手的狀態。在這樣的情況下，也同樣會有三個 case。

Case 1:若切割完的兩個子盤面皆為後手勝。

Case 2:若切割完的兩個子盤面皆為先手勝。

Case 3：若切割完的子盤面一個為先手勝、一個為後手勝。

在這三個 Case 的情況下，我們可以如下證明 Case 1 必為後手勝，Case 3 必為先手勝，而 Case 2 則無法那麼直觀地去分析。

Case 1：若切割完的兩個子盤面皆為後手勝。在這種情況之下若玩家一為先手，玩家二為後手，這邊假設兩個後手勝的子盤面為盤面一、盤面二。

玩家一先到盤面一去下一著手，所以在盤面一為先手。此時玩家二最佳著手會是在盤面一與玩家一纏鬥。若玩家一與玩家二纏鬥至最後，則可以推導出玩家二必取得盤面一最後一顆子。該盤面為後手勝，所以玩家二必勝。於是到盤面二時，玩家二狀態會是後手。因為取完盤面一最後一顆子後，玩家一必然要在盤面二下任意著手。此刻玩家一為先手，推得該盤面必敗，因為盤面二後手勝。

若玩家一不與玩家二纏鬥，盤面一尚未結束就跳到盤面二，此時玩家一為先手，玩家二最佳著手為跟著玩家一到盤面二與玩家一纏鬥，若纏鬥至最後，則必然可以取得盤面二最後一顆子。所以，在這個規則之下遇到此兩種盤面組合，玩家二最佳下法就是只要玩家一跳離盤面，也跟著跳離該盤面，無論如何要與之纏鬥。則可以保證一定會取得兩盤面的最後一顆子，也就是取得這樣盤面組合的勝利。

Case 3：若切割完的子盤面一個為先手勝、一個為後手勝。在這種情況之下若玩家一為先手，玩家二為後手，假設先手勝的子盤面為盤面一、後手勝的子盤面為盤面二。我們已經知道，兩個子盤面若後手勝，這樣可以推倒出整體盤面為後手勝。 所以，只要玩家一下一子到盤面一，使盤面一為後手勝盤面，則對玩家一來說整體盤

面為後手，則必可取得整體盤面的勝利。也就證明了切割完的子盤面，若一個為先手勝、一個為後手勝必為先手勝。

藉由這樣的方式推導我們也可以證明，若一個先手勝盤面搭配多個先手敗盤面必為先手勝。

但是我們將一到七層的勝負結果，使用程式分析完成之後，發現結果並不如一開始猜想那樣，可以將分割結果拿來使用的並不多。而三角殺棋的勝負如表 3，根據三角殺棋一到七層的勝負表，僅有二層三角殺棋的先手敗結果是可以使用的。

表 3 三角殺棋勝負情形(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 |

根據二層三角殺棋的結果，我們對八層三角殺棋中剩餘的梯型盤面重新編碼，並且產生新的可行著手數，而編碼的方式如圖 8。以此編碼方式所需的空間為 $2^{30}$bytes = 1G bytes，產生的可行著手數共有 210 種。在三角殺棋編碼的過程中，最上面的三層是不編碼的。

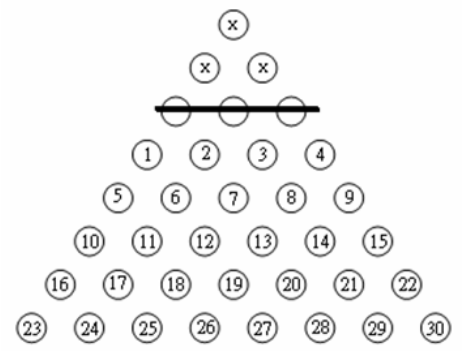


圖 8 三角殺棋梯形盤面編碼

但很可惜的是以這種方式來求解，並沒有達到預期目標。梯形方塊的勝負結果為先手勝，如此一來這樣的結果與二層的先手負結果相搭配，屬於 Case 3，可得知圖 8 的盤面是先手勝，因此前一步取走第 3 層的三顆子是敗著。如此一來，無法推論出任何結果，只能說這個方法失效。

## 第二節　搜尋必勝著手

雖然上一節將三角殺棋盤面以分割的方式來求勝負是失敗的，但在求解的過程中也引發了新的契機。在求三角殺棋勝負解的時候皆有把各盤面資訊記錄下來，因此想藉由觀察三角殺棋三到七層的第一手下在盤面中的何處才會獲勝，進而推導出八層三角殺棋盤面中，第一手最有可能會落在何處。

三層三角殺棋（規則為下到最後一子為勝）搜尋必勝著手結果如圖 9（每個線段皆為一個著手），第一步必勝著手有六步。

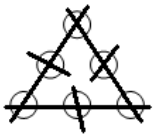

圖 9 三層三角殺棋第一步必勝著手

四層三角殺棋（規則為下到最後一子為勝）搜尋必勝著手結果如圖 10，第一步必勝著手有六步。

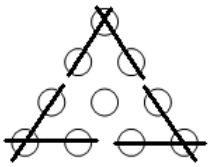

圖 10 四層三角殺棋第一步必勝著手

五層三角殺棋（規則為下到最後一子為勝）搜尋必勝著手結果如圖 11，第一步必勝著手有六步。六層三角殺棋（規則為下到最後一子為勝）搜尋必勝著手結果如圖 12，第一步必勝著手有三步。七層三角殺棋（規則為下到最後一子為勝）搜尋必勝著手結果如圖 13，第一步必勝著手有十步。

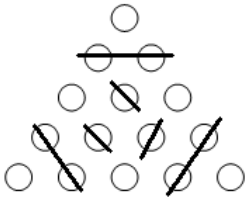

圖 11 五層三角殺棋第一步必勝著手

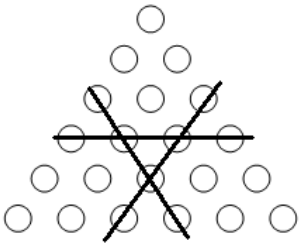

圖 12 六層三角殺棋第一步必勝著手

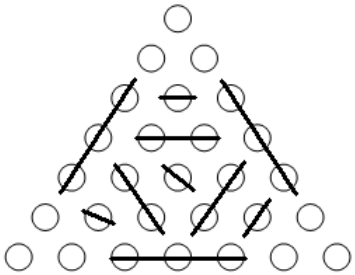

圖 13 七層三角殺棋第一步必勝著手

在規則為下到最後一子為敗的情況下，如表 1，先手必勝的盤面只有二、四、六及七層，二層的三角殺棋因為較簡單在此暫不討論，這個規則之下的盤面我們僅討論四、六及七層的必勝著手。

四層三角殺棋（規則為下到最後一子為敗）搜尋必勝著手結果如圖 14，第一步必勝著手有三步。

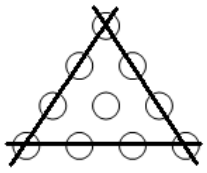

圖 14 四層三角殺棋第一步必勝著手

六層三角殺棋（規則為下到最後一子為敗）搜尋必勝著手結果如圖 15，第一步必勝著手有十二步。

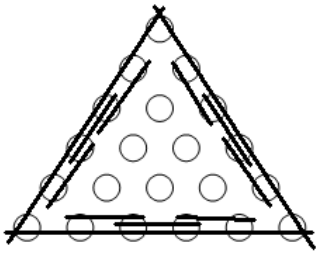

圖 15 六層三角殺棋第一步必勝著手

七層三角殺棋（規則為下到最後一子為敗）搜尋必勝著手結果如圖 16，第一步必勝著手有六步。

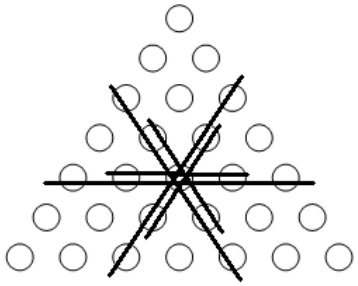

圖 16 七層三角殺棋第一步必勝著手

觀察了各層三角殺棋兩個規則下的第一手，我們首先考慮第一手著手有可能是將三角殺棋切割為兩個盤面的一刀。在規則為下到最後一子為勝的情況下，五層三角殺棋與六層三角殺棋皆有一著手是將三角殺棋切割為兩個盤面。在規則為下到最後一子為敗的情況下，七層三角殺棋有一著手將三角殺棋切割為兩個盤面。有了這樣的考慮為前提，我們又重新設計了八層三角殺棋的編碼，並且對八層三角殺棋進行七種測試，希望在這七種測試之下能夠找到必勝的第一著手。而這七種八層三角殺棋編碼表如圖 17。

八層三角殺棋重新編碼後，每種切割法的可行著手數與所執行時所需要的記憶體如表 4。重新編碼後就利用許舜欽教授的倒推法求解，但由於進行研究時，所使用的個人電腦僅配有 8G bytes 的記憶體，故在第五種以上的八層三角殺棋必須進行盤面狀態的壓縮，由原先的 1 byte 存一個盤面資訊改為 1 bit 存一個資訊，如此一

來即可將這七種測試完成。由於編譯器並未提供 1 bit 的資料型態，所以在下一節，我們會介紹壓縮資料型態的演算法。

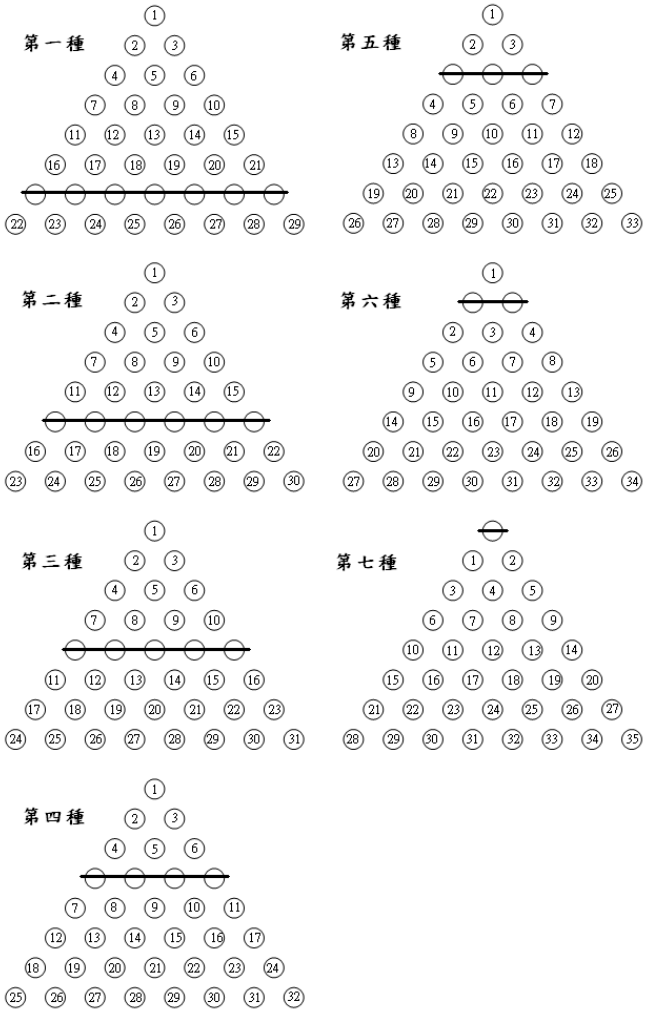


圖 17 七種八層三角殺棋編碼表

利用這個方法我們已經順利找到在規則為下到最後一子為敗的三角殺棋勝負結果。該規則下的八層三角殺棋如同預期般是先手勝，該勝負資訊是在第二種測試方法下找到，故可以知道該規則下的八層三角殺棋至少有一個必勝著手，如圖 18。

**第三節　壓縮盤面狀態資料結構**

對於三角殺棋的盤面狀態的資料結構，每一個狀態的勝負其實僅需 1 bit 就可以儲存，不需使用要到 1 byte 來儲存。但是以 C 語言來說最小的資料型態為 char，而該資料型態佔用的空間為 1 byte。根據這個簡單的想法，我們即對資料結構做編碼壓縮，使之可以用 1 byte 的空間儲存八個盤面狀態的資訊。

表 4 三角殺棋可行著手數及所需記憶體

| | 可行著手數 | 所需要的記憶體 |
|---|---|---|
| 第一種 | 163 種 | 約 $2^{29}$ bytes (512M bytes) |
| 第二種 | 153 種 | 約 $2^{30}$ bytes (1G bytes) |
| 第三種 | 154 種 | 約 $2^{31}$ bytes (2G bytes) |
| 第四種 | 186 種 | 約 $2^{32}$ bytes (4G bytes) |
| 第五種 | 216 種 | 約 $2^{33}$ bytes (8G bytes) |
| 第六種 | 247 種 | 約 $2^{34}$ bytes (16G bytes) |
| 第七種 | 273 種 | 約 $2^{35}$ bytes (32G bytes) |

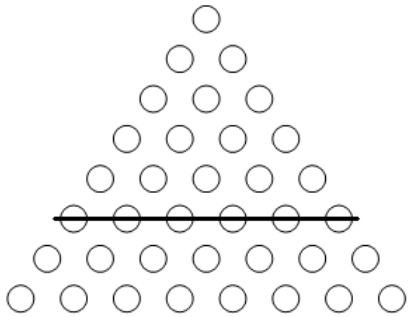

圖 18 規則為下到最後一子為敗，八層三角殺棋必勝的第一步

設定盤面狀態陣列資訊的方式，是用兩個匹配陣列達成，如圖 19。匹配陣列之所以會設計成這個樣子，目的是為了要和原本的盤面狀態資料陣列做 and、or 運算。因為我們知道在程式語言中，做硬體的 and、or 邏輯運算的速度是很快的，而這樣的設計將可以使八層三角殺棋求勝負解的程式多付出的計算時間降低。

```
unsigned char Compare1[8] =
{  {1},  {2},  {4},  {8}, {16}, {32}, {64},{128}};
//將資料陣列設為 1

unsigned char Compare0[8] =
{{254},{253},{251},{247},{239},{223},{191},{127}};
//將資料陣列設為 0
```

圖 19 壓縮盤面資料的匹配陣列

原始盤面狀態陣列中每個元素原本是用 char 來宣告，該陣列元素所佔大小是 1 byte，也就是 8 bit，可表示的範圍為-128~127。為了盤面狀態的壓縮，我們將其宣告為 unsigned char，可表示的範圍為 0~255，這樣的改變使得在程式設計上比較方便。

若我們要將盤面狀態陣列的一個元素設為 1，也就是將該盤面狀態設為先手勝，我們只要將該盤面狀態陣列元素與 Compare1 匹配陣列中對應的元素做 or 運算。若我們要將盤面狀態陣列的一個元素設為 0，也就是將該盤面狀態設為先手敗，我們只要將該盤面狀態陣列元素與 Compare0 匹配陣列中對應的元素做 and 運算。

我們將狀態盤面設為 1 之所以要用 or 運算，主要是因為這樣不會影響到其他狀態的值。因為其他盤面狀態的值與 0 做 or 運算，其盤面狀態不會改變，只有與 1 做 or 運算會改變。將狀態盤面設為 0 用 and 運算也是同樣的道理。其它盤面資料與 1 做 and 運算，其結果將不會改變原本的值，盤面資料原本不論是 0 或 1，對其與 1 做 and 運算後，都還是原本結果。只有與 0 做 and 運算才會改變結果。

要尋找對應的 Compare0 匹配陣列元素或 Compare0 匹配陣列元素時，我們是使用簡單的求餘數計算。舉例來說，若以八層三角殺棋以倒推法從最後一個盤面往前倒推時，遇到的第一個盤面狀態為 $2^{36}-2$，也就是 68719476734，但因為現在資料壓縮的關係其真正儲存的盤面狀態陣列的索引值必須先除以 8，以便取得真正的盤面狀態陣列的索引值。而儲存在盤面狀態陣列裡的位置是由除以 8 之後的餘數來決定，其餘數的值會介於 0~7 之間，我們就利用餘數的值來尋找對應的匹配陣列，這麼一來就可以達到資料壓縮的目的。

## 第四節　改進倒推法的記憶體管理

在第三節中，我們觀察三角殺棋三到七層的必勝著手，進而推導到八層三角殺棋盤面中第一手可能會下在何處的這個方法，很可惜僅有找到三角殺棋規則為取得最後一子為敗的結果。所以三角殺棋規則取得最後一子為勝就只能繼續沿用許舜欽教授的倒推法，利用該方法窮舉所有可能盤面進而求出八層三角殺棋的結果。

我們程式的開發是在 CPU 為 AMD Athlon64 X2 4000+ 2.1GHz、記憶體為 8G Bytes、作業系統為 Microsoft Vista 64 位元的版本、編譯器為 Microsoft Visual Studio 2007 的一台個人電腦上。在該環境之下程式雖然能充份使用 8G Bytes 的記憶體，但卻有著每個陣列最多宣告 2G Bytes 的限制。因為編譯器有這樣的限制，故我們將 8G Bytes 的儲存空間切割為四個 2G Bytes 空間。若假設一開始是在 Unix 環境之下做研究，則就沒有每個陣列 2G Bytes 的限制，在撰寫程式上會比較方便，但可能就不會發現到這樣分割的方式。

若要將八層三角殺棋的每個盤面資訊皆儲存下來，每個盤面資訊就算只儲存 1 bit，也必須耗費 8G Bytes 的記憶體。再加上作業系統本身所耗費的記憶體，則必會動用到虛擬記憶體，而虛擬記憶體相對於實體記憶體的速度是極慢的。若我們一次向系統要求 8G Bytes，則勢必會影響到八層三角殺棋求解的時間。

為了降低這個問題所造成的影響，我們使用一個記體體管理策略，程式有需要該記憶體時才配置給它。前面我們已經將八層三角殺棋所需的 8G Bytes 盤面資訊分割成四段 2G Bytes 的記憶體，如圖 20。我們即可以在程式求解的過程中依程式的需求，根據記憶體的需求再向系統要求記

憶體。如此一來在八層三角殺棋求勝負解程式執行的過程中，該程式的記憶體需求會隨著程式一步步求解的需要，依序向系統要求 2G Bytes，共四次。

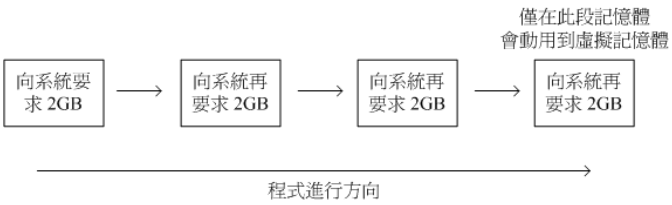


圖 20 記憶體要求示意圖

對三角殺棋盤面資訊記憶體分割的主要概念雖然簡單，但是在設計程式的層面則必須要小心與謹慎，每個區塊間如何與其他區塊的記憶體作存取，將是設計三角殺棋求解程式中的主要關鍵。由圖 21，我們可以清楚的知道每段記憶體向其他記憶體區塊要求的情況。

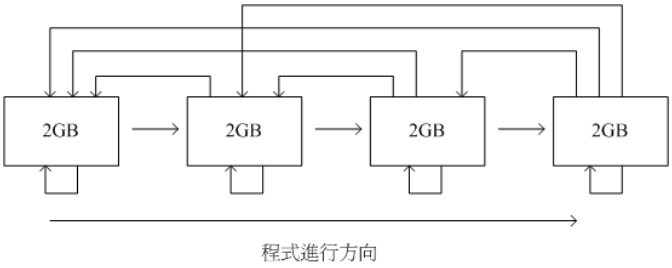


圖 21 記憶體區塊關係

記憶體區塊之所有會有這樣的需求關係，主要是因為觀察倒推法中存取盤面狀態資料的程式碼，會影響到存取盤面狀態的程式碼主要是目前盤面與可行著手做 or 運算。所以在設計程式時，我們可以根據每個盤面狀態與可行著手的 or 運算之後的值，判別計算後的值是屬於哪一個區段的記憶體，再來存取該段記憶體的值。

原本的演算法如圖 4，要更改的地方為求出 S(i)下第 j 著手之後的盤面狀態這個地方，必須增加一個處理步驟，就是判斷該盤面狀態是位於哪一個記憶體區塊，判別完畢之後再將該記憶體區塊減去對應的 offset，每個區塊要減去的 offset 如表 5。

表 5 記憶體區塊 offset 值

| 記憶體 | 區塊 1 | 區塊 2 | 區塊 3 | 區塊 4 |
|---|---|---|---|---|
| offset | 6442450944 | 4294967296 | 2147483648 | 0 |

所有盤面狀態共有 $2^{36}$-1 個，也就是 68719476735，盤面狀態分割成四個記憶體區塊之後，第一個記憶體區塊儲存 51539607552~68719476735 的狀態，第二個記憶體區塊儲存 34359738368~51539607551 的狀態，第三個記憶體區塊儲存 17179869184~34359738367 的狀態，第四個記憶體區塊儲存 0~17179869183 的狀態。因為我們有將資料盤面壓縮，所以第一個記憶體區塊減去的 offset 要從 51539607552 壓縮八倍，也就是除以八即為 6442450944，同理，第二個記憶體區塊的 offset 為 4294967296，第三個記憶體區塊的 offset 為 2147483648，第四個記憶體區塊的 offset 為 0，使這些記憶體區塊都能找到對應的盤面狀態陣列。

## 第五節　倒推法的修改

另外一個改進的方式就是，根據在前面一節中，我們已經推論出八層三角殺棋極有可能為先手勝。也因為在第四次要求 2G bytes 的記憶體時才會動用到虛擬記憶體，故我們將程式的結束條件變更為「若找到第一手可行著手會造成三角殺棋先手勝」。如此一來，我們就可以再替八層三角殺棋求解程式減少一些時間。

而要如何判斷何者為第一手可行著手？在程式撰寫上也不會太困難。若一個盤面資訊恰等於可行著手，即代表此盤面若下該可行著手為第一手，則此盤面為先手勝或先手負。而此盤面資訊若為負，則表示八層三角殺棋為先手勝，但若為勝則沒有任何結論。倘若八層三角殺棋求解程式未找到任何第一手的可行著手，則代表八層三角殺棋為先手負。

經由改良後的演算法我們順利找到了八層

三角殺棋的結果，其必勝的第一步可行著手的結果如圖 22。

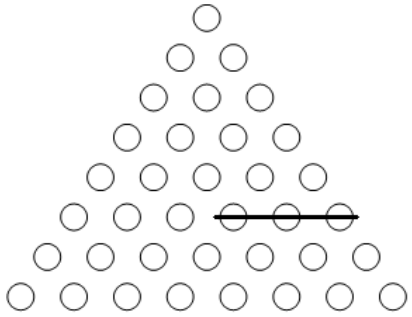

圖 22 規則為下到最後一子為勝，八層三角殺棋必勝的第一步

我們的研究過程到此，已經證明了八層三角殺棋在這兩個規則之下皆為先手勝，是一個對先手有利的遊戲。而我們使用了這個方法共花費了 52431 秒，也就是 14.56 個小時。

## 第四章　尋找所有必勝可行著手

我們證明了八層三角殺棋，不論是哪一種規則皆會是先手勝，也就是八層三角殺棋是個對先手有利的遊戲。由於第三章第五節搜尋第一個必勝走步的方法，花費的時間為 52431 秒，也就是 14.56 小時，還在我們容許的範圍。為了八層三角殺棋的完整性，我們也將兩個規則下的三角殺棋的所有必勝著手都找了出來。

以第三章第四節改進記憶體管理的方法，向系統依序要求 2G bytes 共四次，規則為取到最後一子勝的三角殺棋共花費了 61393 秒，規則取到最後一子敗的三角殺棋共花費了 61452 秒。但若以規則為取到最後一子勝的三角殺棋來說，在不改進倒推法的記憶體管理，直接向系統要求配置 8G byte 的記憶體，則花費了 63027 秒，也就是會多花 63027 – 61393 = 2614 秒。

八層三角殺棋（規則為下到最後一子為敗）搜尋所有的必勝著手結果如圖 23(每個線段皆為一個著手)，必勝可行著手有六步。

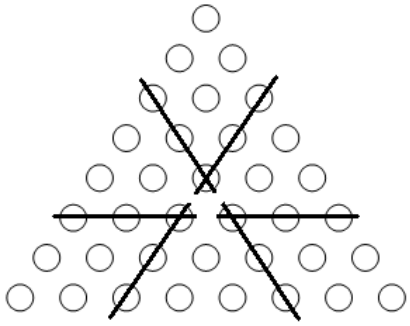

圖 23 規則為下到最後一子為勝，八層三角殺棋第一步必勝著手

八層三角殺棋（規則為下到最後一子為敗）搜尋必勝著手結果如圖 24(每個線段皆為一個著手) ，必勝可行著手有十二步。

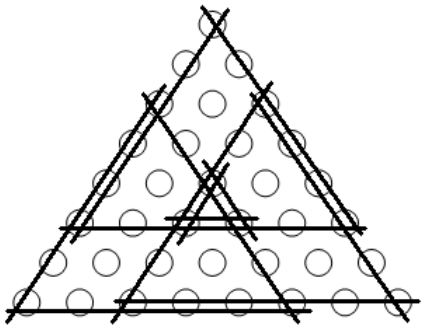

圖 24 規則為下到最後一子為敗，八層三角殺棋第一步必勝著手

## 第五章　結論與未來研究方向

本研究證明了八層三角殺棋的兩個規則下的勝負問題，並且將其必勝的可行著手皆尋找了出來。表 6 為兩個規則下，一到八層三角殺棋的勝負情形。

在研究的過程中，我們也針對三角殺棋的特性，提出了多種的改進方法。以搜尋必勝著手這個方法來說，若我們能效猜測必勝著手，將可降低求出勝負時花費的時間與空間。根據我們的實驗，在下到最後一子為勝的規則之下，七層三角殺棋使用倒推法完整搜尋須花費 161 秒。若直接猜測到必勝著手時，僅需花費 3 秒。八層三角殺棋使用倒推法完整搜尋花費 63027 秒，若直接 猜

測到必勝著手時，僅花費 564 秒。而在研究過程中，我們認為必勝著手是將三角殺棋分割為兩個盤面的著手機率是比較高的。

表 6 一到八層勝負情形

| 規則 / 層數 | 取到最後一子為敗 | 取到最後一子為勝 |
|---|---|---|
| 一層 | 先手敗 | 先手勝 |
| 二層 | 先手勝 | 先手敗 |
| 三層 | 先手敗 | 先手勝 |
| 四層 | 先手勝 | 先手勝 |
| 五層 | 先手敗 | 先手勝 |
| 六層 | 先手勝 | 先手勝 |
| 七層 | 先手勝 | 先手勝 |
| 八層 | 先手勝 | 先手勝 |

雖然在研究過程中花費許多時間在倒推法上，但我們也研究出來所有先前求出的盤面是可以運用到往後幾層的三角殺棋。所以在研究的角度上，獲得了許多寶貴的經驗，對往後研究有很大幫助。我們也提出了一個管理記憶體的方式，使得在求解多層三角殺棋的過程中，盤面資訊狀態可以儲存，這樣就可以利用較少量記憶體解多層三角殺棋的結果了。希望這些經驗能作為未來三角殺棋演算法相關研究的參考。

我們已經知道各層三角殺棋的結果其實是可以拿來利用的，所以未來在解九層的三角殺棋時。或許可以把一些比較簡單的子盤面，先利用倒推法求出結果。類似用填表的方式，把各個子盤面儲存在九層三角殺棋的狀態陣列裡。這樣做的話可以節省一些時間。

然而，九層的三角殺棋棋盤面狀態總數成長得更為龐大，以現有想到的方法要想求得解答仍力有未逮，這是值得未來繼續加以研究的一個方向。

## 誌謝

本研究承蒙國科會提供研究計畫(NSC97-2221-E-003-008)經費補助，謹誌謝忱。

## 參考文獻

[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”，網址：http://en.wikipedia.org/wiki/NimC.

[3] Alan Tucker，“Applied Combinatorics”，John Wiley & Sons; 3rd edition，June 1994.

[4] 群想網路科技，“ＣＹＣ遊戲大聯盟”，網址：http://cyc165.cycgame.com/cyc/cgi-bin/manual.php?i=manG&game=Nim

[5] 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.

[6] 許舜欽，“利用電腦探討七層三角殺棋的勝負問題”，Proc. of 1985 NCS，pp. 798-802，Dec. 1985.