2016 年臺灣國際科學展覽會
優勝作品專輯

作品編號　**190010**

參展科別　電腦科學與資訊工程科

作品名稱　運用資訊熵以及動態結構描述的兩種象棋開局庫表示法之探討與實作

得獎獎項　大會獎：四等獎

就讀學校　臺中市葳格高級中學附設國中

指導教師　梅秀琴

作者姓名　黃克崴

關鍵字　資訊熵、動態資訊結構、開局庫

作者簡介

我是黃克崴，目前就讀於葳格國中三年級。我在學校的成績還不錯，喜歡研究數學。數學從國小開始就是我的興趣與強項，也是我最喜歡的科目。平常除了讀書之外，我也喜歡畫圖。我從國二開始學習程式語言，後來漸漸變成我的興趣。

為了在電腦科學的方面更進一步，我學習並研究人工智慧，並參加了這次科展。我希望未來可以發掘更多的興趣與專長。

摘要

象棋程式主要都是使用評估函數搜索，但是搜索有一定的不穩定性。因此，它們大多會使用開局庫，避免在開局時搜索出不好的棋步，進而影響勝負。不過，一個完整、強大的開局庫通常會佔用很大的空間，而且在檢查、維護與修改上，均有一定的難度。

本研究分別運用靜態和動態結構兩種不同的方式來表示象棋盤面，藉此減少表示一個象棋局面所需要的空間。減少它的空間後，一個相同大小的開局庫中不但可以放入更多資料，增強棋力，而且在搜尋、檢查與修改上，都能夠顯著的增加效率。

使用這兩種新的編碼方法，可以把含有超過十萬局棋(上千萬個局面)的開局庫編碼成 6MB 的大小。而這個開局庫，不像傳統的開局庫，只要檢查到一個不好的著法，就可以迅速的把包含這個著法的局面全部刪除。另外，從這個開局庫也可以直接取得開局的棋譜。

Abstract

Generally, Chinese chess programs use evaluation functions to search moves, but search algorithms are often unstable. Therefore, most chess programs use opening books to prevent their search functions from making bad moves. However, a complete and strong opening book usually takes up a lot of space, and is difficult to examine or modify.

In this study, I will discuss and use two different kinds of board representation methods, static and dynamic coding, to reduce the size needed to represent a chessboard. Once the space needed for a chessboard is reduced, not only can more data be put into an opening book with a same size, but also it can increase the searching efficiency and reduce the difficulty to examine or modify it.

By using the static and dynamic coding methods, an encoded opening book, consisting of more than 10 million moves, takes up only 6 megabytes. Unlike traditional ones, if you find a bad move in it, all chessboards that contain it can easily be deleted. In addition, you can get all the chess games directly from the book.

壹、 前言

近年來，人工智慧的發展已經相當成熟，而其中對電腦象棋的研究更是發達。國內發展的象棋軟體如象棋世家、象棋旋風等，也常在國際的電腦象棋比賽中獲得好成績。自從「深藍」在西洋棋上打敗人類的世界冠軍後，電腦象棋的發展也已經達到可以挑戰人類棋王的實力[20]。

有一天，我從網路上找到了「象棋巫師」。我原本以為電腦程式的棋力應該不強，但是象棋巫師卻能在每一步都思考不到一秒的情況下打敗我。於是我開始對電腦象棋產生好奇心。

象棋巫師引擎的搜尋實力雖能打敗許多業餘玩家，但是有時候還是會走出奇怪的走法，這是搜索的不穩定性所造成的。如果搜索函數不是完美的，那它就會不穩定。

一般在開局的時候，子力與勝負都不明顯，因此程式很難判斷這個局面的好壞，進而增加其不穩定性。不穩定性越高，走出壞著法的機會也越高，最終可能使程式輸掉整盤棋。解決這個問題的方法有兩種：

一、 使這個搜索函數成為完美的。這個方法顯然是不可行的，因為評估函數不是準確的，而且搜索的層數也有限。

二、 建立一個開局庫。這個想法非常簡單，就是把「好的」開局走法放到一個資料庫中，就可以避免程式走壞的著法。

我發現象棋巫師沒有很強的開局庫，常常走沒幾步就「脫譜」了。於是，我用 VB 寫了一個以開局庫為主的電腦象棋程式，叫做「棋怪」，希望能透過一個強大、完整的開局庫來彌補一般搜尋方法的不足。

後來我發現，建立開局庫並不是很難，但如果要把大量的局面放入開局庫，就必須要有一個好的編碼方法。象棋巫師的開局庫內有 10,000 個局面和著法，大小為 95 KB [29]。如果使用相同的編碼方法，在自己的開局庫中放 100 萬個局面，那大約會使用 10 MB 的空間。而且，這個開局庫很難再做檢查或修正。

因此，如果要建立一個非常強大的開局庫，就必須先想辦法減少表示一個局面所需的空間，這也就是本研究的目的。

研究設備與器材：

一、 電腦一台。

執行程式的環境：Intel® Pentium® Processor N3540, 2.66 GHz; 4.0GB DDR3; Windows 10

二、 Microsoft Visual Basic

三、 象棋巫師軟體。

貳、 研究方法與過程

一、 基本搜尋技術

(一) 棋譜搜索

一局象棋可以分為開局、中局和殘局三個部分。開局時，雙方都必須將己方的棋移動到適合攻擊或防禦的位置，以利接下來局勢的發展。但是，棋局剛開始時，雙方棋子數目都很多，勝負也不明顯。如果只使用搜尋法來搜尋開局局面，可能會因此落入對方佈下的陷阱[10]。

人類棋手大多有累積許多的開局經驗，知道開局時哪些著法會對自己比較有利。如果只用搜尋的程式和人類棋手對戰，在開局時就會大大失利。因此，大部分電腦象棋程式都會選擇建立一個「開局庫（Opening Book）」。

開局庫儲存著開局常遇到的盤面和其應對的走法，當遇到開局庫內有的盤面時，就可以直接從開局庫內取得這個盤面所對應的著法。它是幾乎所有象棋程式必備的，因為它既可以縮短程式思考的時間，也可以避免落入對方的佈局陷阱中。

開局時的搜尋方法為：

1. 我們必須收集大量的開局局面和其對應的走法。
2. 將這些局面全部編碼之後，和它所對應的著法寫到開局庫中。
3. 若要在開局庫中找到一個棋局N，就檢查開局庫中每一項是否與N相同。為了增加搜尋效率，可以使用二分搜尋。
4. 如果相同，就找出這個棋局中的對應走法，並執行它。

5. 執行流程：

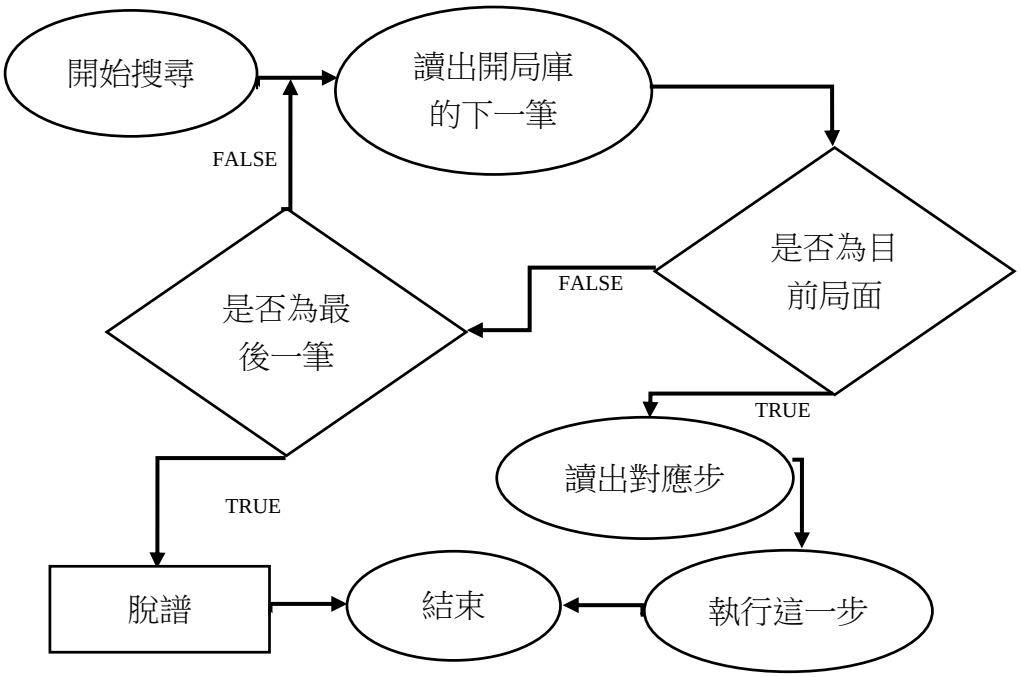


圖 1-1-1 執行流程圖

6. 如果在開局庫沒有找到資料，就代表「脫譜」了，也進入中局階段。
7. 一個局面可能會有數個對應的走法。如果不只一個，就把它們視為不同的資料，並從中隨機選擇一個。

(二) 評估函數搜索

在象棋對弈時，紅黑雙方都知道每顆棋子的位置，敵我雙方輪流走棋，並且可以走任何合法的著法。下棋的目的是以對自己最有利的著法來將死對方和避免自己被將死。

電腦象棋的程式，是使用搜索函數來尋找對於程式來說最好的著法。搜索函數大多使用「樹狀搜索」：一個象棋局面可以看成一棵很大的搜尋樹，而目前棋盤上的局勢就是「根節點」。從這個局面走一步棋，局面就到達它的「分枝」，這些局面稱為「後續節點」。每個局面(只要不是結束局面)的後面都還有一系列分枝，也就是這個局面所有合法的著法[29]。

當棋局進入中局時，已經無法使用開局庫內的資料。因此，我們必須要用一個局面評估函數來進行樹狀搜索。

(三)　局面評估函數

當樹狀搜索達到葉節點時，就必須傳回這個局面的分數，作為父節點比較的依據。局面評估函數可以對一個盤面評分，進而判斷哪個局面對己方最有利[25]。評分函數要考慮的因素有：

1. 子力

子力就是棋盤上的棋子價值總和，一般棋子越多子力也就越高。每一種棋的分數都不同：將帥是最重要的，分數也是最高的；車的分數其次；而兵卒的分數最低。各種棋子的分數大致如下[20]：

種類	將、帥	士、仕	象、相	馬、傌	車、俥	包、炮	卒、兵
分數	5000	200	250	450	1000	500	100

表 1-3-1 子力價值表

子力的計算方法就是將己方的棋子分數總和減掉對方的總和。

例如：圖 1-3-1 中，黑方子力較高，佔優勢。

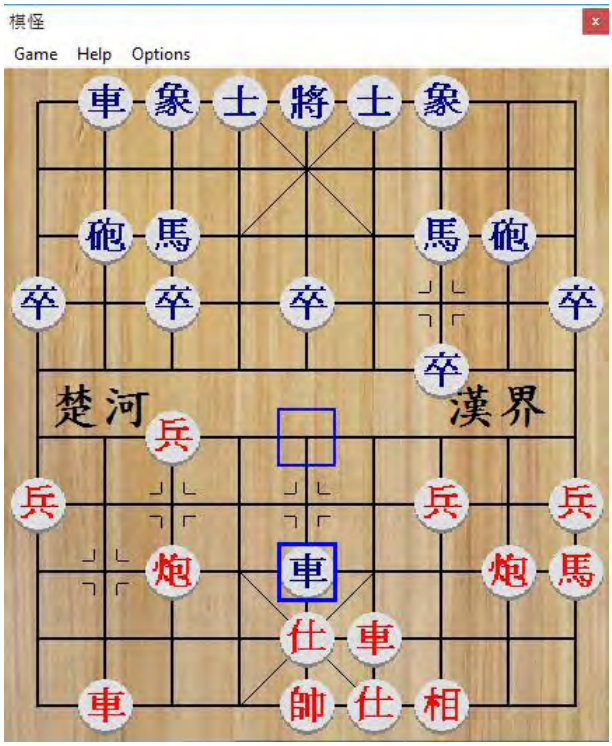


圖 1-3-1 子力

同一顆棋子在不同的階段中，分數可能不同，例如殘局中兵卒的數量往往是決勝的關鍵[19]。

2. 位置

象棋中除了子力以外，位置也是需要考慮的。每顆棋子在不同位置時會有不同的分數，例如兵卒過河之後會較有利。舉例：

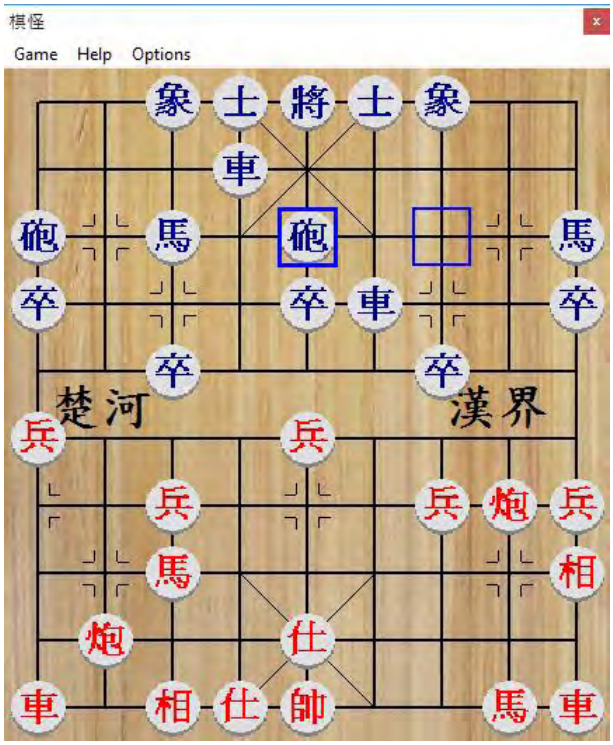


圖 1-3-2 位置

圖 1-3-2 中，雙方子力相當，不過黑方棋子的位置比紅方好，明顯占優。

3. 將帥的安全性

當一方被將軍時，合法的步數會減少，還可能因此失去一些子力[29]。如圖 1-3-3，將軍抽車就是典型的例子。因此，開局和中局時，保護將帥是很重要的。

另外，有時到了殘局，將帥就有了攻擊的作用。在圖1-3-4中，紅方就是利用「王不見王」的特殊規則來將死黑方。（中國象棋中，將、帥不能在同一直線上直接照面，稱作「王不見王」。）

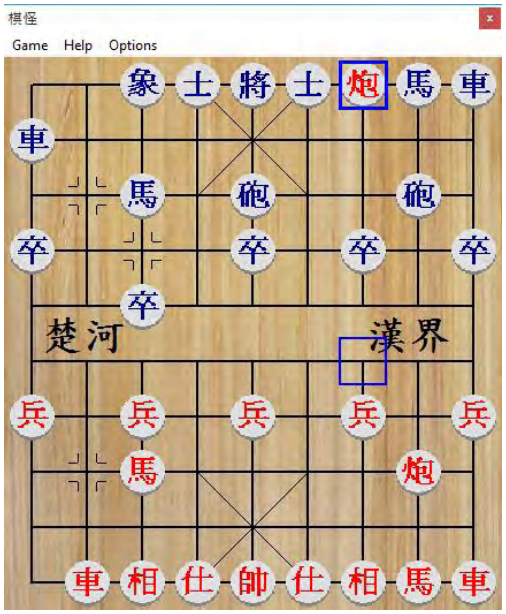


圖 1-3-3 將軍抽車

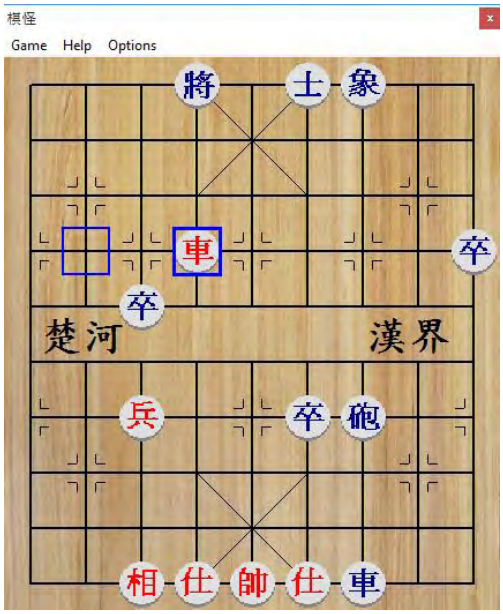


圖 1-3-4 白臉將殺

4. 特殊棋型

有些特殊的棋型如連環馬、巡河車、沉底炮等，對局勢的發展是非常有利的，必須給予加分[20]。

舉例來說：

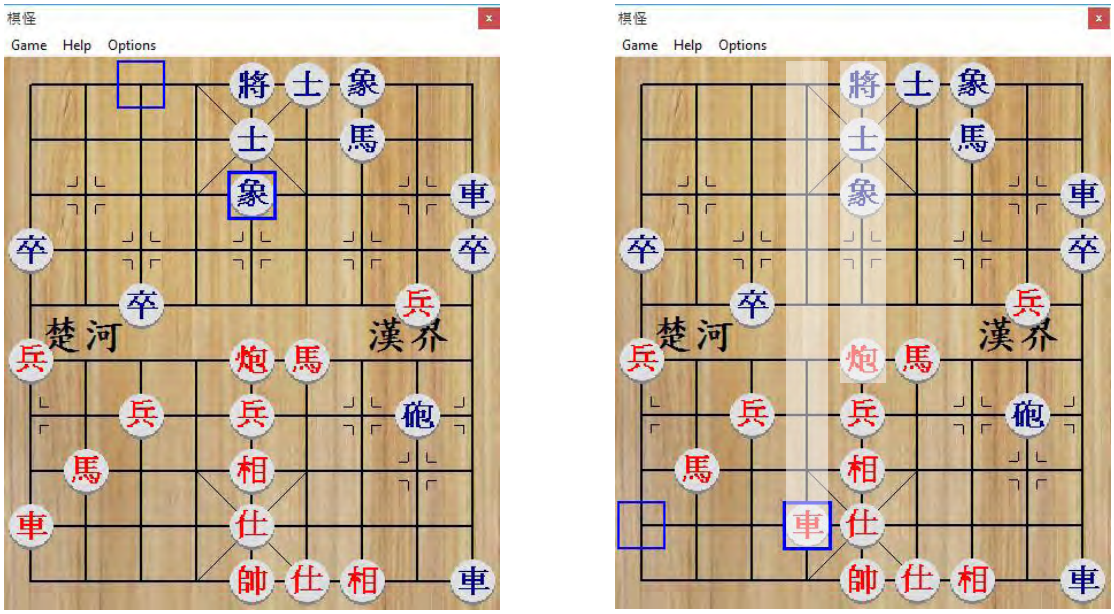


圖 1-3-5、1-3-6 鐵門閂

目前輪紅方走。雖然紅方走「馬四退二」吃掉黑砲可以取得優勢，但走「車九平六」形成鐵門閂會對目前局勢更有利。事實上，走「車九平六」之後可以找到十四層的殺局；走「馬四退二」之後則形成和局。

(四)　MiniMax 搜尋演算

象棋的搜尋樹非常廣也非常深，且每個節點有 35 個分枝。每局棋都是一棵非常巨大的樹。有一種演算法能通過樹狀搜索找到棋局中對雙方來說最好的著法，稱為「最小-最大搜尋」 (Minimax Search or Min-Max Search)。

我們可以用基於最小-最大搜尋的程式來下象棋，但由於著法搜尋樹的龐大，因此用最小-最大搜尋整棵樹是不可行的，所以只能在一個給定的局面下搜尋有限的幾層[29]。

Minimax 的原理：當輪到我方走時，我方會走出對我方最有利的一步；輪到對方走時，他會走出對我方最不利的一步。也就是說，我方會盡量使局面對我方有利，而對方會盡量使局面對我方不利。使用樹狀搜索時：

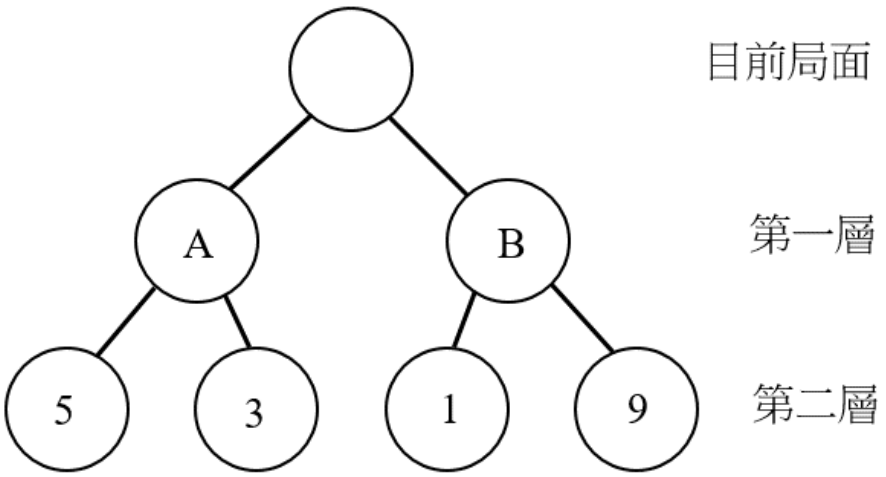


圖 1-4-1 Minimax 搜索

葉節點的數字就是對我方來說的分數，分數越高，局面就對我方最有利。如果要走的一方選擇 A 節點，那對手就會選分數為 3 的那個節點。如果要走的一方選擇 B 節點，那對手就會選分數為 1 的那個節點。因此，對於要走的一方來說，A 節點是較好的選擇，因為在最壞的情況下還是能得到 3 分。

以下是一個簡單的 Minimax 程式碼(以 VB 為例)：

```
Function Minimax(depth As Integer) As Long        'Depth 層數
    Dim best As Long = -Infinity
    If depth = 0 Then Return Evaluate()    葉節點
    Call GenerateMoves()    產生所有合法著法
    Dim M As Move, val As Long
    For Each M In LegalMoves    對於每一步
        Call MakeMove(M)    走棋
        遞迴呼叫，層數-1
        val = -Minimax(depth - 1)    注意 Minimax 前有負號。
        Call UndoMove(M)    還原走的那一步
        計算最大值
        If val > best Then best = val
    Next
    Return best    返回最大值
End Function
```

程式 1-4-1 Minimax 搜索

這個函數可以這樣呼叫：

```
Bestmove = Minimax(5)
```

程式 1-4-2 Minimax 呼叫

傳回的是搜尋 5 層深的結果。

要注意的是，對方走棋時，傳回的是對我方而言的分數，所以要加負號，才是對對方來說最好的分數。

(五)　Alpha-Beta 搜尋演算

使用 Minimax 搜尋雖然可以找到對雙方來說最好的一步，但是卻非常沒有效率。以每個節點有 35 個分枝來說，搜尋 1 層有 36 個節點；2 層有 1261 個；要是搜尋 10 層就有多達2.84×10^{15}個節點。因此，如果想用樹狀搜索來搜尋很大的深度時，必須盡可能的減少分枝數量才行[5]。Alpha-Beta 裁剪法是利用兩個變數：最大值α(Alpha)和最小值β(Beta)，來達成減少分枝數的目的。

就如上面所說，Alpha 就是對我方最有利的一步，也就是 Minimax 中的 Best；而 Beta 值就是最不利的一步。

Alpha-Beta 和 Minimax 在邏輯上是一樣的，只是在遞迴時多了兩個引數而已。

在搜尋時，假如有一個子節點的分數超過 Beta，那麼這棵子樹可以被裁剪掉（不用搜尋了）。

用圖形來說明：

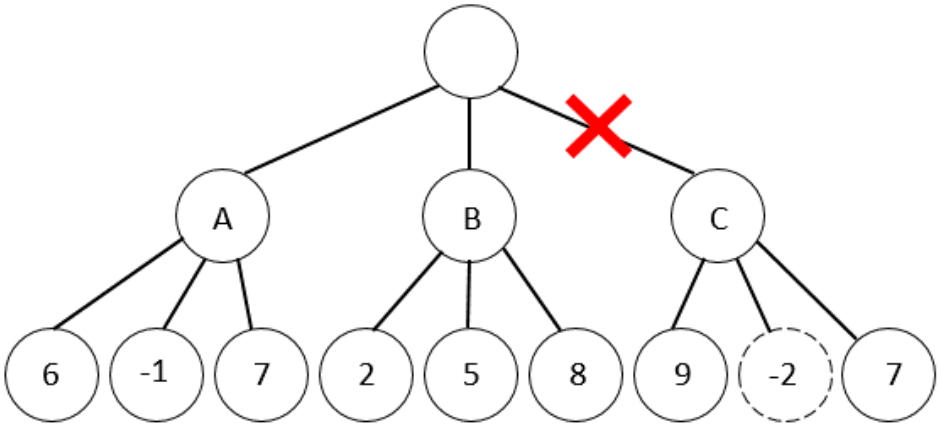


圖 1-5-1 Alpha-Beta 搜索

請看 C 節點的第二個子節點 -2。這個局面的分數低於 B 節點的最差值 Beta (也就是 2)，所以若我方選擇了這個節點，就會得到比 B 更低的分數。因此，程式可以不用再搜尋 C 的子樹了。

Alpha-Beta 的程式和 Minimax 差不多，只是多了兩個變數和裁剪（Cut-off）的部分。用 VB 寫的程式如下：

```
Function AlphaBeta(depth As Integer, alpha As Long, _
beta As Long) As Long
    If depth = 0 Then Return Evaluate()
    Call GenerateMoves()
    Dim M As Move, val As Long
    For Each M In LegalMoves
        Call MakeMove(M)
        val = -AlphaBeta(depth - 1, -beta, -alpha)
        Call UndoMove(M)
        If val > alpha Then alpha = val
        If val >= beta Then Return beta      'Cut-off
    Next
    Return alpha
End Function
```

程式 1-5-1 Alpha-Beta 搜索

己方的最大值會等於對方的最小值，而對方的最大值會等於己方的最小值；對對方而言的分數會等於我方分數的相反數，所以程式遞迴呼叫時要將 Alpha 和 Beta 取負數並對調[29]。

這個算法的確大幅增加了搜尋的效率。若每秒搜尋 10000 個節點，則使用 Minimax 搜尋 4 層約需 2 分鐘，但使用 Alpha-Beta 在相同盤面上搜尋 4 層，只需要不到 1 秒。

Alpha-Beta 搜尋能裁掉大部分沒有用的著法，這就是它能節省大量時間的原因。

(六)　Principal Variation

PVS(主要變動搜索)是增加 Alpha-Beta 搜尋效率的一種方法。

在 PV Search 中，所有節點都可被分為三種[29]：

1. Alpha 節點。這個節點的所有值都小於或等於 Alpha，代表對走棋的一方不利。
2. Beta 節點。子節點中有大於或等於 Beta 的值。Beta 節點會被裁剪掉。
3. Principle variation(PV)節點。子節點都小於 Beta，且有一個或數個著法大於 Alpha（PV 著法）。

程式有時可以一開始就判斷出目前的節點是屬於哪一種。如果得到的第一個值大於或等於 Beta，那麼這個節點就是 Beta 節點。如果第一個節點小於或等於 Alpha，那麼它可能是 Alpha 節點。如果第一個值介於 Alpha 和 Beta 之間，那它可能是 PV 節點。

然而，程式卻不能在一開始就保證它是哪一種節點。如果第一個值大於或等於 Beta，那就可以確定它是 Beta 節點，但如果第一個值是 PV 著法或是小於 Alpha，後面的著法可能會有更高的值。

對於一個 PV 節點，可以假設它是著法裡面最好的，並且用寬度為 1 的邊界 (depth - 1, -alpha, -alpha + 1) 搜尋（稱作「零窗口裁剪」），讓裁剪做得更快。假如假設錯誤，也就是有著法比它更好，再使用正常的邊界去搜尋。

PV Search 的程式只比 Alpha-Beta Search 多了幾行，其它大致上相同。

PVS 的 VB 程式如下：（***斜粗體***是比 Alpha-Beta 搜尋多的部分）

```
Function AlphaBeta(depth As Integer, alpha As Long, _
beta As Long) As Long
If depth = 0 Then Return Evaluate()
Call GenerateMoves()
Dim M As Move, val As Long, PV As Boolean = False
For Each M In LegalMoves
   Call MakeMove(M)
   If PV Then
      val = -AlphaBeta(depth - 1, -alpha, -alpha + 1)
      If val > alpha And val < beta Then
         val = -AlphaBeta(depth-1, -beta, -alpha)
      End If
   Else
   val = -AlphaBeta(depth - 1, -beta, -alpha)
   End If
   Call UndoMove(M)
   If val > alpha Then
      alpha = val
      PV = True
   End If
   If val >= beta Then Return beta 'Cut-off
Next
Return alpha
End Function
```

程式 1-6-1 PVS 搜索

PVS 用的時間比 Alpha-Beta 少了約 10%。副作用是，可能提高搜尋時的不穩定性[29]。

(七)　寧靜搜索

象棋中，常有「換子」的情況會發生。如果使用 Alpha-Beta 搜尋，可能會有不適當的分數出現[18]。當 Alpha-Beta 搜索達到葉節點時，會立即傳回 Evaluate，即使一方的車被吃，或者被將軍。

一種解決的方法叫「寧靜搜索」（Quiescent Search）。當搜索達到葉節點時，改用呼叫寧靜搜索代替 Evaluate。

它也是局面評估函數的一種，而且可以避免在換子時給予錯誤的評分。

用 VB 寫的寧靜搜索函數如下：

```
Function Quiescent(alpha As Long, beta As Long) As Long
    Dim val As Long
    val = Evaluate()
    If val >= beta Then Return beta
    If val > alpha Then alpha = val
    Call GenerateCaptures()
    Dim M As Move
    For Each M In LegalCaptures
        Call MakeMove(M)
        val = -Quiescent(-beta, -alpha)
        Call UndoMove(M)
        If val > alpha Then alpha = val
        If val >= beta Then Return beta
    Next
    Return alpha
End Function
```

程式 1-7-1　寧靜搜索

一般來說，寧靜搜索只考慮吃子。它的原理和 Alpha-Beta 相同，但是沒有「深度」的引數。因此，如果它找到一個非常好的著法，它將會搜索到很深的深度。

(八)　MTD(f)

MTD-f（Memory-enhanced Test Driver）搜尋是一種增加 Minimax 搜尋效率的方法，並且可以用來取代 Alpha-Beta 剪枝法。

MTD 搜尋使用零窗口裁剪進行搜尋，和 Alpha-Beta 的窗口相比，明顯提高了剪枝的效率。它的缺點是，會大幅提高搜尋的不穩定性，除非有一個非常精確的局面評估函數。

MTD 的 VB 程式碼：

```
Function MTD(f As Long, d As Long) As Long
    Dim g, up, low, b As Long
    g = f
    up = Infinity
    low = -Infinity
    Do While up > low
        b = IIf(g = low, g + 1, g)
        g = AlphaBeta(d, b - 1, b)
        If g < b Then
            up = g
        Else
            low = g
        End If
    Loop
    Return g
End Function
```

程式 1-8-1 MTD(f)搜索

(九)　迭代加深

一般來說，象棋比賽有限定思考時間，要預測搜尋的時間是非常困難的。

迭代加深是一個決定搜尋深度的方法。先從一層的搜尋開始，每次搜尋深度多一層，直到時間用完為止[29]。只要時間一到，就可以立刻停止搜索。

這個方法還有一個優點。第一層搜尋時，已經將第一層節點的順序排列好了。搜尋第二層時，可以將第一層的結果做為排序參考。

二、　棋盤表示方法

在象棋程式執行時，常常要比較兩個盤面是否相同。若一一比較棋盤上每顆棋子的位置，會非常沒有效率[29]。因此，程式必須使用一個值來代表這個盤面。

(一)　Zobrist 鍵值

其中一個方法就是建立一個標籤，通常是 64 位元。Zobrist 鍵值就是利用隨機數來建立標籤的一種方法。建立 Zobrist 鍵值的方法為：

1. 先建立一個多維的 64 位元陣列，每個陣列包含一個隨機數。這個陣列要有三維：
   (1) 棋子屬於哪一方（紅、黑）
   (2) 棋子的種類（帥、仕、相、俥、傌、炮、兵）
   (3) 棋子的位置（0～89）
2. 程式開始時，把這個陣列用 64-Bit 的亂數填滿。
3. 將鍵值設為 0。

4. 找到棋盤上的每個棋子，並讓鍵值和那顆棋子的鍵值做 XOR 運算。

5. 若目前輪紅方走，就要再把鍵值和一個隨機的常數做 XOR 運算，若輪黑方走則不用。

6. 實例：

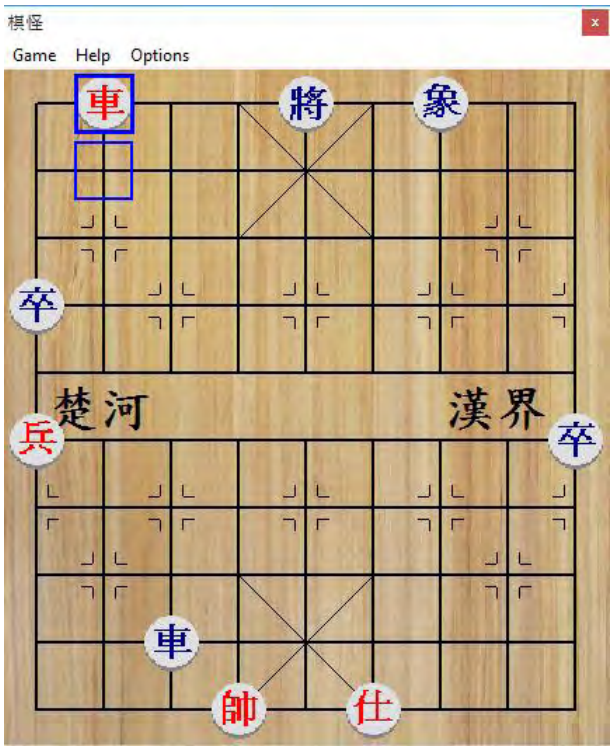


圖 2-1-1 Zobrist Key 範例

象在圖 2-1-1 中位置的鍵值為 4544540635928351949，和目前的 Zobrist Key (也就是 0) 做 Xor 運算，得到 4544540635928351949。我們再把 Zobrist Key 4544540635928351949 和將的鍵值 14910944495801097990 做運算，得到 3143544568248613674。以此類推，最後得到此盤面的鍵值 10916958495384572327。因為目前輪黑方走，所以不須做其他運算。

另外，64 位元的鍵值無法表示棋盤上所有可能出現的盤面，所以會有發生碰撞的可能。

(二)　置換表

象棋中，不同的走法可能走到同一個盤面。重複搜尋同一個盤面會浪費很多時間。如果之前已經搜尋過 X 盤面，又重複遇到 X，這時可以不用再搜尋 X 的子樹，直接用 X 在置換表中的分數代替[20]。

置換表可以利用 Zobrist 鍵值來實現。搜尋的過程中，程式將搜尋過的盤面鍵值和其分數存到記憶體中，當第二次遇到同一個盤面時，就使用第一次保存的值當作這次的分數。

置換表是一個 Hash Table。一個盤面的 Zobrist Key 除以置換表總項數的餘數，就是這一個盤面在置換表裡的位址。為了縮短計算餘數的時間，置換表的大小幾乎都是 2^n。

如果要在置換表中找到棋局 N，就算出它的鍵值，然後將它除以置換表總項數，得到餘數 R。接著，檢查置換表中第 R 項的鍵值是否與 N 的鍵值相同。如果是，就把它在置換表中的分數找出來，當作盤面 N 的分數。如果不是，就把 N 的鍵值和分數放到置換表中。

置換表的大小和電腦的搜尋速度有關。一般來說，搜尋速度越快，置換表的大小就越大。ChessBase 作者史蒂夫・洛佩茲提出過這樣的公式：

置換表大小(KB) = 2 × CPU 速度(以 Mhz 計算) × 搜尋每步平均秒數

(三)　FEN 記號法

FEN (Forsyth-Edwards Notation)是一種用利用 ASCII 字元來表示中國象棋與西洋棋盤面的方法。在中國象棋中，使用 FEN 記錄局面 N 的方法為[10]：

1. 從 N 的第一個橫列開始，由左至右編碼。遇到棋子時，就用那顆棋子對應的字母來表示。每一種棋子都各有一個對應的字母：

	將、帥	士	相、象	車	馬	炮	兵、卒
紅方	K	A	B	R	N	C	P
黑方	k	a	b	r	n	c	p

表 2-3-1 棋子對應字母

遇到空格時，就用數字表示。若有連續 k 個空格，就用阿拉伯數字 k 來表示。

2. 每一橫列表示完後，就用斜線 / 來表示換行，然後開始表示下一列。
3. 十列都表示完後，還有五個部分，每個部分用一個空格隔開：
   (1) 輪到誰走。”r”表示紅方，”b”表示黑方。
   (2) 易位可行性。由於中國象棋沒有易位的規則，所以用 – 表示。
   (3) 吃過路兵目標格。由於中國象棋沒有吃過路兵的規則，所以這項也用 – 表示。
   (4) 半回合計數，指從上一次吃子到現在經過的回合數。中國象棋有六十步不吃子就和棋的規則。
   (5) 從棋局開始到現在所經過的回合數。

4. 實例：

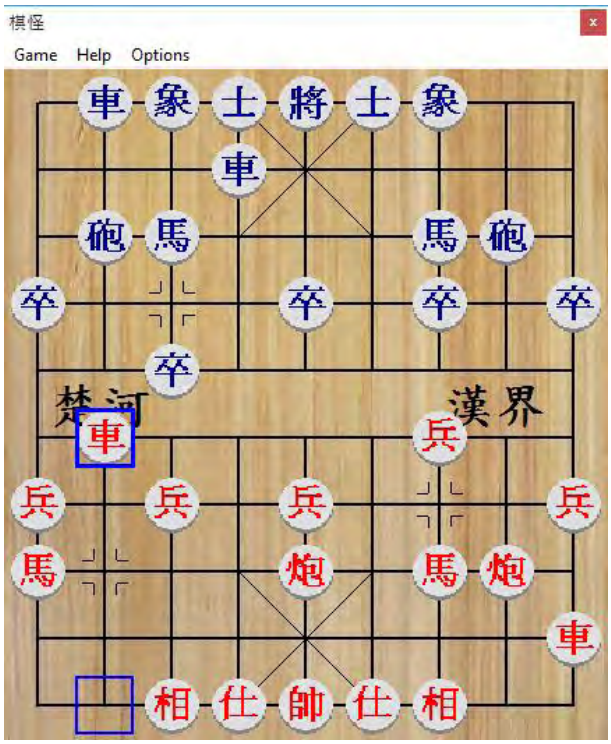


圖 2-3-1 FEN 記號法範例

這個盤面以 FEN 記號表示為：

1rbakab2/3r5/1cn3nc1/p3p1p1p/2p6/1R4P2/P1P1P3P/N3C1NC1/8R/2BAKAB2 b - - 7 7

(四)　PGN 棋譜格式

PGN 是一種用來表示中國象棋棋譜的方法。它是一種通用的格式，幾乎所有象棋軟體都可以進行讀取與編輯，而且人類在閱讀和打譜上也很方便。

一個完整的 PGN 棋譜包含以下幾個部分：

1. 標籤部分，例如：

```
[Game "Chinese Chess"]
[Event "過宮炮對進左馬"]
[Round ""]
[Date ""]
[Site ""]
[Red ""]
[RedTeam ""]
[Black ""]
[BlackTeam ""]
[Result "*"]
[ECCO "A61"]
[FEN "rnbakabnr/9/1c5c1/p1p1p1p1p/9/9/P1P1P1P1P/1C5C1/9/RNBAKABNR r - - 0 1"]
```

表 2-4-1 棋譜標籤

2. 棋譜部分，例如：

```
1.  炮八平四 馬２進３
2.  兵七進一 車１平２
3.  馬八進七 炮２平１
4.  馬二進三 卒７進１
5.  炮二進四 象３進５
6.  炮二平七 炮８平７
7.  相三進五 車９進１
8.  車九進一 車９平６
9.  炮四進四 馬８進９
10. 車一平二 車２進４
11. 車二進四 卒９進１
12. 車九平四 士６進５
 *
```

表 2-4-2 棋譜

3. 註解。PGN 裡的註解是寫在大括弧{ }裡面，例如：

60. 馬五進六 車６平４ {這一步使得紅方的相動彈不得。} 61. 馬六退八 {紅方可能因為太過緊張，丟了士。} 61. 車４進１ {黑大優。}

表 2-4-2 棋譜註解

為了更方便收集和分析棋譜，棋怪也有讀取 PGN 棋譜的功能。

(五) 傳統的盤面表示方法

Zobrist 鍵值可以用於置換表，只有 64 Bits，計算容易，但是無法倒推回原本的棋盤，而且還會有碰撞的問題。

傳統的盤面紀錄方法，不但可以倒推回原本的棋盤，而且不會互相碰撞。它的紀錄方法為：

1. 用 4 個位元代表每一種棋子，0000 代表空格。

種類	將、帥	士	象、相	車	馬	炮	卒、兵
紅	0001	0010	0011	0100	0101	0110	0111
黑	1001	1010	1011	1100	1101	1110	1111

表 2-5-1 棋子對應編碼

2. 將棋盤上的每個格子編號 1～90。

3. 依序將棋盤上的每個格子上的棋子編號連接起來。

4. 最後加上一個 0 或 1，代表輪紅方(0)還是黑方(1)走。

5. 實例：

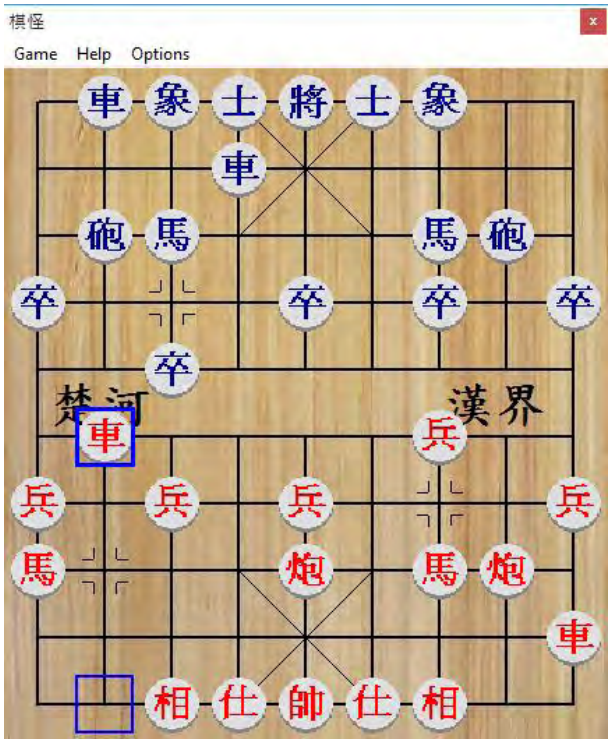


圖 2-5-1 傳統盤面編碼範例

這個盤面用位元來表示為：

0000 1100 1011 1010 1001 1010 1011 0000 0000
0000 0000 0000 1100 0000 0000 0000 0000 0000
0000 1110 1101 0000 0000 0000 1101 1110 0000
1111 0000 0000 0000 1111 0000 1111 0000 1111
0000 0000 1111 0000 0000 0000 0000 0000 0000
0000 0100 0000 0000 0000 0000 0111 0000 0000
0111 0000 0111 0000 0111 0000 0000 0000 0111
1101 0000 0000 0000 0110 0000 0000 0110 0000
0000 0000 0000 0000 0000 0000 0000 0000 0100
0000 0000 0011 0010 0001 0010 0011 0000 0000
1

總共用了 361 個位元，十分浪費空間。

三、 利用資訊熵的象棋盤面表示法

(一) 資訊熵

若共有 i 筆資料，出現的機率分別為 P_i，根據 Claude Shannon 的資訊量（夏儂熵）公式，它的平均資訊量 Q(以 Bit 為單位)為[27]：

$$Q = -\sum_{k=1}^{i} P_k \log_2 P_k$$

夏農熵證明了，這 i 筆資料至少需要 Q 個位元來表示。也就是說，如果能根據這個公式算出一個象棋盤面的資訊量，那就可以使用最少的位元數來表示一個象棋局面。

如果要使用資訊熵編碼來為一筆資料編碼，那這筆資料最理想的編碼長度是 $-\log_2 P_k$ Bits。但是，因為一個象棋盤面的變化量有10^{48}種，所以算出每一個象，只能分別算出每顆棋子在每個位置出現的機率，再個別做編碼。

(二) 機率統計

為了對盤面進行資訊熵編碼，我必須先統計每顆棋子在每個位置上出現的機率。我從網路上找了 60000 多個 PGN 棋譜，並從裡面選擇了 1200 局作分析。

分析完成後，我將結果左右合併，並且轉換為每十萬次中出現的機率。最後的結果如下：

將、帥				不在棋盤上(處於死亡狀態)的次數：0				
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
楚河				漢界				
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	235	280	235	0	0	0
0	0	0	1488	867	1488	0	0	0
0	0	0	4263	85332	4263	0	0	0

表 3-2-1 將出現在各個位置的機率

士、仕				不在棋盤上(處於死亡狀態)的次數：4215				
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
楚河				漢界				
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	13537	0	13537	0	0	0
0	0	0	0	19359	0	0	0	0
0	0	0	24675	0	24675	0	0	0

表 3-2-2 士出現在各個位置的機率

象、相				不在棋盤上(處於死亡狀態)的次數：10389				
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
楚河				漢界				
0	0	2322	0	0	0	2322	0	0
0	0	0	0	0	0	0	0	0
1785	0	0	0	25707	0	0	0	1785
0	0	0	0	0	0	0	0	0
0	0	27832	0	0	0	27832	0	0

表 3-2-3 象出現在各個位置的機率

馬				不在棋盤上(處於死亡狀態)的次數：30604				
37	56	35	28	6	28	35	56	37
32	135	365	104	18	104	365	135	32
86	109	354	106	199	106	354	109	86
278	446	945	588	436	588	945	446	278
68	186	135	1007	697	1007	135	186	68
楚河				漢界				
56	1256	793	1844	392	1844	793	1256	56
73	154	358	292	1480	292	358	165	73
3109	226	13391	340	248	340	13391	226	3109
259	63	213	658	563	658	213	63	259
15	4143	163	171	4	171	163	4143	15

表 3-2-4 馬出現在各個位置的機率

車				不在棋盤上(處於死亡狀態)的次數：35079				
126	429	237	110	45	110	237	429	126
69	352	226	441	63	441	226	352	69
101	374	288	186	145	186	288	374	101
407	1072	1090	1168	774	1168	1090	1072	407
188	474	290	593	420	593	290	474	188
楚河				漢界				
186	1361	978	1209	518	1209	978	1361	186
206	743	404	720	585	720	404	743	206
330	464	174	385	95	385	174	464	330
777	501	246	816	24	816	246	501	777
7710	4822	455	409	18	409	455	4822	7710

表 3-2-5 車出現在各個位置的機率

炮				不在棋盤上(處於死亡狀態)的次數：29909				
610	416	232	58	31	58	232	416	610
179	293	213	166	40	166	213	293	179
112	360	253	114	138	114	253	360	112
838	941	1119	438	1203	438	1119	941	838
253	277	149	295	1053	295	149	277	253
楚河				漢界				
322	811	543	299	808	299	543	811	322
96	347	248	236	386	236	248	347	96
2073	7653	2285	3234	8618	3234	2285	7653	2073
430	431	850	506	486	506	850	431	430
73	149	348	380	95	370	347	149	73

表 3-2-6 炮出現在各個位置的機率

卒、兵				不在棋盤上(處於死亡狀態)的次數：35752				
1	8	19	6	2	6	19	8	1
11	16	71	88	20	88	71	16	11
16	30	83	38	49	38	83	30	16
94	116	310	345	331	345	310	116	94
265	156	460	264	572	264	460	156	265
楚河				漢界				
1584	0	4031	0	1440	0	4031	0	1584
12953	0	5005	0	9897	0	5005	0	12953
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

表 3-2-7 兵出現在各個位置的機率

(三) 減少空間的證明

若根據統計的機率來對棋子做編碼，使用的空間一定不比沒統計的還多。也就是說，使用統計出的機率進行編碼，可以節省空間。這是可以用數學證明的。

設一顆棋子有n種可能出現的位置，出現的機率皆為P，則這顆棋子在某個位置的平均資訊量為：

$$\mathrm{Q} = -\sum_{k=1}^{n} P \cdot \log_2 P$$

$$= -n \cdot P \cdot \log_2 P$$

$$\because P = \frac{1}{n}$$

$$\therefore \mathrm{Q} = -n \cdot \frac{1}{n} \cdot \log_2 \frac{1}{n} = -\log_2 \frac{1}{n} = -\log_2 n$$

假設有統計機率，且同一顆棋子出現在第k個位置的機率為P_k，則這顆棋子在某個位置的平均資訊量為：

$$\mathrm{Q} = -\sum_{k=1}^{n} P_k \cdot \log_2 P_k$$

於是：

$$\mathrm{Q} = -\sum_{k=1}^{n} P_k \cdot \log_2 P_k \leq \log_2 n$$

設$x_1 = x_2 = x_3 = \cdots = x_n = \frac{1}{n}$，$f(x) = \log_2 x$，則原不等式變成：

$$\mathrm{Q} = -\sum_{k=1}^{n} P_k \cdot \log_2 P_k \leq \log_2 n$$

$$\Rightarrow -\sum_{k=1}^{n} P_k \cdot \log_2 P_k \leq -\log_2 \frac{1}{n}$$

$$\Rightarrow -\sum_{k=1}^{n} P_k \cdot f(P_k) \leq -f\left(\frac{1}{n}\right)$$

$$\Rightarrow -\sum_{k=1}^{n} P_k \cdot f(P_k) \leq -f\left(\sum_{k=1}^{n} x_k \cdot P_k\right)$$

此時即是廣義的琴生不等式。

(四)　熵編碼的實作

為了得到較佳的編碼，我使用了霍夫曼樹。霍夫曼編碼的做法：

1. 建立一個森林，而這個森林中的每一棵樹就是一筆資料。給每棵樹一個權值，也就是這筆資料出現的機率。
2. 當有超過一棵樹時，就重覆以下動作：
   (1) 建立一棵新的樹，並把權值最低的兩棵樹加到它的子樹。

(2) 這棵新的樹的權值等於兩個子節點的權值總和。

(3) 將這棵新的樹加到森林中。

3. 最後剩下的樹就是霍夫曼樹。

使用霍夫曼樹將資料編碼，可以得到一個接近它們平均資訊量的編碼長度。若要使用 VB 對一顆象棋棋子進行霍夫曼編碼，方法如下：

1. 宣告一個類別「節點」：

```
Type Node
    Key As Integer          權值
    Address As Integer      位址
    Children(1) As Integer  兩個子節點的位址
    Coding As String        編碼
    Leaf As Boolean         是否為葉節點
End Type
```

程式 3-4-1 宣告類別

2. 建立 91 個葉節點，每個葉節點的權值就是這顆棋子在每個位置上出現(包括死亡)的機率。

3. 重覆以下動作：

   建立一個新的節點，並將權值最低的兩個節點加到它的子節點中。而它的權值就是兩個子節點的權值之和。

4. 全部完成之後，就可以得到棋子 N 在不同位置的編碼。

我使用了這個方法為象棋中的七種棋子編碼，得到以下結果：

1. 帥、將：平均約 1.31 Bits
2. 士：平均約 2.49 Bits
3. 相、象：平均約 2.35 Bits
4. 馬：平均約 4.12Bits
5. 車：平均約 4.53Bits

6. 炮：平均約 4.61Bits

7. 兵、卒：平均約 3.27 Bits

因此，使用此方法表示一個象棋盤面平均約需要：

$$(1.31 + (2.49 + 2.35 + 4.12 + 4.53 + 4.61) \times 2 + 3.27 \times 5) \times 2 + 1 = 108.72\text{Bits}$$

我拿了 100 個盤面做測試，發現都可以成功的編碼和解碼，平均用了 107.3 個位元。和傳統的比較起來，這個方法省了非常多的空間。

編碼方法：

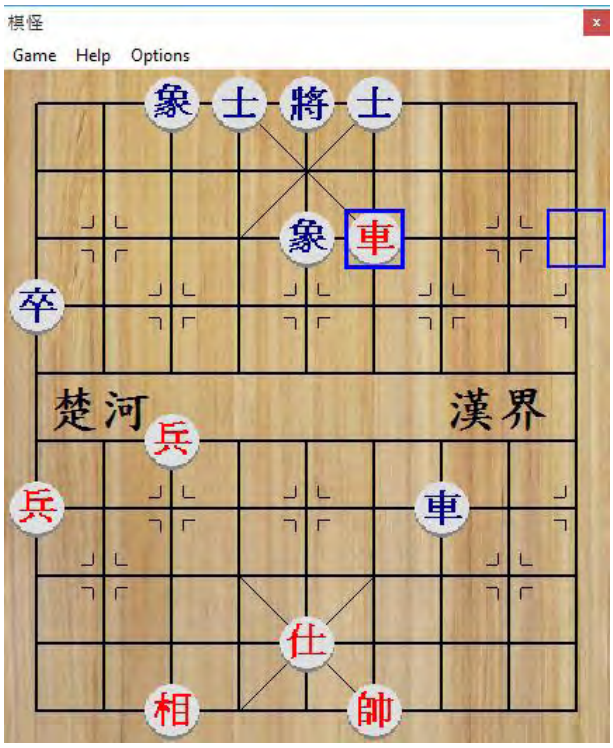


圖 3-4-1 盤面熵編碼範例

以圖 3-4-1 為例，我要對這個盤面進行編碼。

首先找出每顆存活的棋子在它們位置上，依據第 2 節機率統計的結果，得到的霍夫曼編碼：

將：0　　帥：101　　黑的兩個士：10 和 01　　黑的兩個象：00 和 10

黑卒：011　　黑車：110000　　紅仕：11　　紅車：011111000

紅的兩個兵：011 和 1110　紅相：00

然後，不在棋盤上的棋子有 19 顆，所以使用它們死亡時的編碼：

紅仕：0001　紅相：110　紅炮兩隻：01　紅馬兩隻：00　紅車：00

紅兵三隻：00　黑馬兩隻：00　黑炮兩隻：01　黑車：00

黑卒四隻：00

接下來，再把所有 32 顆棋子的編碼按照順序連接起來，再加上目前輪到黑方走的 1，得到這個盤面的編碼：

10111000100110000001111100000010101111100000 00010010010000011000000010101100000000 1

使用 83 個位元，即可記錄圖 3-4-1 中每顆棋子的位置。

四、　動態棋譜

(一)　動態資訊量

夏農熵證明了，要表示任何 N 筆資料，所使用的平均位元數會小於或等於他們的夏農熵。但真的是如此嗎？

若集合{A(N)}表示樹狀資料中節點 N 的所有祖先，{C(N)}表示節點 N 的所有子節點，P(N)表示{C(N 的父節點)}中，N 所出現的機率，則樹狀資料 T 中的一筆資料$\mathrm{T}(n)$的平均動態資訊量$\mathrm{Q}(\mathrm{T}(n))$為：

$$\forall \mathrm{T}(n) \in \mathrm{T} \wedge \mathrm{B}(m) \in \left\{\mathrm{C}(N) | N \in \left\{\mathrm{A}\left(\mathrm{T}(n)\right)\right\}\right\},$$

$$\mathrm{Q}\left(\mathrm{T}(n)\right) = \sum -\mathrm{P}(\mathrm{B}(m)) \cdot \log_2(\mathrm{P}(\mathrm{B}(m)))$$

而當 $Q(T(n))$小於 $T(n)$的資訊熵時，可以使用一種特殊的動態編碼方式來為 $T(n)$編碼。一般編碼方式是利用一筆資料的資訊熵做編碼，而動態編碼法則式根據一筆資料的動態資訊量來編碼。如下圖：

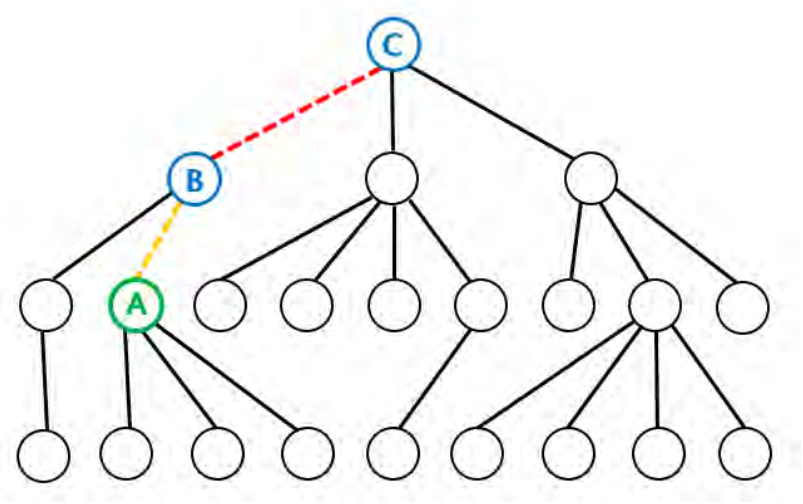


圖 4-1-1 動態資訊量

若要為圖 4-1-1 中的節點 A 編碼，有兩種方式：

1. 根據整棵樹的資訊熵編碼。

   若圖中整棵樹的平均資訊量為 4.32Bits，則使用霍夫曼編碼來表示節點 A 就需要大約 4~5Bits。

2. 根據節點 A 的動態資訊量編碼。

   節點 A 有 B 和 C 兩個祖先。首先，我們必須根據根節點 C 所有子節點的資訊量總和來為紅線編碼，接著根據 B 所有子節點的資訊量來為黃線編碼。將紅線與黃線的編碼接在一起，就是節點 A 的動態資訊碼。

(二) 動態資訊量的應用

在中國象棋的開局中，因為動過的棋很少，而且存活的棋很多，如果使用一般的表示法來表示，會浪費很多空間。因此，我們可以應用動態資訊量使用「動態棋譜」的方法來記錄盤面。

動態棋譜的原理：

一個象棋局面的變化量非常大(有約10^{48}種變化)。但是在開局的時候，每一個盤面大約只有 30 到 40 個分支。以動態資訊量的角度來看：

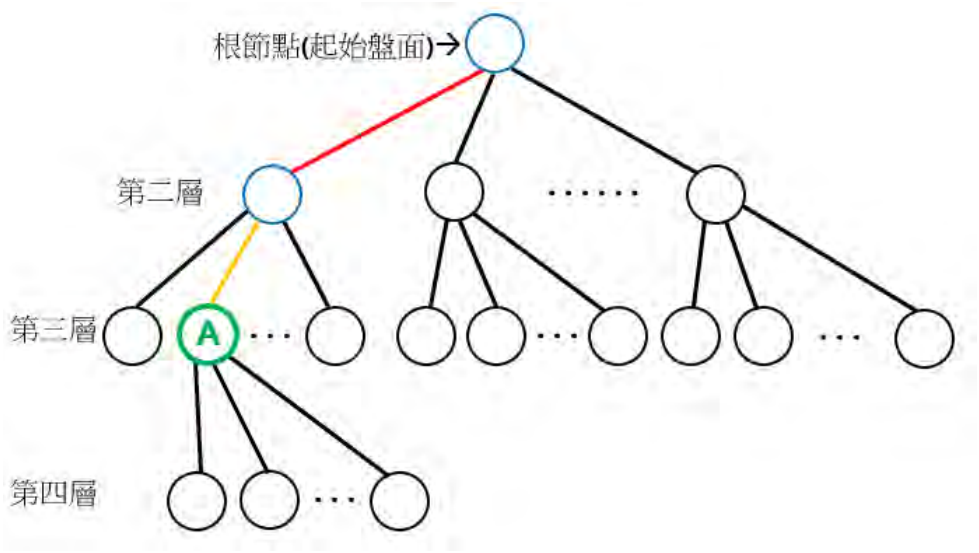


圖 4-2-1 動態資訊量

同樣的，我們要為盤面 A 編碼。編碼方式也分為兩種：

1. 根據中國象棋整棵搜尋樹的資訊熵編碼，大約需要 109 個位元。
2. 根據中國象棋的動態資訊量進行編碼。

   從動態資訊量的公式可推得：

   $$\forall \mathrm{T}(n) \in \mathrm{T} \wedge \mathrm{B}(m) \in \left\{\mathrm{C}(N) | N \in \left\{\mathrm{A}\left(\mathrm{T}(n)\right)\right\}\right\} \wedge \mathrm{D}(m) \in \{\mathrm{C}(Root)\},$$

   $$\mathrm{Q}\left(\mathrm{T}(n)\right) = \sum -\mathrm{P}(\mathrm{B}(m)) \cdot \log_2(\mathrm{P}(\mathrm{B}(m)))$$

   $$\approx \left(\mathrm{D}\left(\mathrm{T}(n)\right) - 1\right) \cdot \sum -\mathrm{P}(\mathrm{D}(m)) \cdot \log_2(\mathrm{P}(\mathrm{D}(m)))$$

   $$\approx \left(\mathrm{D}\left(\mathrm{T}(n)\right) - 1\right) \cdot \log_2 i$$

   其中 Root 表示根節點，D(N)表示節點 N 所在的層數，i 表示分支因子(在象棋中約為 35)。

   因此，利用$(\mathrm{D}(\mathrm{A}) - 1) \cdot \log_2 i \approx 10$個位元就可以表示圖中的節點 A。

當一個象棋盤面的動態資訊量小於它的資訊熵時，使用動態編碼法會使用較少空間。

(三)　動態棋譜的實作

在中國象棋的開局裡，一個盤面所經過的步數(層數)還很少，所以它的動態資訊量會小於它的資訊熵。根據動態資訊量來表示象棋盤面的方法，我們可以稱為動態棋譜。動態棋譜的紀錄方法：

1. 我們必須把一個棋局中的每一步編碼。

   編碼方式：

   (1) 將正在移動的棋子編碼。

   方法：將所有正在走的那一方的棋子，按照所在位置的編號排序，則正在移動的棋子排序後的索引值就是這顆棋子的代碼。

   把棋子代碼轉成二進位時，為了不使程式混淆，代碼長度一律設為 $\lceil \log_2(\text{移動方的棋子總數}) \rceil$。($\lceil x \rceil$表示不小於 x 的最小整數。)

   例如：輪黑方走，而黑方有 13 顆存活的棋，則無論動哪一顆棋，編碼後的長度一定是 4。

   (2) 將棋子所要移動的那一步編碼。

   方法：將正在移動的棋子所有合法的著法列出來，然後按照移動後所在位置的編號排序，則要移動的那一步排序後的索引值就是這一步的代碼。

   將代碼轉成二進位時，為了不使程式混淆，代碼長度一律設為 $\lceil \log_2(\text{移動中的棋子的合法著法總數}) \rceil$。

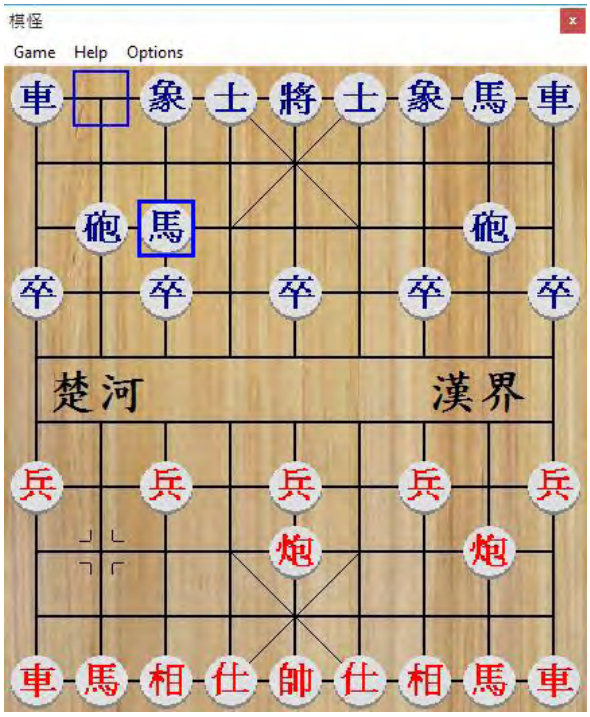
   例如：輪黑車走，而黑車有 7 種合法的著法，則無論動哪一步，編碼後的長度一定是 3。

(3) 將棋子代碼和著法代碼連接在一起，即是這一步的代碼。

2. 把每一步的代碼連接起來，形成一個整數。

3. 這個整數就是動態棋譜。

4. 舉例：



(1) 炮八平五

(2) 馬二進三

圖 4-3-1 動態棋譜範例

第一步：炮八平五

因為目前輪紅方走，而紅方有 16 顆棋，所以炮的代碼長度為

$\lceil \log_2(16) \rceil = 4$。

將所有棋子排序，炮排第四，所以炮的代碼是 0100。

炮目前有 12 種合法著法，所以著法的代碼長度為$\lceil \log_2(12) \rceil = 4$。

將炮所有的著法排序，炮八平五排第十，所以這個著法的代碼是 1010。

所以，「炮八平五」的位元值是 01001010。

第二步：馬二進三

因為目前輪黑方走，而黑方有 16 顆棋，所以馬的代碼長度為

$\lceil \log_2(16) \rceil = 4$。

將所有棋子排序，馬排第四，所以馬的代碼是 0100。

馬目前有 2 種合法著法，所以這個著法的代碼長度為$\lceil \log_2(2) \rceil = 1$。

將馬所有的著法排序，馬二進三排第二，所以這個著法的代碼是 0。

所以，「馬二進三」的位元值是 01000。

將第一步和第二步的代碼連接起來，得到 0100101001000，也就是這個盤面的動態棋譜值。

動態棋譜的動態資訊量：

在每一個盤面下，平均有 35 個合法的著法。在可以移動的棋子中，有越多種動法的，被移動的機率就越大。假設一個棋子的合法著法數量與它被移動的機率成正比，那麼移動一步的平均資訊量是：

$$Q = -\sum_{k=1}^{\text{合法著法數}} P_k \cdot \log_2 P_k$$

$$= -\sum_{k=1}^{\text{總棋子數}} \left(\sum_{n=1}^{\text{棋子(k)的合法著法數}} P_n \cdot \log_2 P_n \right)$$

設一顆棋子的每個合法著法被執行的機率(即P_n)皆相等，且$m =$ 棋子(k)的合法著法數，則：

$$\because m \propto \sum_{k=1}^{m} P_k \text{ ，} P_a = P_b (a, b \in \mathbb{N}\text{，a} < \text{b} \leq \text{m})$$

$\Rightarrow$ 對於所有$a \in \mathbb{N}$，a $\leq$ 總棋子數，恆有$P_a = \dfrac{1}{\text{總棋子數}}$

$$\therefore -\sum_{k=1}^{\text{總棋子數}} (m \cdot P_k \cdot \log_2 P_k)$$

$$= -\sum_{k=1}^{\text{總合法著法數}} P_k \cdot \log_2 P_k$$

$$= -\log_2(P_{\text{總合法著法數}}) = \log_2(\text{總合法著法數})$$

由以上計算可知，每一層平均需要 4～6 個位元來表示。

動態棋譜的特性：

1. 動態棋譜的長度會隨著經過的步數變多而增加。
2. 無法動其中一步或一顆棋的位置做直接的存取或二分搜尋。如果要存取的話，必須將整個棋局展開。
3. 如要增加一步，只需要將那一步的代碼接在後面，不需要全部展開。
4. 如要刪除最後一步，只需要將最後一步的代碼從動態棋譜中移除，也不需要全部展開。

使用範圍：

動態棋譜使用於開局較理想。如果使用於 20 層以上的局面，就可能會浪費空間。

另外，應用在開局庫也是很好的方法，因為一個棋局包含了數十個局面，還有每一個局面的對應手。

五、 棋怪的製作過程

(一) 圖形介面

象棋程式需要一個圖形介面，才能讓人類使用。我必須畫出棋盤和棋盤上的所有棋子，並把它們移動到他們正確的位置。棋怪的棋子與移動棋子的模組是由 Planet-Source-Code 的 Simple Chinese Chess 作者 Kaki Cheung 所提供的。

我將每顆棋子一開始在棋盤上的正確位置儲存在棋子的 Tag 屬性中。當程式開始時，它會把所有的棋子移動到它們正確的位置，並記錄下每顆棋子的位置。這個程式如下：

```
Dim tmp As Integer, i As Integer
For i = 0 To 31
        tmp = Right(imgPieces(i).Tag, 2)
        imgPieces(i).Move imgBoard(tmp).Left + 60, imgBoard(tmp).Top + 60
        imgPieces(i).DragMode = vbAutomatic
        imgPieces(i).Visible = True
Next
```

程式 5-1-1 初始化棋子

程式執行後的結果：

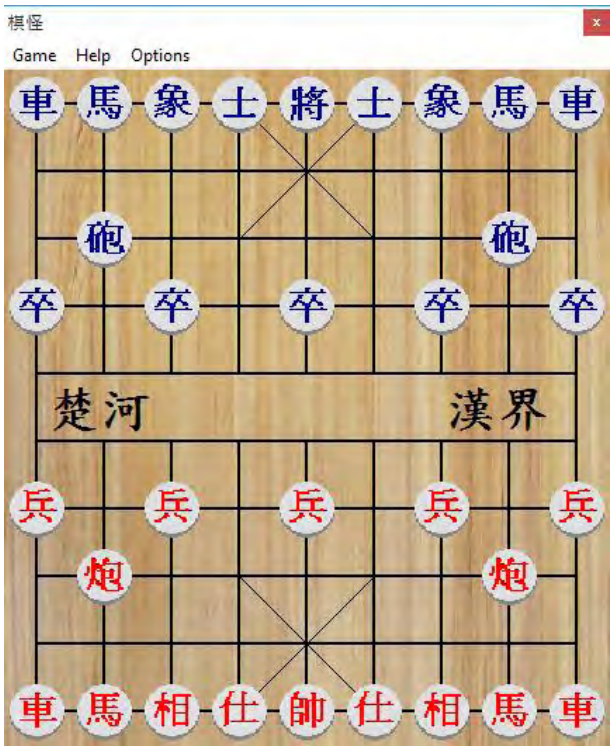


圖 5-1-1 棋怪的棋盤與棋子

(二) 移動棋子

只有棋盤、棋子還不夠，還要能移動棋子和吃子。

移動棋子的方式是拖曳（Drag）。當拖曳（DragDrop）到空白的位置上時，就把它移到那個格子。如果那個位置有其他棋子，就把那顆棋子吃掉。移動棋子可以用 Move 陳述式，被吃的棋子就把 Visible 屬性設成 False。

另外，棋子移動之後，在移動的棋子和它原本的位置周圍，會有一個藍色的框出現，讓使用者能夠清楚知道剛才被移動的是哪一顆棋。

移動棋子的程式如下：

```
Private Sub imgBoard_DragDrop(Index As Integer, Source As Control, _
X As Single, Y As Single)   棋子移到格子上
    Selector(1).Move Source.Left + 60, Source.Top + 60
    Source.Move imgBoard(Index).Left + 60, imgBoard(Index).Top + 60
    Selector(0).Move Source.Left + 60, Source.Top + 60
End Sub
```

程式 5-2-1 移動棋子

程式執行後的結果：

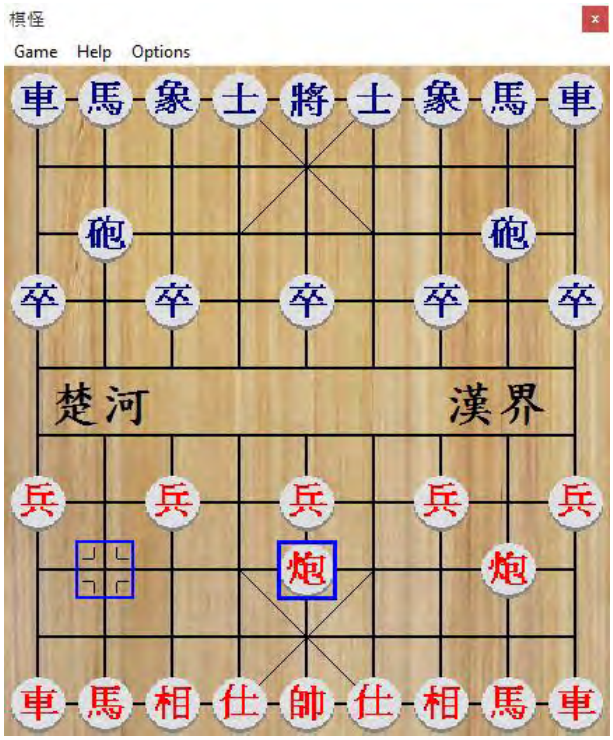


圖 5-2-1 移動棋子

(三) 合法移動棋子

象棋中，每顆棋子都有它移動的規則。如果程式允許所有的著法，那麼它絕對不能執行。

因此，我們必須判斷一個著法是否合理。如果是合法的著法，才允許動這步。除此之外，程式還要能判斷將軍和將死。如果輪被將軍的一方走，他就必須要設法解除將軍。假如某方沒有任何合法的著法可以走，那就是被將死了。

檢查將軍的方法：程式先產生所有的著法，並且一一執行和還原每個著法，然後看要檢查的一方的帥（將）是否還活著。程式如下：

```
Function IsCheck(Side As Integer) As Boolean
    Dim M As Move
    GenerateMoves
    For Each M In LegalMoves
        Call MakeMove(M)
        If 將軍死亡 Then
            Call UndoMove(M)
            IsCheck = True
            Exit Function
        End If
        Call UndoMove(M)
    Next
    IsCheck = False
End Function
```

程式 5-3-1 檢查將軍

但是，這個方法的速度實在太慢了。在對弈時，程式每秒必須檢測上千至上萬個盤面。如果使用這個方法，會嚴重影響計算的速度。

於是，我想到了另一個方法：與其執行每個合法的著法，不如檢查對方的棋吃掉我方的帥是否合法。另外，對方的士和象也不可能吃到我方的帥。

所以，檢查將軍的程式變成：

```
Function IsCheck(Side As Integer) As Boolean
    IsCheck = True
    For 對方的每一顆棋(不包含士、象)
        If 這顆棋吃掉我方的帥合法 Then Exit Function
    Next
    IsCheck = False
End Function
```

程式 5-3-2 檢查將軍改良版

這個方法的確大幅增加了計算的速度。對於判斷勝負以及後面的人工智慧部分，使用這個方法將會有很大的幫助。

(四) 人工智慧的引擎

棋怪已經可以讓兩個人對弈，但是還沒有自己走棋的能力。於是，我幫它加上了 Alpha-Beta 和寧靜搜尋的引擎，使它有與人類對戰的能力。

搜索的程式如下：

```
Function AlphaBeta(ByVal Depth As Integer, ByVal alpha As Long, _
ByVal beta As Long) As Long
    If Depth = 0 Then
        AlphaBeta = 寧靜搜索
        Exit Function
    End If
    If 置換表有目前盤面 Then
        AlphaBeta = 置換表分數
        Exit Function
    End If
    Dim Value As Long, Best As Long
    Best = -Infinity
    GenerateMoves
    For Each 合法的著法
        執行這個著法
        Value = -AlphaBeta(Depth - 1, -beta, -alpha)
        還原這個著法
        If Value > Best Then Best = Value
        If Value > alpha Then alpha = Value
        If Value > beta Then 裁剪
    Next
    把目前盤面加到置換表
    AlphaBeta = IIf(Best = -Infinity, 到根節點的距離 - Best, Best)
End Function
```

程式 5-4-1 中局搜索

不過，它還沒有開局庫，所以開局的時候容易走出不好的著法。它同時也證明了，搜索程式在開局時不穩定性較高。

(五) 開局庫的建立

棋怪是一個以開局庫為主的象棋程式，所以它必須有一個強大的開局庫。為了建立開局庫，我想到了幾個方法：

1. 人工輸入。但是，如果輸入一局棋需要一分鐘的時間，那輸入一萬局棋就要連續七天才能完成。所以，這個方法不能用來建立巨大的開局庫。
2. 讓程式自動展開。這是一個不錯的方法，因為程式的速度非常快。這個方法的基本思想是這樣的[10]：
   (1) 假設在某個盤面 N 下，輪到紅走時，有 a 到 z 這 26 個合法的著法。但是，程式並不知道哪一個是最好的，所以先挑 a 走。
   (2) 走完 a 之後，就讓程式與自己對戰，看最後結果如何。如果是紅勝，那 a 的分數就+1；如果是黑勝，那 a 的分數就-1；如果是和局，那 a 的分數不變。
   (3) 將局面還原成盤面 N，然後依次走 b、c、d…，直到 z 為止，並計算每一步的分數。
   (4) 重複這些步驟幾次，發現的 t 的分數最高，所以把它加進開局庫。
3. 蒐集大量的古譜或歷屆比賽的棋譜並分析，然後加到開局庫中。

後來我發現，第 3 種方法能在最短的時間內建立最大的開局庫，所以使用第三種方法。

然後，我必須選擇要用哪一種方法來將局面編碼。因為開局庫內的局面經過的步數都不是很多，而且為了更容易增加、減少以及檢查資料，所以使用動態棋譜較為理想。

動態棋譜開局庫的建立方法很簡單，就是將收集的棋譜編碼成動態棋譜的格式，放入開局庫時使用換行 CR+LF 隔開。而為了能夠使用二分搜尋，所有的資料必須經過排序。例如：

```
00110001101111100001111100011001100001110011101 1
01100011100011100011011000011100000111010000010000010 0
01100011100011100011110010011100110001000111 1
11100100111011110110111 11
```

表 5-5-1 開局庫範例

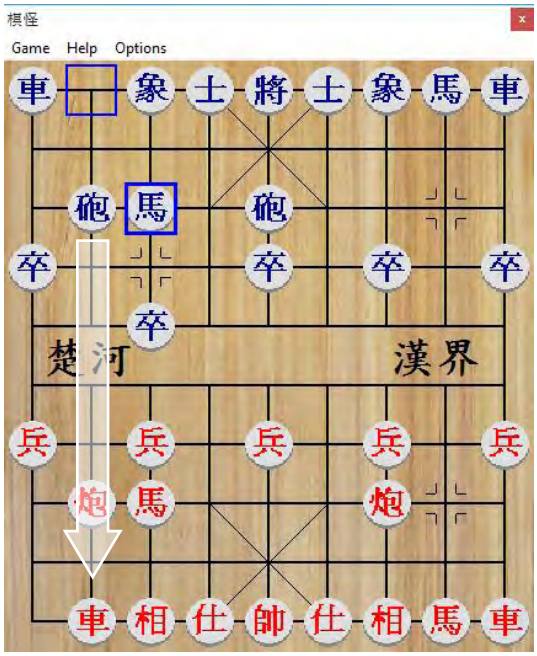
(六) 棋譜收集與分析

為了建立開局庫，我收集了大約 20 萬局的對局棋譜，並使用象棋橋將全部轉成 PGN 格式。接下來，我在棋怪中加入讀取 PGN 棋譜的功能。

然後，程式就把所有的棋譜放入開局庫。於是，一個包含 20 萬局棋的開局庫大功告成了。

(七) 開局庫的改良

完成開局庫之後，我和程式對戰，發現開局庫的資料非常完整。不過，它有時會走出奇怪的著法。例如：



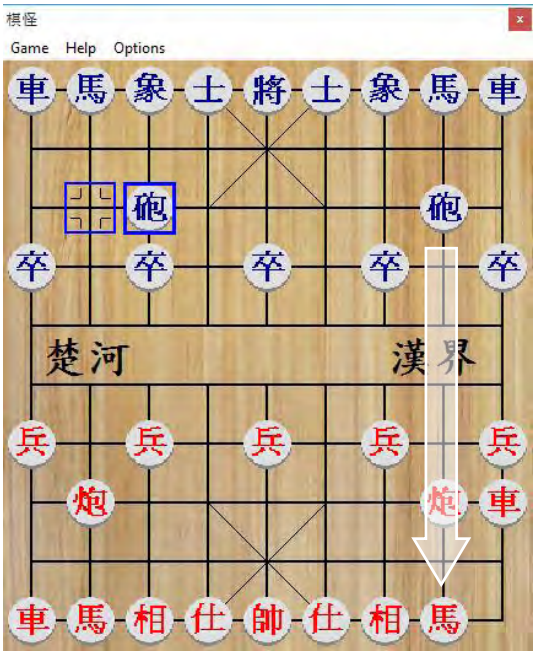
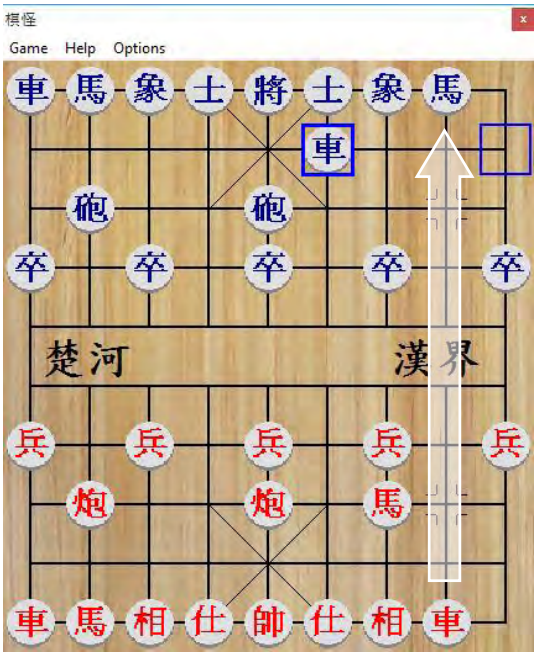


圖 5-7-1、5-7-2 不好的開局著法

還有：



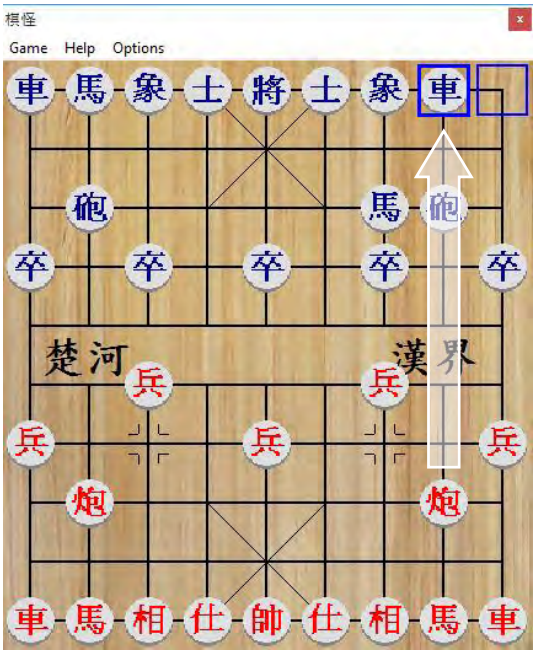


圖 5-7-3、5-7-4 劣著

依照我的分析與判斷，造成這個情況的原因可能有：

1. 程式分析棋譜的部分有問題。
2. 程式將棋譜編碼的部分有問題。
3. 程式將開局庫解碼的部分有問題。
4. 棋譜裡面隱藏著一些不好的著法。

對於每一種可能的原因，我想了以下的檢查與解決方法：

1. 程式分析棋譜的部分有問題。

   檢查：讓棋怪、象棋巫師與象棋橋分別讀入同一個棋譜，並檢查最終局面是否相同。

   結果：使用 100 個棋譜作檢查，發現有 4 個出錯，共通點為皆有這一類的註解：

37. 馬六進四 {將軍。} 37. 將５平６ 38. 馬七進五 車六退一

表 5-7-1 使程式出錯的註解

解決方法：使用象棋橋刪除所有註解。

2. 程式將棋譜編碼的部分有問題。

檢查:由於沒有其他象棋軟體支援動態棋譜的格式,所以使用人工檢查。

結果：觀察執行結果，沒有任何錯誤發生。

3. 程式將開局庫解碼的部分有問題。

檢查：將棋譜編碼之後再解開，看是否與原本棋譜相同。

結果：檢查的數十個棋譜中，皆沒有發生錯誤。

4. 棋譜裡面隱藏著一些不好的著法。

檢查：使用象棋橋將所有棋譜合併，再用人工方式檢查是否有劣著。

結果：在某些較冷門的開局(如金鉤炮局)中，找到了許多不好的著法。

解決方式:使用局面評估函數,找出並過濾掉開局庫中明顯不好的著法。

但是，由於棋譜數量過多，所以使用評估函數可能不太理想。

後來,我發現不好的棋譜的來源都是快棋,因此將快棋的部分刪除並重新編碼。

(八) 開局庫改良的結果

棋怪的開局庫已經完成了，而且也有了中局的程式。因此，它已經可以與人類或其他象棋軟體對戰。

我拿棋怪與象棋巫師對戰，發現在開局庫的部分平均晚 3 手才脫譜。由此可見，開局庫改良的效果很好。

參、 研究結果與討論

一、 棋盤表示法的分析

有許多方法可以表示一個象棋棋盤，而它們都各有優缺點。可以用來表示一個象棋盤面的方法，主要有以下幾種：

種類	使用空間	優點	缺點
傳統表示法	361 位元	• 編碼非常簡單	• 使用的空間大
FEN 表示法	最好情況：33 位元組(264 位元) 最壞情況：大於 600 位元	• 編碼、解碼非常簡單 • 人類閱讀方便	• 使用的空間大 • 長度不一
Zobrist 鍵值	64 位元	• 使用的空間小 • 運算快速	• 亂數鍵值會碰撞 • 無法倒推回原本局面
利用資訊熵的表示法*	平均約 109 位元	• 使用的空間相對較小	• 長度不一
PGN 棋譜	非常大，通常可達 1KB (8000 位元)	• 編碼、解碼非常簡單	• 使用的空間大 • 長度不一
動態棋譜*	約為總經過步數乘以 5.2	• 開局時使用空間非常小	• 長度不一 • 殘局時使用空間非常大

表 1 常見盤面表示法的比較(註：標有*為自製的方法)

二、 適用於不同情況的棋盤表示法

每一種表示法都有優缺點，因此都適用在不同的地方。我分析了 6 種表示法的優點，發現：

種類	適用於	原因
傳統表示法	無	使用的空間過大，也不易閱讀。
FEN 表示法	局面分析與輸出	空間大，但易於人類閱讀。
Zobrist 鍵值	置換表	空間小，而且計算快速。雖然有碰撞的可能，但是機率非常低。
利用資訊熵的表示法*	局面分析、輸出與傳送，開局庫	使用的空間小，但不易於人類閱讀。
PGN 棋譜	棋譜分析與輸出	空間大，但人容易閱讀。
動態棋譜*	開局庫	開局時，使用的空間很小。

表 2 常見盤面表示法的分析(註：標有*為自製的方法)

後來我也發現，將同樣的棋譜放到開局庫中，使用動態棋譜能省下較多空間，所以開局庫使用動態棋譜編碼較佳。

肆、 結論與應用

一、 應用於開局庫的新盤面表示法

動態棋譜是一種應用於開局庫的盤面表示法。這種方法不僅大幅縮小了表示一局棋所需的空間，也增加了開局庫的搜索效率。

傳統的開局庫在維護時，必須將每一筆資料展開做檢查。但是使用動態棋譜的開局庫中，只要找到一個不好的著法，就可以利用 Left 函式來檢查並刪掉開局庫中的這個著法。例如：

(一) 假設在 01001010010001001 這個局面下，走 0100 是不好的。

(二) 檢查開局庫的每一筆資料中，是否以" 0100101001000010010100"開頭。

(三) 如果是的話，就把那一筆資料刪除。

(四) 全部檢查完之後，這個開局庫內就沒有這一步了。

而此開局庫還有另一個優點。傳統的開局庫(以象棋巫師為例)無法直接把開局庫內的棋譜取出，但是如果是使用動態棋譜，就可以直接把用來製作開局庫的棋譜拿出來。

二、 未來展望

使用動態棋譜的開局庫雖然擁有許多優點，但是還是有可以改進的地方。此外，動態與靜態的編碼是否可以合併？按照目前的方式是不可行的，因為每個棋譜後面必須加上每顆棋子被動的機率。但是如果可以的話，那就能省下更多的空間。

不過，我希望動態棋譜不只能減少空間，還能為象棋的盤面表示法開拓一個新的領域。不僅如此，除了中國象棋以外，動態棋譜也能應用在西洋棋、黑白棋等許多棋類上。如果能夠基於動態棋譜來研究新的盤面表示方法，那我相信一定會有更好的結果。

伍、 參考文獻

[1] 方裕欽，2008，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

[2] 甘崇緯，2011，適用於象棋開局庫之工作層級極小極大化搜尋，交通大學多媒體工程研究所碩士論文

[3] 白聖秋，2006，DTS 演算法效能改良之研究，臺灣師範大學資訊工程研究所碩士論文

[4] 李任軒，2005，電腦象棋知識庫的切捨技術，臺灣師範大學資訊工程研究所碩士論文

[5] 林子哲，2007，「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

[6] 林玉祥，2006，電腦圍棋中考慮使用損劫之打劫策略研究，臺灣師範大學資訊工程研究所碩士論文

[7] 林伯翰，2013，應用於象棋開局庫之工作層級 AB-DUAL*搜尋演算法，國立中央大學資訊工程學系碩博士論文

[8] 林順喜，2006，叢集計算於電腦象棋的應用，行政院國家科學委員會專題研究計畫

[9] 高暐倫，2009，電腦象棋審局評分自動調整系統，國立交通大學資訊科學與工程研究所碩士論文

[10] 張修正，2008，電腦象棋開局庫之改進研究，國立臺灣師範大學資訊工程學系碩士論文

[11] 許俊彬，2006，象棋棋形辨識之研究，國立交通大學資訊科學與工程研究所碩士論文

[12] 郭哲宇，2007，電腦象棋擴大空步裁剪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

[13] 陳志昌，2005，電腦象棋知識庫系統之研製，國立臺灣大學資訊工程學研究所博士論文

[14] 陳俊佑，2009，八層及九層三角殺棋的勝負問題之改進與研究，國立臺灣師範大學資訊工程研究所碩士論文

[15] 陳俊嶧，2010，一個蒙地卡羅之電腦圍棋程式之設計，國立交通大學資訊科學與工程研究所碩士論文

[16] 陳家齊，2004，通用棋譜編製系統之研究，國立交通大學資訊科學與工程研究所碩士論文

[17] 陳漢鴻，2006，電腦象棋的自我學習，國立雲林科技大學資訊工程研究所碩士論文

[18] 勞永祥，2011，電腦暗棋之人工智慧改良，國立臺灣師範大學資訊工程研究所碩士論文

[19] 曾汶傑，2008，象棋殘局庫之研究，國立交通大學資訊科學與工程研究所碩士論文

[20] 黃文樟，2006，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

[21] 詹傑淳，2010，電腦圍棋打劫的最佳策略之研究，國立臺灣師範大學資訊工程研究所碩士論文

[22] 楊泰寧，2011，電腦圍棋樣式處理之研究，行政院國家科學委員會專題研究計畫

[23] 蔡數真，2010，電腦六子棋開局庫系統之設計與研製，國立臺灣大學應用數學研究所碩士論文

[24] 賴隆億，2012，電腦象棋棋譜分析之研究，國立交通大學資訊學院資訊學程碩士論文

[25] 謝政孝，2010，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

[26] 謝曜安，2008，電腦暗棋之設計及實作，國立臺灣師範大學資訊工程研究所碩士論文

[27] C. E. Shannon, 1948, A Mathematical Theory of Communication

[28] C. E. Shannon, 1949, Programming a Computer for Playing Chess

[29] 象棋百科全書 http://www.xqbase.com/index.htm

【評語】190010

運用熵的觀念來做象棋開局的編碼，並利用動態結構來做編碼的最佳化。可以節省編碼所需的記憶空間。但編碼的深度，受限於編碼的執行時間，其實用度仍有改善的空間。