師大學報：數理與科技類
民國88年，44(1)，43-61

利用電腦探討中國古代益智遊戲—「華容道」之解法

魏仲良　林順喜

國立台灣師範大學資訊教育系

在本文中，我們嘗試設計演算法，利用電腦找出中國古代流傳下來的益智遊戲—「華容道」的最少步數，以驗證前人資料上所記載各盤面的最少步數是否正確。此遊戲中許多盤面之解答的移動步數超過100步，因此不能直接用暴力法搜尋，目前文獻上尚未見到電腦之解法，只有一些人爲的解答有記錄，也有一些程式將這些人爲的、不是最佳的解答直接記錄下來作展示。因此我們構思如何解決此困難之問題。在此論文中，我們發展了一些技術，目標是求出完全的最佳解，並實際撰寫程式測試，要求在可忍受的時間內解出。程式的執行結果與先前得到的前人資料有所出入，有些與資料記載的吻合，有的則較記錄爲多，還有一些比資料上的少上三至五步之多。驗證了一下程式輸出到檔案的最佳解，發現程式所求得比資料記載還要少的結果應是正確的。至於程式求得較前人資料爲多的部份，可能是前人的文獻資料有誤，因爲資料上只記載著各盤面最少步數的解題記錄，並無參考的解法。

關鍵詞：遊戲樹、二元搜尋樹、暴力法

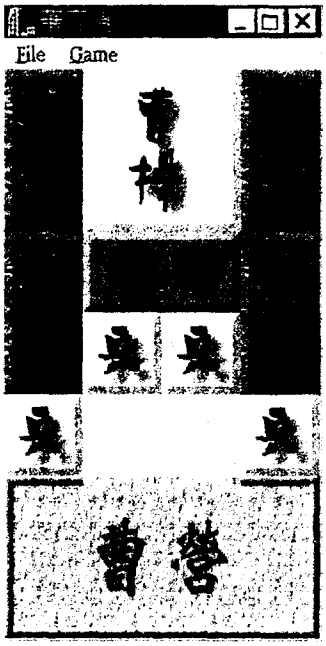
緒論

一、華容道遊戲的介紹：

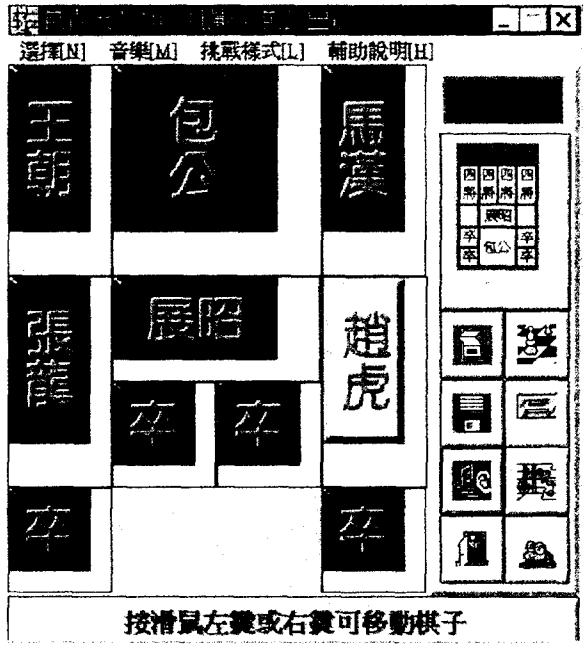
華容道，這是從中國古代就流傳下來的智慧遊戲，出自三國演義中「關羽橫讓捉放曹操」的情節。由於盤面變化多端，曾被譽爲「智力遊戲界的三大不可思議」的遊戲之一。其玩法爲在所有棋子均不離開棋盤的前提之下，如何使曹操越過重重的包圍，逃回曹營。這個遊戲可在下列網站均可找到程式下載來玩（用人工嘗試錯誤法來玩）：

http://www.tces.chc.edu.tw/center/school1.htm
http://www.sivs.chc.edu.tw/goodware/13.htm
http://www.nchu.edu.tw/~tnes/13-3.htm
http://content.edu.tw/primary/music/tp_ck
/softgood-03.htm

華容道基本上其棋盤爲橫四格縱五格的盤面，曹操爲占四格的2x2的正方形棋子；占兩格的爲五虎將的關羽、張飛、趙雲、馬超與黃忠，其中只有關羽橫兩格，其餘四將均爲豎兩格；還有四個各占一格的兵（如圖一）。所以在遊戲的時候，只能利用盤面上留下的兩個空格來移動棋子，只要想辦法將曹操移到鄰接曹營上方的正中央的位置，即表示我們已成功讓曹操逃回曹營，遊戲至此於是結束。移動的步數愈少愈好。而在移動的過程中我們可以發現，關羽跟曹操不能在同一排，關羽必須要橫讓，曹操才有可能通過向下走，這正好符合了三國演義中的情節，所以這項遊戲又被稱爲「捉放曹」。除了上述標準的盤面之外，橫擺的五虎將除了關羽之外，也可以將其他四將橫擺，使得盤面有更多變化，更富挑戰性。



圖一 華容道—橫刀立馬



圖二 包青天—包公出巡

在本文中，盤面資料來源爲「阿集好用軟體區之休閒益智軟體 (http://www.ntctcps.tc.edu.tw/13-3.htm) 內的「華容道益智遊戲中文版（視窗 95 版）」。除了華容道之外，還有兩個類似的棋戲：「包青天」（圖二）與「箱入娘」。其中，包青天亦可在「阿集好用軟體區之休閒益智軟體」內找到。這些軟體都只是純粹爲了遊戲之用，並無參考的解答。另外，網路上還可以找到一個 DOS 下的華容道遊戲軟體（ 604.zip ），它有提供參考的解答，但大部份盤面所解的步數比我們多，所以並不是最佳解（請見「程式執行結果與分析」中說明）。上面提到的這些程式，已經收集整理在 ftp://alg.ice.ntnu.edu.tw/pub/chess/ ，有興趣的讀者，可以上網去取得。

二、以人腦移動棋子與以電腦求解的不同：

一般人在玩這類的遊戲的時候，大概會很直覺地看到哪顆棋子可以移動，就移動那顆棋子。等到發現錯誤的時候，再將棋子擺回原位。但是，當移動的步數一多，大概很少人能記得如何從一開始的盤面一步步移動棋子，走到目前的狀態。而且在移動棋子的過程中，很可能出現重覆的盤面而不自知。

利用電腦求解，可以記錄下哪些盤面已經走過，避免浪費時間在相同搜尋重覆的路徑；除此之外，可以依循一定的規則來展開每一個盤面中所有能移動的方法。如此，窮舉所有的可能，只要給定的盤面有解，一定能找到解，剩下的只是求解需要時間的長短罷了。但因爲解答的步數有過 100 步以上的，所以不能直接以暴力法 (Neapolitan 等， 1996 ； Hororitz 等， 1994) 去窮舉。以下我們說明如何去克服此難題。

電腦解題之演算法

在正式開始寫程式求解之前，我們先利用目前已知的資料。華容道的棋盤爲橫四格縱五格，爲了辨別棋子在棋盤上的位置，先對棋盤上的每個格子編號：左上角爲 0，右下角爲 19，共二十個格子（表一）。爲了識別每顆棋子，我們也將棋子加以編號：曹操爲 1 號，五虎將給予 2 至 6 的編號，剩下四個兵則編爲 7 至 10 號，而以 0 表示空白（表二）。最後，以棋子的長寬分類，共可分爲四類：第一類爲 2x2 大小，第二類爲 2x1，第三類爲 1x2，而第四類則爲 1x1（表三）。

表一　橫盤編號

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19

表二　棋子編號

棋子	編號	棋子	編號
空白	0	黃忠	6
曹操	1	兵(1)	7
關羽	2	兵(2)	8
張飛	3	兵(3)	9
趙雲	4	兵(4)	10
馬超	5		

表三　棋子分類

類別	長x寬
1	2x2
2	2x1
3	1x2
4	1x1

以每顆棋子左上角所佔據的格子編號來表示棋子的位置，如此，每個盤面的狀態則可以用一個陣列表示。例如圖一「橫刀立馬」的盤面，曹操的（編號爲 1）的位置爲 1、關羽（編號爲 2）的位置爲 9，其餘可類推，我們可以表示爲：

[1 9 0 3 8 11 13 14 16 19]

因爲知道十顆棋子的位置就可以推知兩個空格的所在，因此，我們並不需要額外記錄空格的位置。

在每個盤面要尋找下一步該怎麼走，若檢查每顆棋子可以走的路，棋盤上共有十顆棋子，則十顆棋子都要檢查，而且大部份的棋子可能都無法移動。爲了減少麻煩，改由空白格下手。由於盤面上只會有兩個空格，只要檢查空格周圍的棋子是否能往空格移動，如此即可減少許多不必要的判斷，而加快搜尋的速度。爲了推知空格的位置，首先將棋盤上所有的格子都先填入 0，再依棋子的位置以及種類，將棋子的編號填入棋盤內，填完之後，即可得知空格的所在。舉例說明，下面是我們將棋子編號填入「橫刀立馬」盤面之後所得的結果（表四）：

表四

3	1	1	4
3	1	1	4
5	2	2	6
5	7	8	6
9	0	0	10

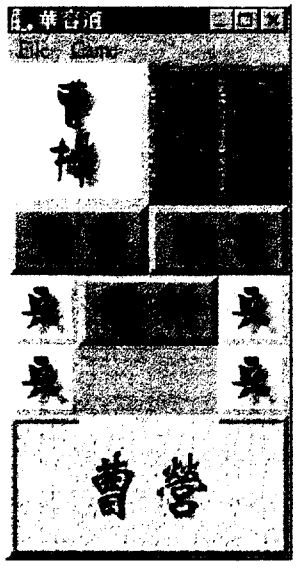
由此可以得知兩個空格的位置分別爲 17 與 18。接著則分別判斷每個空格上下左右四個方向緊鄰的棋子能否往空格移動，若可，更動該棋子的位置後則可得到新的盤面。

由於我們的目的在尋找由給定盤面開始，最終

將曹操移動至曹營上方中央的位置最少需要多少步，因此第一個念頭即是利用 BFS (Breadth-First Search) 配合 Branch and Bound 的方式 (Neapolitan 等，1996；Hororitz 等，1994) 求解，如此，第一個找到的解一定是最佳解。但是，利用 BFS 需要額外的空間來存放等待著要被展開盤面的 Queue，因此，當我們要展開的深度很深時（例如 100 層以上），需要耗費的空間就很驚人了。所以改而採用 DFS (Depth-First Search) 來展開 Game Tree。爲了加快 DFS 的效率，每當找到一個解時，則立即記錄下至目前爲止的最少步數，倘若目前步數超過最少步數則不繼續往下展開；除此之外，另外用 Binary Search Tree 記錄已經走過的盤面（這些盤面有一次序值，如下所述），以及先前走到此盤面時的步數：第一步，先將同一類棋子的位置加以排序，例如圖三「三軍聯防」盤面中，十顆棋子的位置如表五所示：

表五　圖三棋子位置與類別

棋子	位置	類別
曹操	0	1(2x2)
關羽	13	2(2x1)
張飛	8	2(2x1)
趙雲	10	2(2x1)
馬超	2	3(1x2)
黃忠	3	3(1x2)
兵(1)	12	4(1x1)
兵(2)	15	4(1x1)
兵(3)	16	4(1x1)
兵(4)	19	4(1x1)



圖三　三軍聯防

我們將十顆棋子的位置依序存放在一陣列中：

[0 13 8 10 2 3 12 15 16 19]

再依照棋子的類別可以分爲四組：

[0]、[13 8 10]、[2 3]、[12 15 16 19]

這四組分別排序過後再組合可得：

[0 8 10 13 2 3 12 15 16 19]

這樣的目的在於減少重複的盤面。相同類別的棋子，位置互調，其實是一樣的狀況。經過此步驟之後，可過濾掉這些重複狀況的盤面，因而能減少展開的盤面，縮短程式搜尋的時間。

第二步，再透過公式將十個棋子的位置編碼成一個 64-bit 大小可表示的數值。這裡所採用的公式如下：(假設儲存棋子位置的陣列爲 C[])

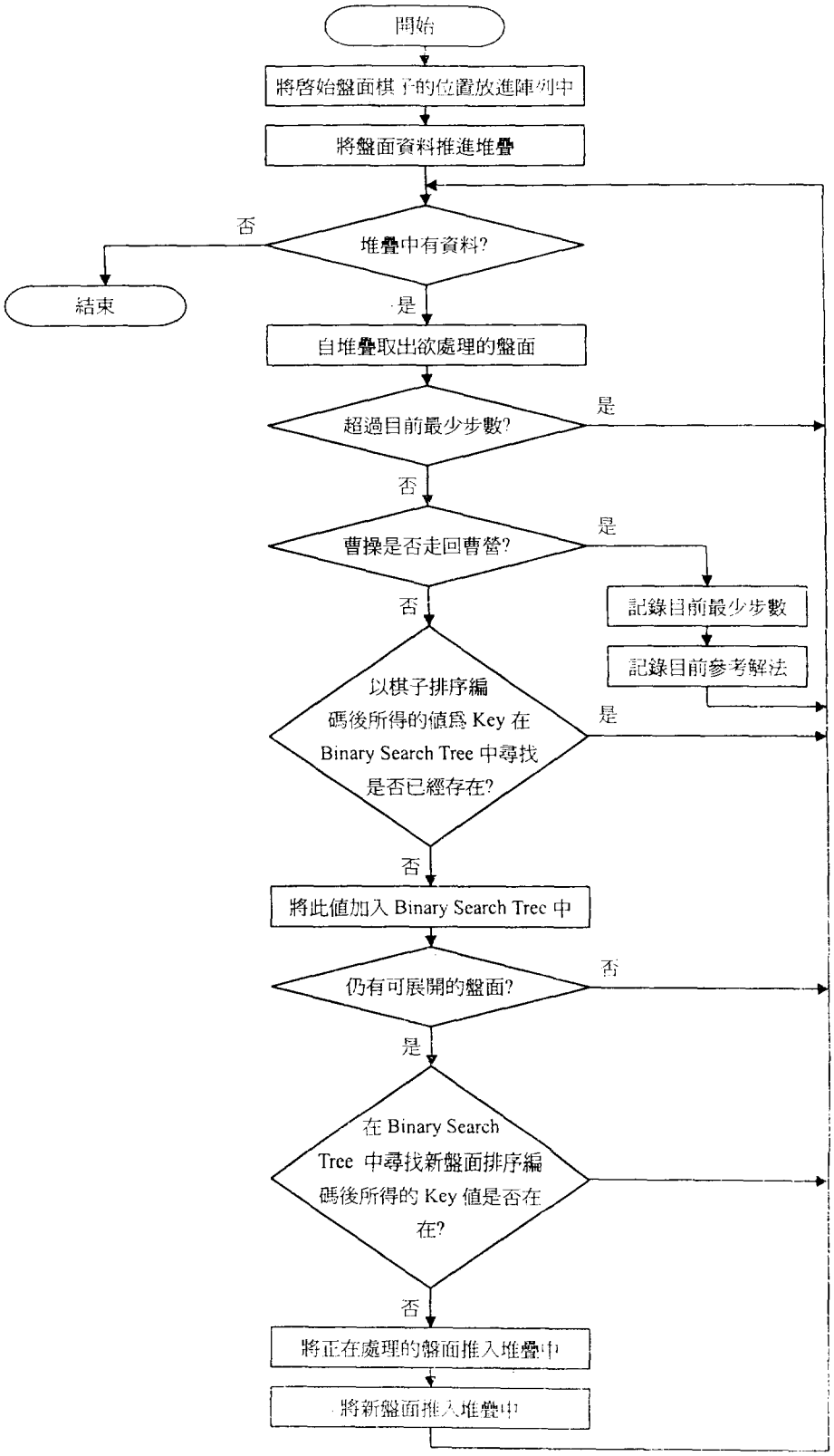
$$\begin{aligned}Value = {} & C[0]*20^9 + C[1]*20^8 + C[2]*20^7 + C[3]*20^6 \\ & + C[4]*20^5 + C[5]*20^4 + C[6]*20^3 + C[7]*20^2 \\ & + C[8]*20 + C[9]\end{aligned}$$

以圖一爲例，我們可將盤面換算成：

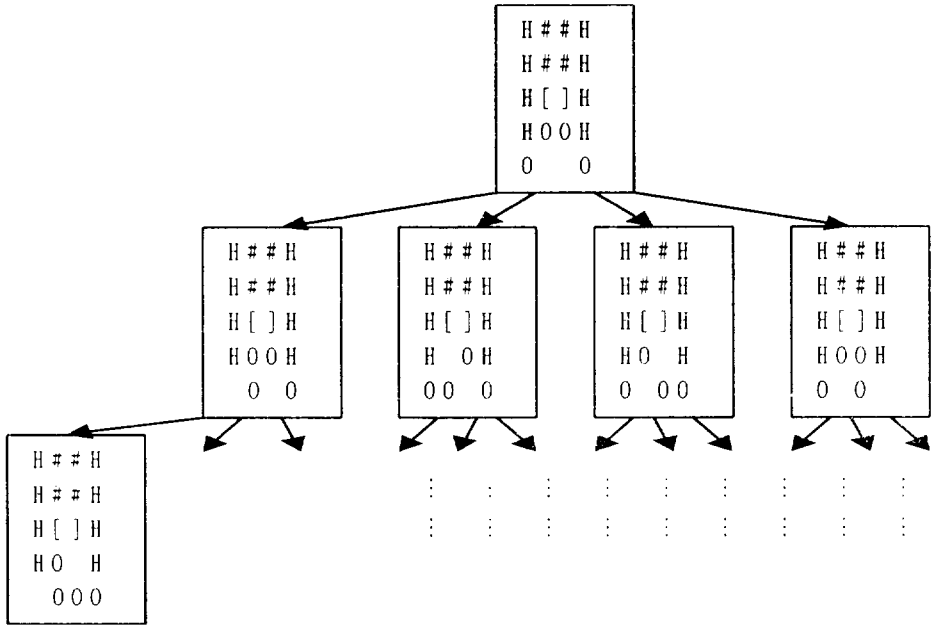
$$\begin{aligned}& 0*20^9 + 8*20^8 + 10*20^7 + 13*20^6 + 2*20^5 + 3*20^4 \\ & + 12*20^3 + 15*20^2 + 16*20 + 19 = 218438982339\end{aligned}$$

這樣有兩個好處：一是可減少空間的使用，另一則爲同一類別的棋子，若在相同的位置上，實際上是同一種情況，如此可大大減少展開盤面的次數。Binary Search Tree 內所記錄的盤面就是透過公式計算得來的數值 insert 進去的。

原則上，在展開一個盤面之前，會先自 Binary Search Tree 中搜尋該盤面是否已經展開過，若無，則繼續展開，若有，則表示此盤面先前已經走過，所以不展開此重複的盤面，但是除了一種特殊的情況：雖然先前已經走過此盤面，但當時的步數比目前走到此的步數還多，則得重新由此盤面展開一次，並更新 Binary Search Tree 裡所記錄盤面的步數。目前的步數較少，表示此時由起始盤面至此盤面的解較先前走到此盤面時的解爲佳，重新展開的目的則在於更新 Binary Search Tree 裡的步數記錄。本演算法之流程圖如下所示：


開始
將啓始盤面棋子的位置放進陣列中
將盤面資料推進堆疊
堆疊中有資料?
否
結束
是
自堆疊取出欲處理的盤面
超過目前最少步數?
是
否
曹操是否走回曹營?
是
記錄目前最少步數
記錄目前參考解法
否
以棋子排序編碼後所得的值爲 Key 在 Binary Search Tree 中尋找是否已經存在?
是
否
將此值加入 Binary Search Tree 中
仍有可展開的盤面?
否
是
在 Binary Search Tree 中尋找新盤面排序編碼後所得的 Key 值是否在在?
否
將正在處理的盤面推入堆疊中
將新盤面推入堆疊中

圖四顯示啓始盤面爲「橫刀立馬」的展開情況。



圖四 啓始盤面爲「橫刀立馬」的 Game Tree 展開情況

程式執行結果與分析

我們使用一台普通的個人電腦做爲測試的工具：Petium 166MMX CPU,160MB PC-100 SD-RAM。使用的 Compiler 爲 Borland C++ Builder 3.0，原始程式碼約 480 行，以下爲程式執行後所得的最佳解結果（表六）：

表六 程式執行結果

盤面名稱	啟始盤面	最少步數	展開盤面總數	執行時間	盤面名稱	啟始盤面	最少步數	展開盤面總數	執行時間
左右佈兵（又名兵臨曹營）	O##O O##O H[ ]H HHHH HH	34 步	3859	71 秒	層層設防	H##H H##H O[ ]O O[ ]O [ ]	103 步	30375	1474 秒
橫刀立馬	H##H H##H H[ ]H HOOH O O	81 步	23853	1835 秒	兵將聯防	O##O H##H H[ ]H O[ ]O [ ]	121 步	36557	1127 秒

將守角樓	H##H H##H O[ ]O HOOH H H	70 步	23111	1660 秒	四路進兵	##OH ##OH OO [ ][ ] [ ][ ]	65 步	13753	1005 秒
屯兵東路	##HH ##HH [ ]OO HHOO HH	74 步	20935	1245 秒	比翼橫空	[ ]## [ ]## [ ][ ] O OH O OH	28 步	5779	307 秒
插翅難飛	H##O H##O [ ]OO H[ ]H H H	63 步	42290	3023 秒	夾道藏兵	##OH ##OH [ ][ ] [ ][ ] O O	76 步	13753	858 秒
重重包圍	H##O H##O H[ ]H H[ ]H O O	74 步	42271	2738 秒	水洩不通	O##H O##H [ ][ ] [ ][ ] O O	80 步	13600	813 秒
雲遮霧障	H##H H##H H[ ]O H[ ]O O O	81 步	42165	3080 秒	將擋後路	[ ]## [ ]## [ ][ ] [ ]OO OO	21 步	2761	137 秒
守口如瓶	O##O H##H HH H OH O [ ][ ]	100 步	43669	2483 秒	前呼後擁	OO## [ ]## [ ][ ] [ ][ ] OO	22 步	6234	173 秒
三軍聯防 (又名 交錯堵道)	##HH ##HH [ ][ ] O[ ]O O O	69 步	16227	1046 秒	調兵遣將	#_#OO ##OO [ ][ ] [ ][ ] [ ]	52 步	5078	240 秒
四將聯防 (又名 四將連關)	##[ ] ##[ ] HH[ ] HHOO O O	40 步	5476	126 秒	巧過五關	O##O O##O [ ][ ] [ ][ ] [ ]	33 步	6052	189 秒

計算步數的方式有兩種：第一種爲只要棋子移動一格就算一步，第二種狀況則爲若兩個空格相鄰，棋子得以連續移動兩格只算一步。依據文獻上的記錄推測，前人文獻計算步數的方法應爲後者，故本程式亦採用後者爲計算步數的方式。程式所得到的結果，與先前得到的文獻資料比較後，有以下三種狀況（表七）。

表七　文獻資料數據與執行結果比較

盤面名稱	起始盤面	資料數據	執行結果	盤面名稱	起始盤面	資料數據	執行結果
與前人資料相符者							
左右佈兵	O##O O##O H[ ]H HHHH HH	34 步	34 步	橫刀立馬	H##H H##H H[ ]H HOOH O O	81 步	81 步
水洩不通	O##H O##H [ ][ ] [ ][ ] O O	80 步	80 步				
步數較前人資料記錄少者							
三軍聯防	##HH ##HH [ ][ ] O[ ]O O O	74 步	69 步	四路進兵	##OH ##OH OO [ ][ ] [ ][ ]	67 步	65 步
步數較前人資料記錄多者							
插翅難飛	H##O H##O [ ]OO H[ ]H H H	62 步	63 步	層層設防	H##H H##H O[ ]O O[ ]O [ ]	102 步	103 步

與先前得到的前人數據有所出入，有幾個可能的原因：一是因爲原先得到的數據就不是正確的，二則是因爲計算步數的方法不一樣。但是仔細驗證幾組程式執行得到較少的參考解（附錄一），發現我們的程式並無錯誤。但是先前收集到的資料只註明盤面的最少解題步數，並沒有附上參考的移動過程，所以應該是以前文獻資料有誤所致。

此外，我們在網路上收集到一個 DOS 下的華容道遊戲軟體 (604.zip)，它有提供參考解答，但大部份盤面所走的步數都比本程式得到的結果爲多，如下表所示（表八）。特別注意的是，在此處計算步數的方式和附錄一中的不同，步數的計算乃是移動一格即算一步，而附錄一則採用若連續移動兩格只算一步的方式計算步數。

表八　604.zip與本演算法結果之比較

盤面名稱	啟始盤面	604.zip	本演算法	盤面名稱	啟始盤面	604.zip	本演算法
左右佈兵 (又名 兵臨曹營)	O##O O##O H[ ]H HHHH HH	49 步	49 步	層層設防	H##H H##H O[ ]O O[ ]O [ ]	140 步	138 步
橫刀立馬	H##H H##H H[ ]H HOOH O O	118 步	116 步	兵將聯防	O##O H##H H[ ]H O[ ]O [ ]	158 步	158 步
將守角樓	H##H H##H O[ ]O HOOH H H	102 步	100 步	比翼橫空	[ ]## [ ]## [ ][ ] O OH O OH	46 步	39 步
屯兵東路	##HH ##HH [ ]OO HHOO HH	104 步	102 步	夾道藏兵	##OH ##OH [ ][ ] [ ][ ] O O	110 步	107 步
插翅難飛	H##O H##O [ ]OO H[ ]H H H	77 步	77 步	水洩不通	O##H O##H [ ][ ] [ ][ ] O O	115 步	114 步
重重包圍	H##O H##O H[ ]H H[ ]H O O	94 步	92 步	將擋後路	[ ]## [ ]## [ ][ ] [ ]OO OO	30 步	30 步
雲遮霧障	H##H H##H H[ ]O H[ ]O O O	105 步	104 步	前呼後擁	OO## [ ]## [ ][ ] [ ][ ] OO	41 步	31 步
守口如瓶	O##O H##H HH H OH O [ ][ ]	132 步	131 步	調兵遣將	##OO ##OO [ ][ ] [ ][ ] [ ]	74 步	72 步
三軍聯防 (又名 交錯堵道)	##HH ##HH [ ][ ] O[ ]O O O	96 步	89 步	巧過五關	O##O O##O [ ][ ] [ ][ ] [ ]	46 步	45 步
四將聯防 (又名 四將連關)	##[ ] ##[ ] HH[ ] HHOO O O	59 步	55 步				

結論

本程式使用DFS的方式求解，只用一些粗略的方法加快程式求解的速度也減少程式之發展時間，雖然多數的盤面在半個小時內能找到最佳解，最慢也能在一個小時內完成。但是如果運用在遊戲中隨時給玩家提示在任一種盤面狀況下，下一步該怎麼走，則似乎又太久了點。因此，如何找出更快的演算法，讓任一個有解的盤面均能在很短的時間內找到最佳解，是未來需要再努力的目標。另外，將來也可以考慮以資料庫來存放各種盤面之最佳解，那麼就可以更快地將下法展示出來。

參考文獻

冼鏡光 (1990).：名題精選百則技巧篇－使用C語言。格致圖書公司。

Grimaldi, R. P. (1994). Discrete and combinatorial mathematics : an applied introduction, 3rd ed.

Ronald, E. W. & Raymond, H. M. (1993). Probability and statistics for engineers and scientists, 5th Edition, by Prentice-Hall, Inc.

Wackerly, & Mendenhall, & Scheaffer, (1996). Mathematical statistics with applications, 5th Edition, by Wadsworth Publishing Company.

David, K. & Ward, C. (1996). Numerical analysis: mathematics of scientific computing, 2nd ed.

Harry, R. L. & Christos, H. P. (1998). Elements of the theory of computation, 2nd ed.

Neapolitan, & Naimipour, (1996). Foundations of algorithms, by D. C. Heath and Company.

Horowitz, & Sahni, (1994). Fundamentals of data structures in Pascal, 4th Edition, by Computer Science Press.

Gilbert, S. (1988). Linear algebra and its applications, 3rd Edition, by Harcourt Brace Jovanovich, Inc.

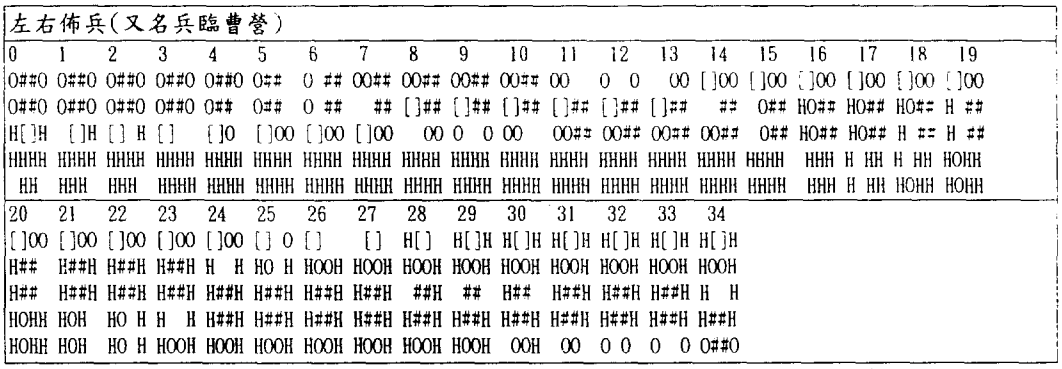
誌謝

本研究部份由國科會專案補助，計畫編號為NSC-88-2815-C-003-002-E，特此誌謝。

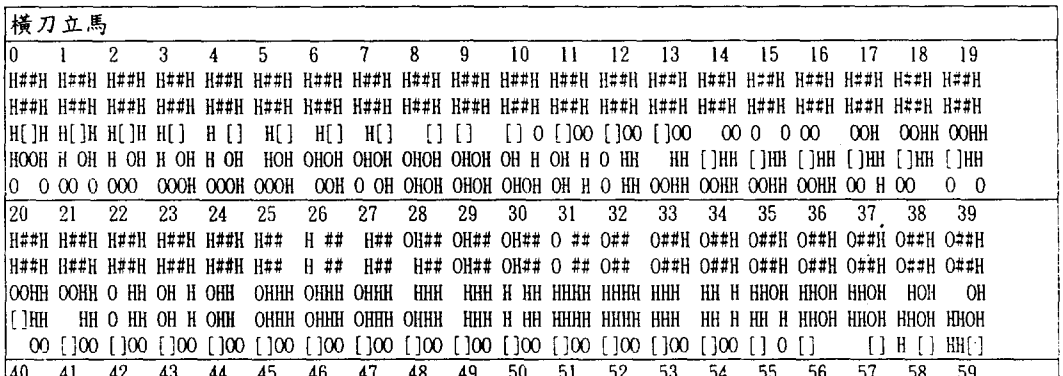
收稿日期：民國88年1月11日
修正日期：民國88年3月 5 日
接受日期：民國88年3月 9 日

附錄一（各盤面本演算法所得的最佳解）

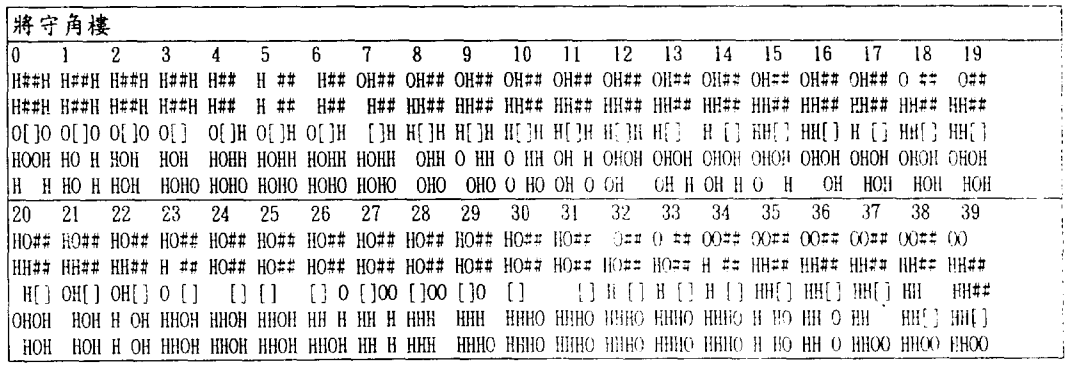
左右佈兵(又名兵臨曹營)

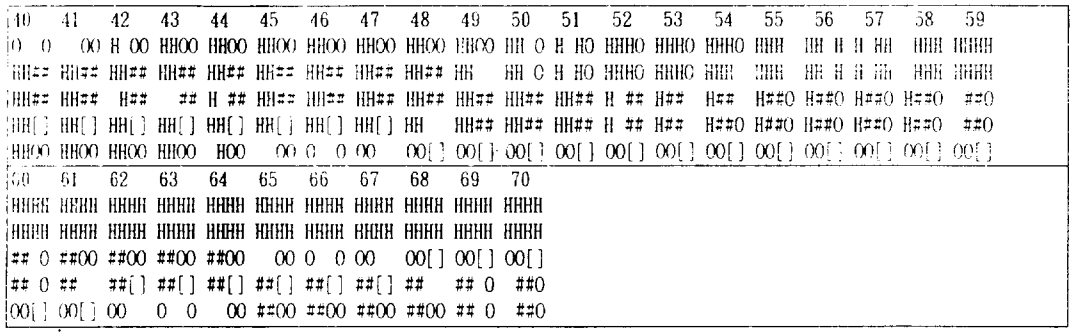


橫刀立馬

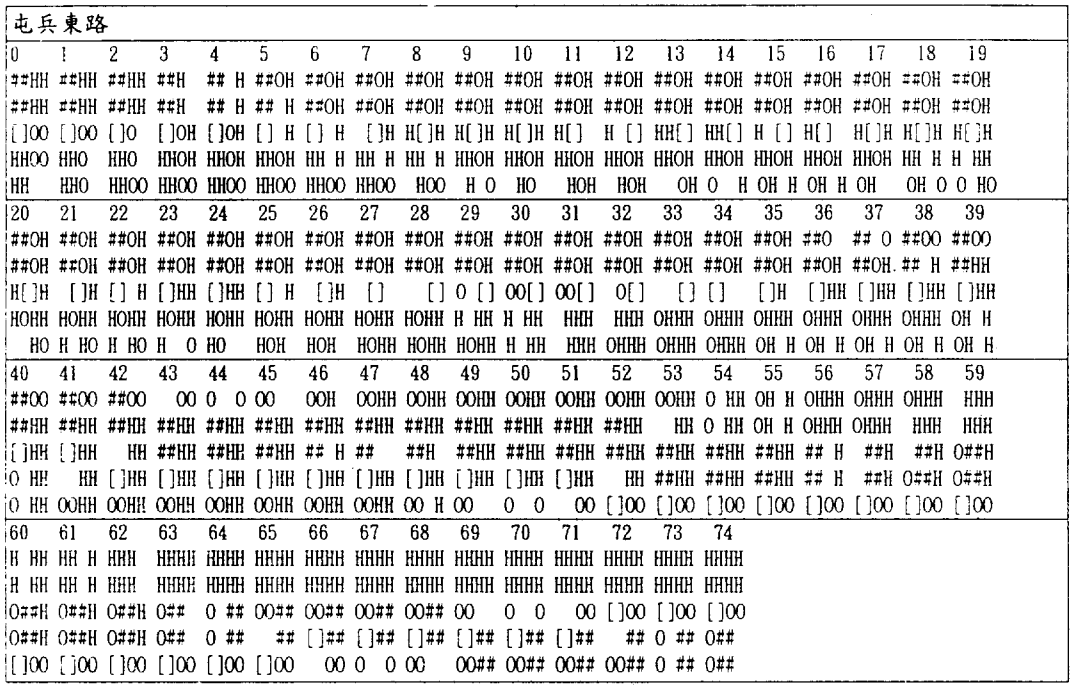


將守角樓

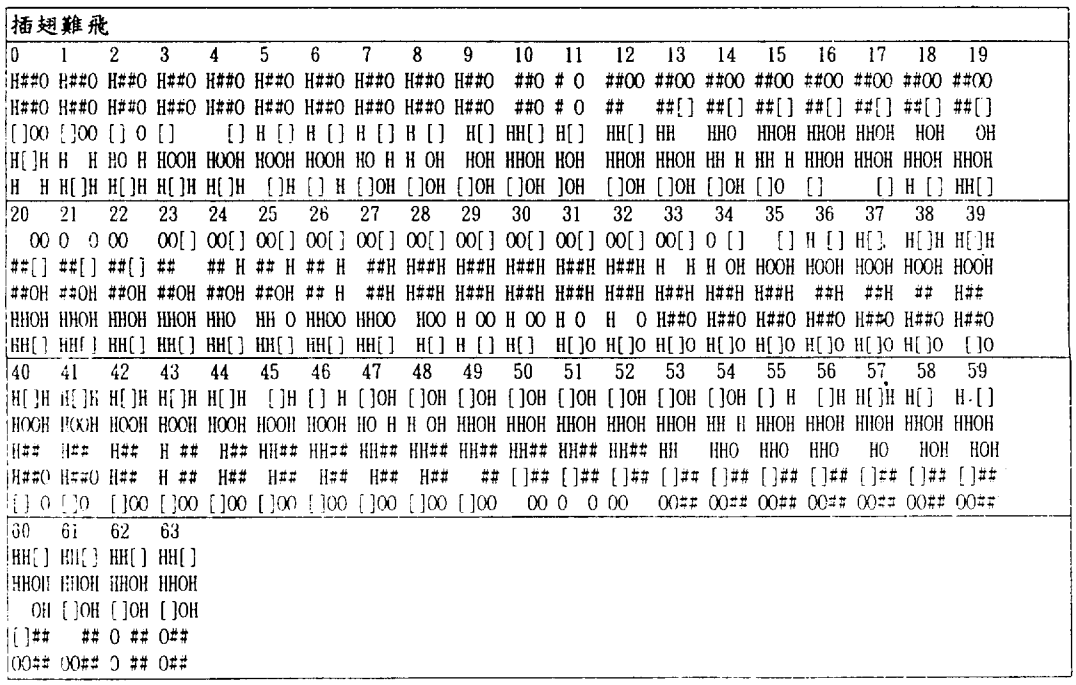




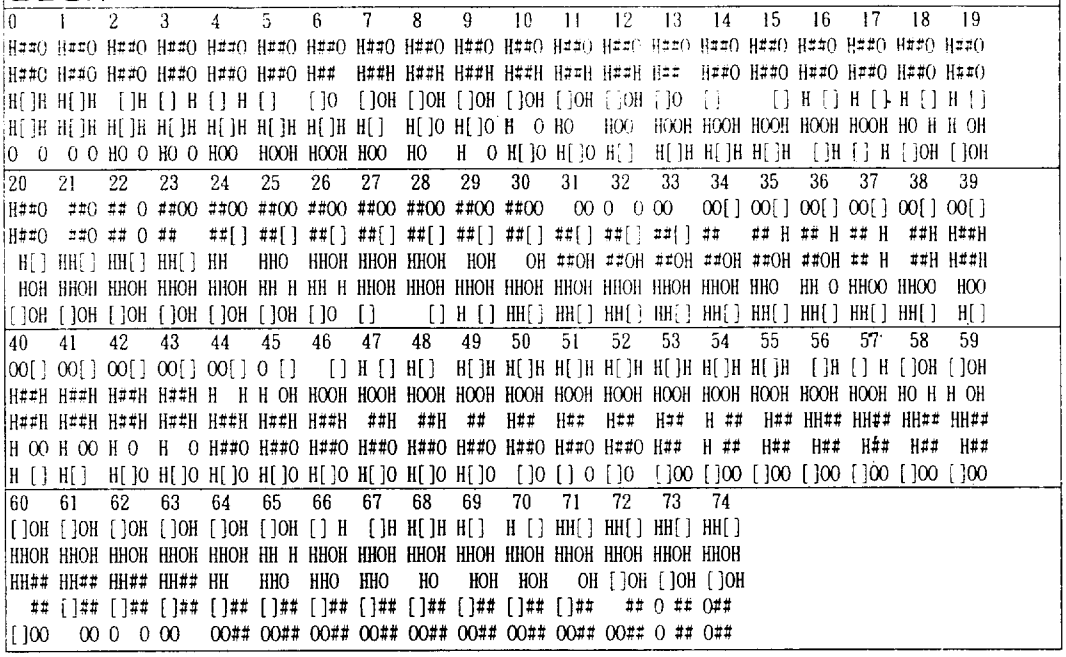
屯兵東路



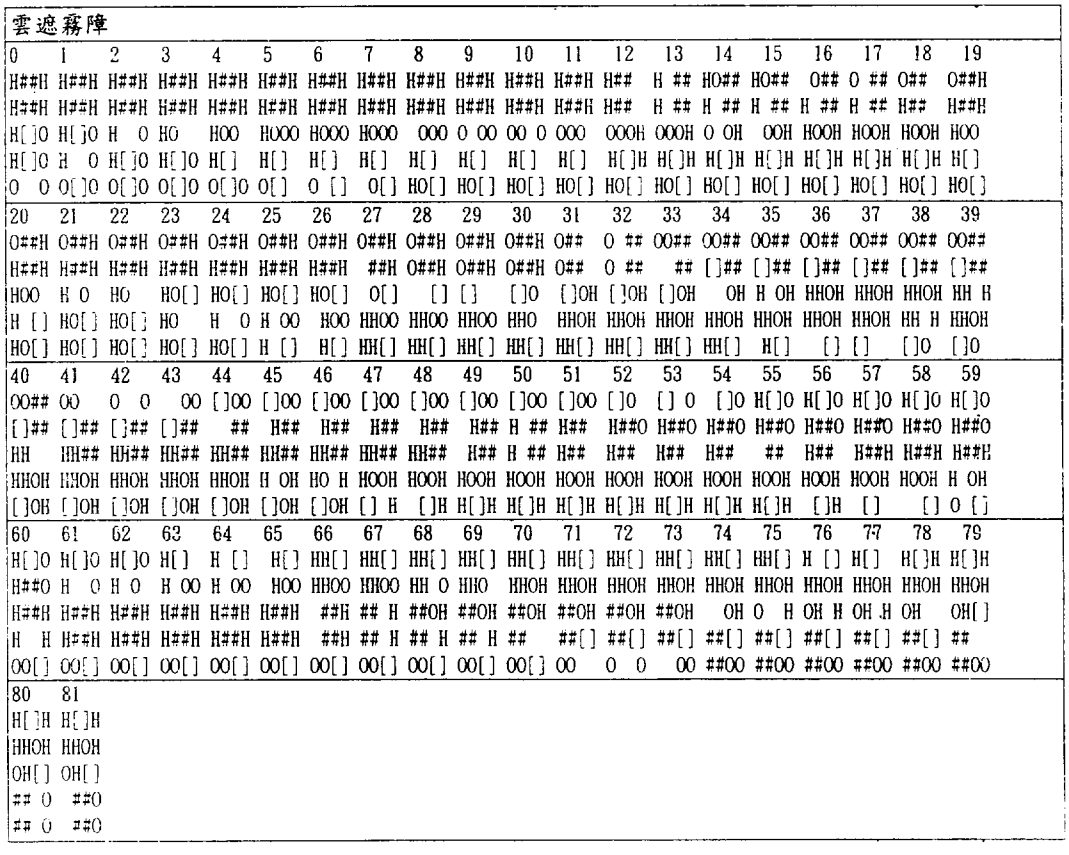
插翅難飛



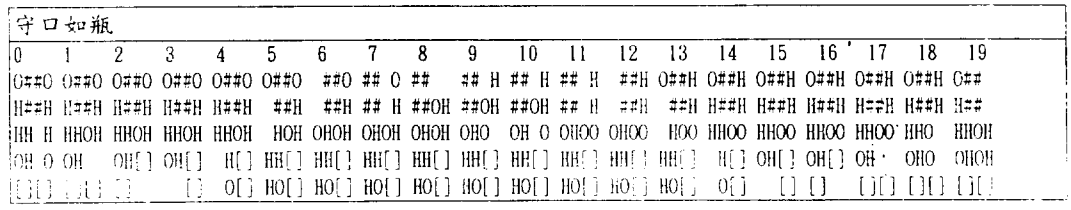
重重包圍

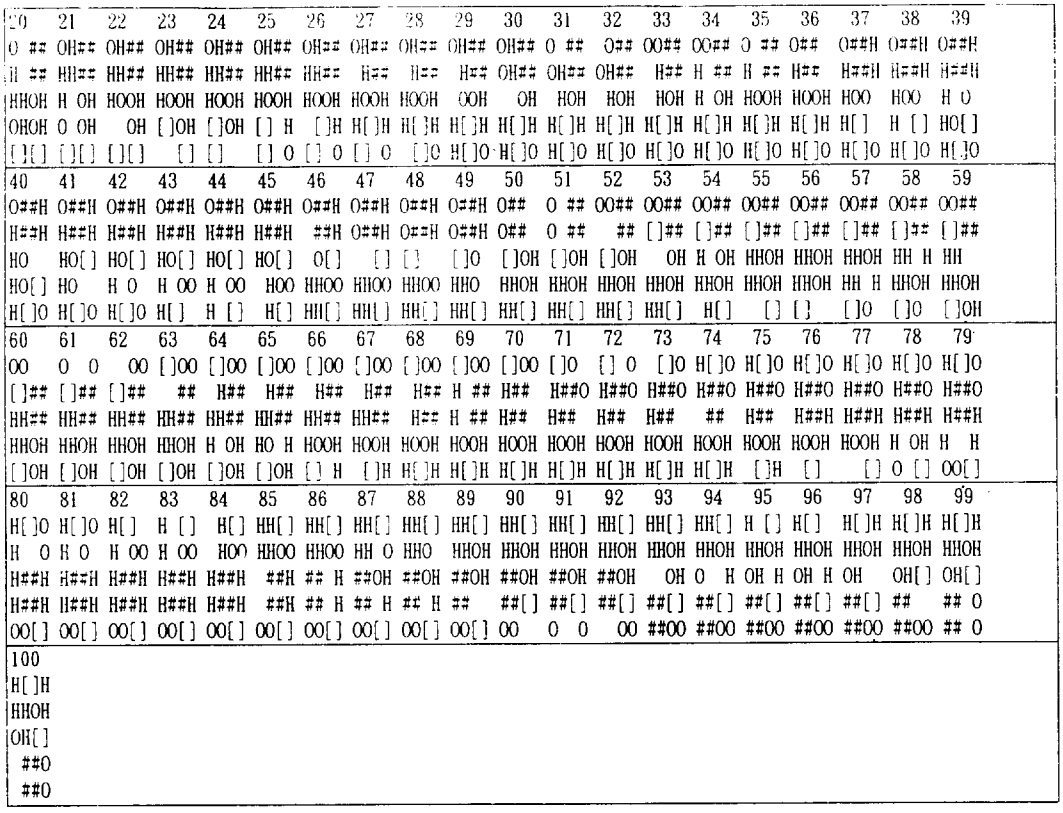


雲遮霧障

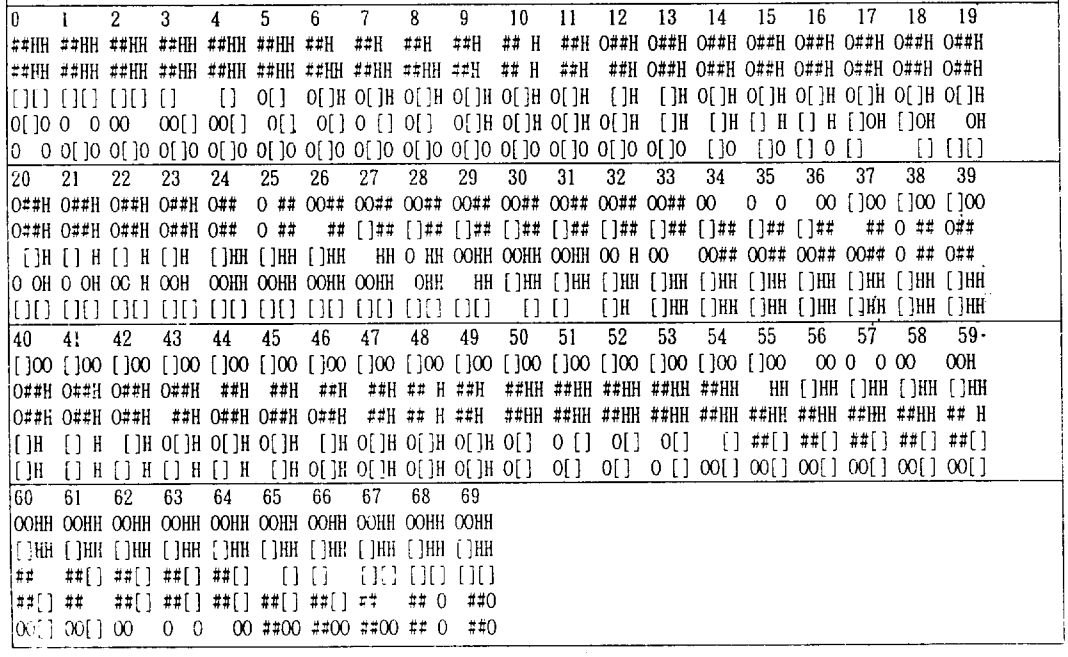


守口如瓶

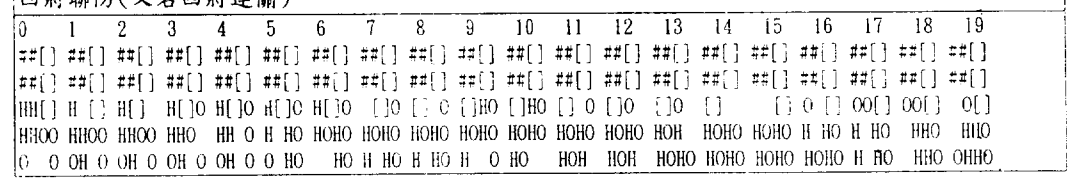


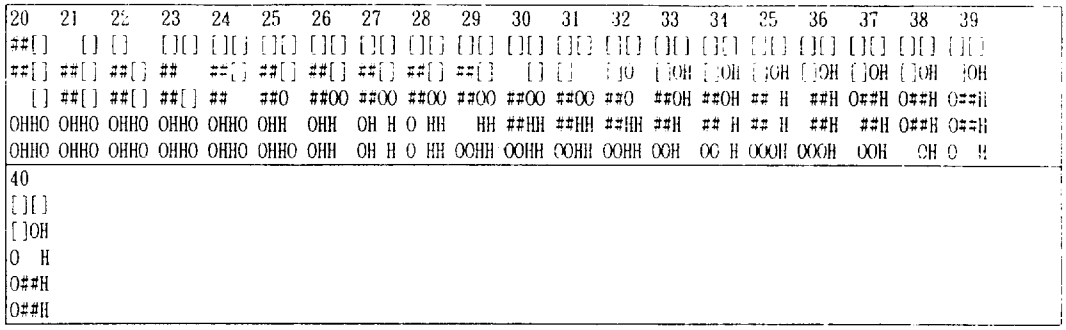


三軍聯防(又名交錯堵道)

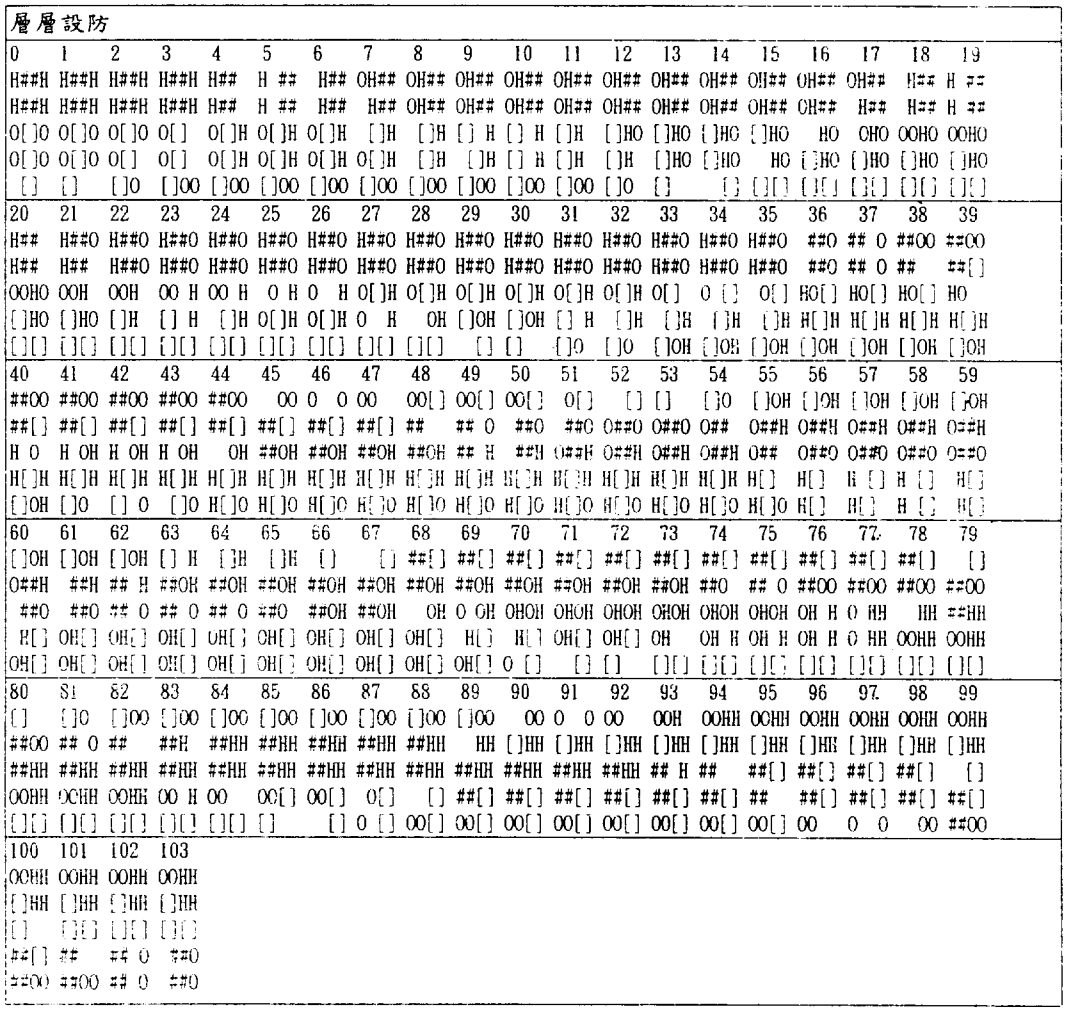


四將聯防(又名四將連關)

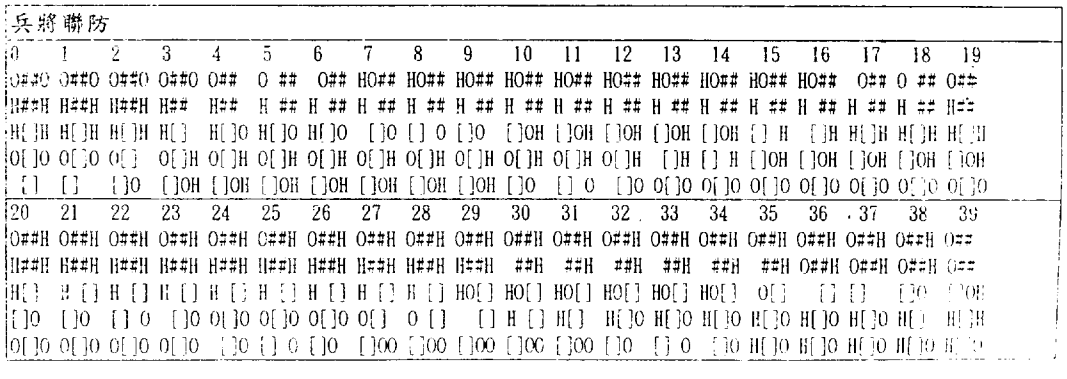


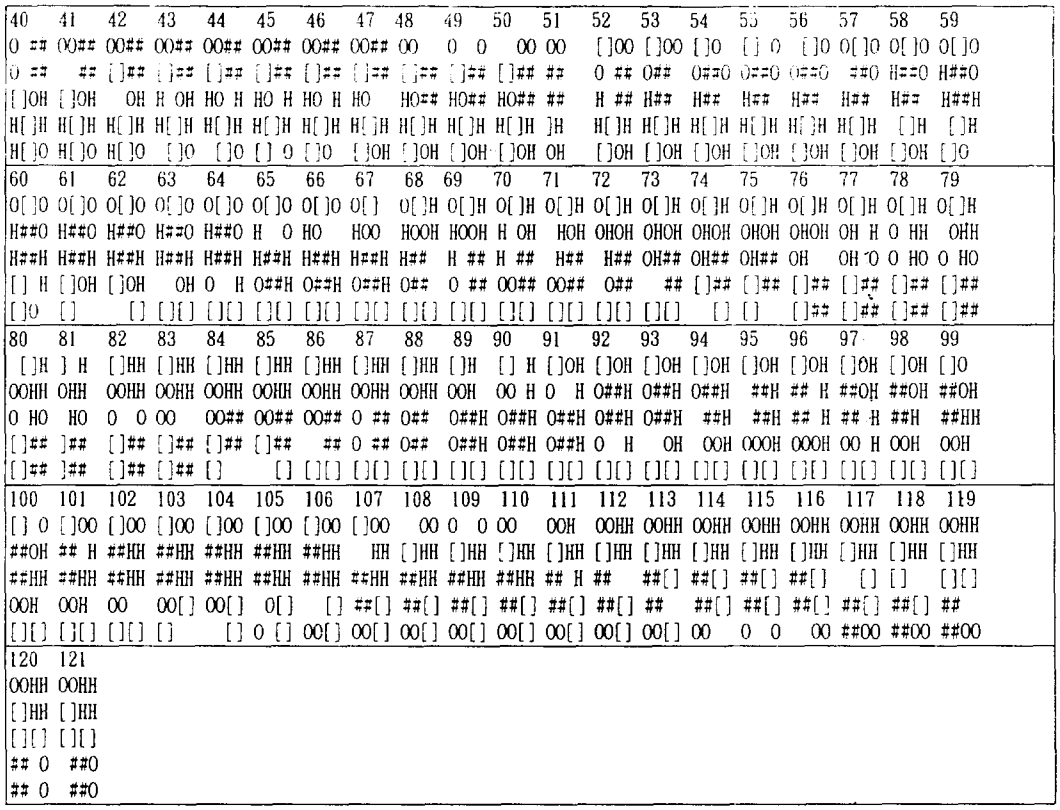


層層設防

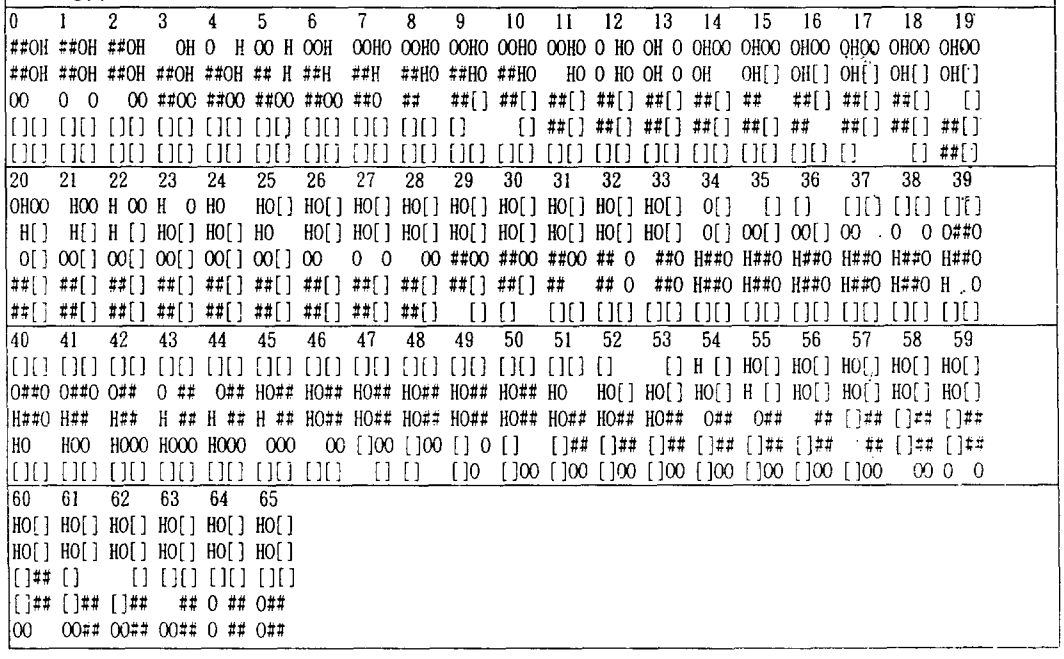


兵將聯防

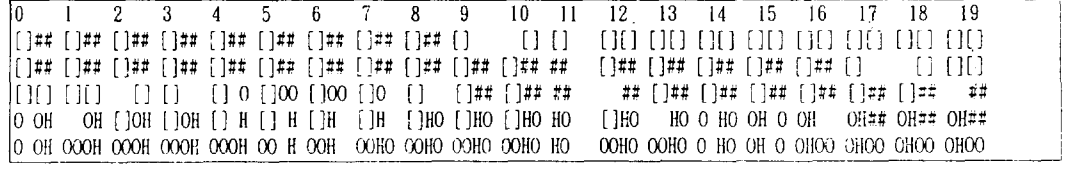


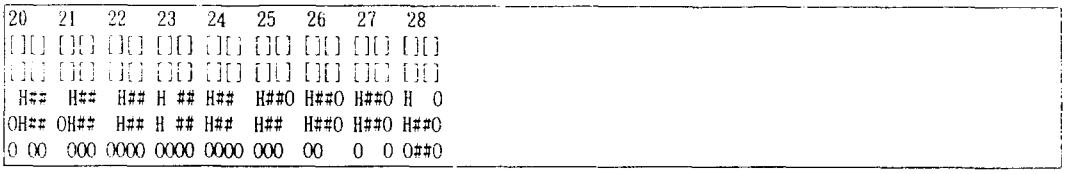

40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
120 121

四路進兵

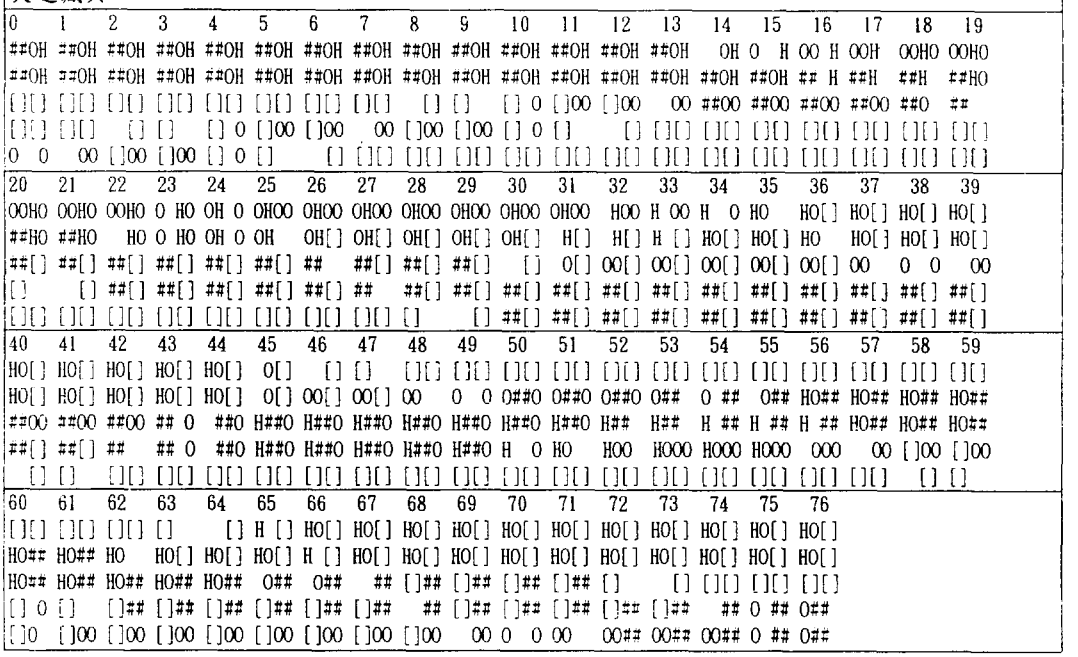

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65

比翼橫空

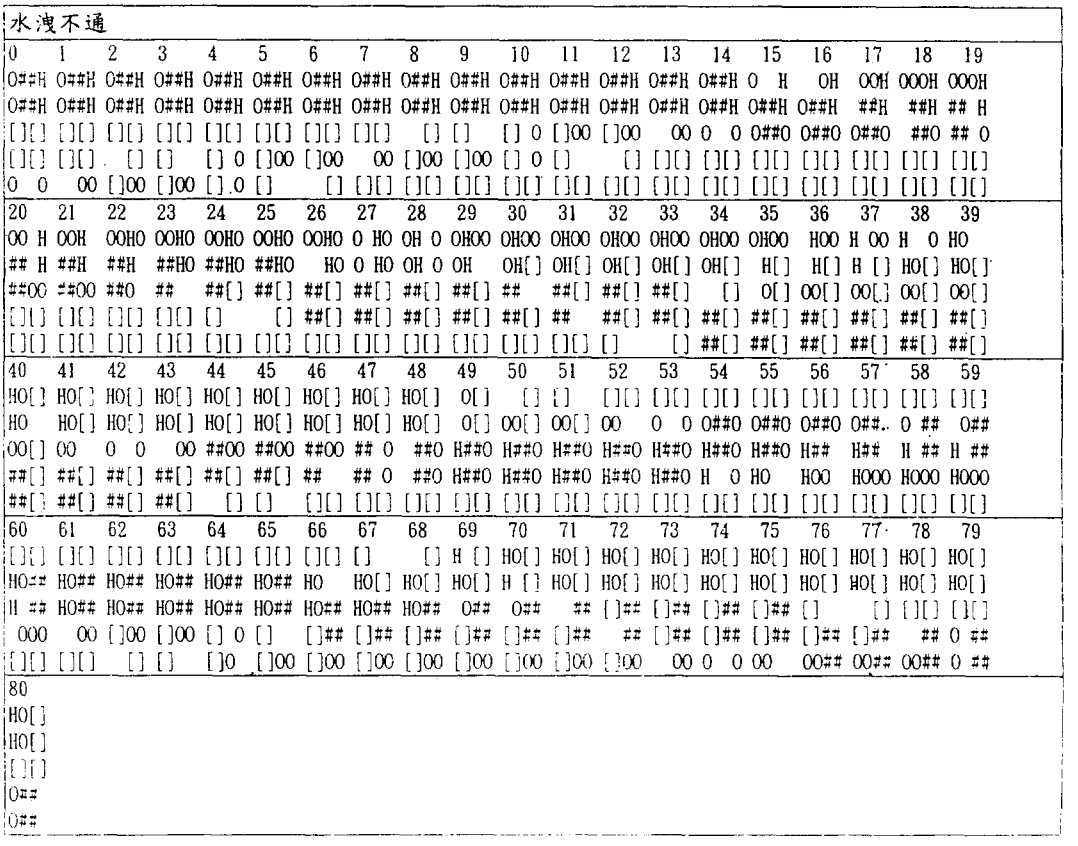

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19



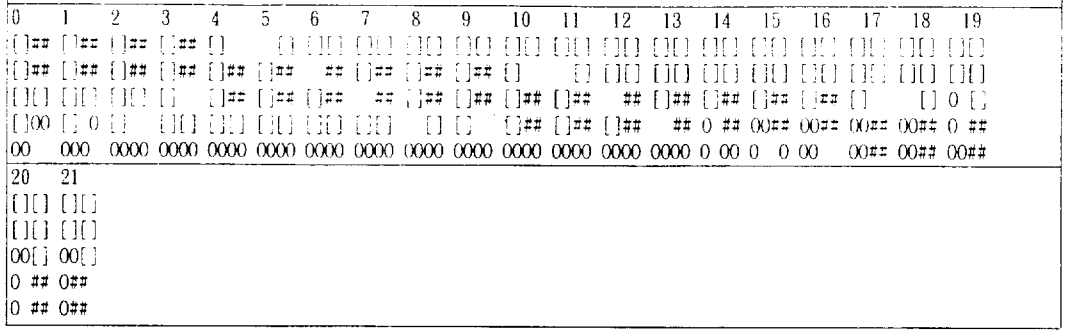
夾道藏兵



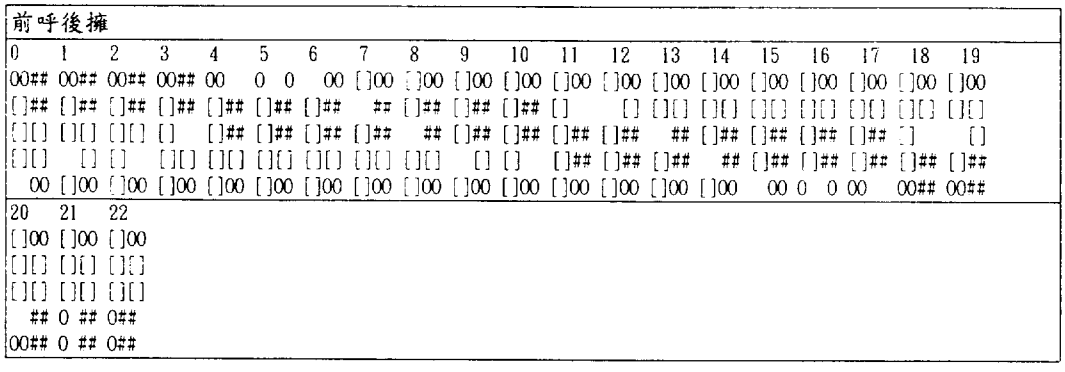
水洩不通



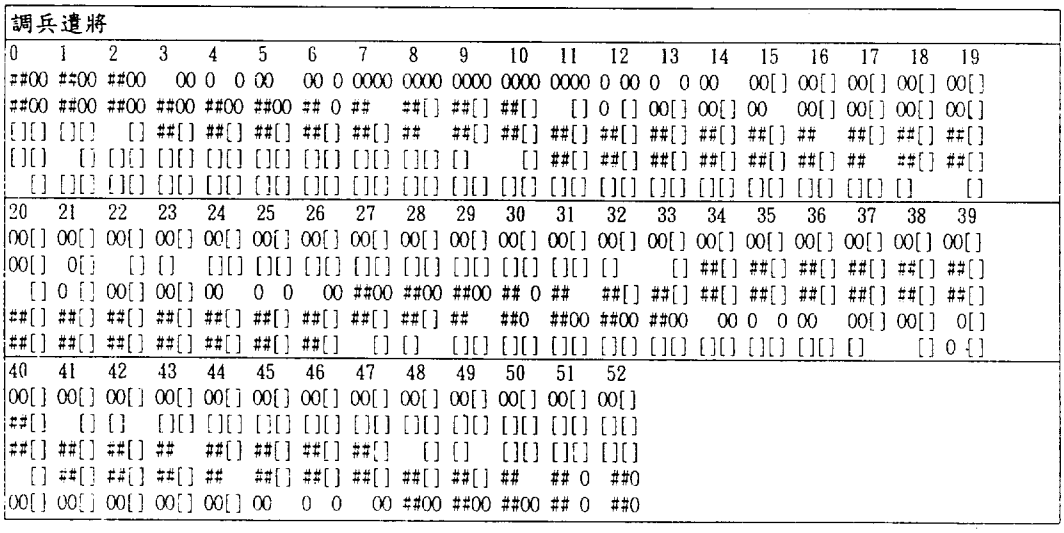
將擋後路



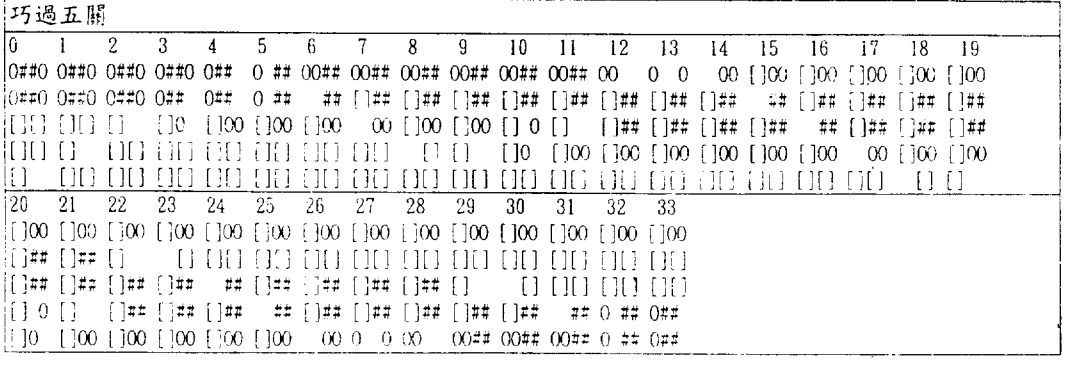
前呼後擁



調兵遣將



巧過五關



Use Computers to Study the Solutions of the Chinese Game "Hua Rong Daw"

Chun-Ling Wei Shun-Shii Lin

Department of Information and Computer Education
National Taiwan Normal University
Taipei, Taiwan, Republic of China

Abstract

In this paper, we will design algorithms to derive the solutions for the Chinese game "Hua Rong Daw". In addition, we want to verify and compare these results with the previous literatures. Since many initial configurations of this game need more than 100 steps to reach the final configurations, we could not search the entire game tree with the "brute force" approach. Previously, there are no computer solutions for this hard problem, but there are many manual trials that are found in many documents. In this paper, we explore some techniques for totally solving this game. The results show that some previous results are not the optimal solutions. We also list our optimal solutions in this paper.

Keywords: game tree, binary search tree, brute force approach