象棋殘局回溯分析演算法之實作與評估

Retrograde Analysis Algorithms for Chinese Chess Endgames Implementation and Evaluation

Haw-Ren Henry Fang(方浩任)
Institute of Information Science
Academia Sinica, Nankang 11529 Taipei, Taiwan
hawren@iis.sinica.edu.tw

論文摘要

電腦象棋的程式設計中，可分爲知識庫設計與對局搜尋兩個方向[7,8,9]。在象棋殘局裏，對於一般的審局函數，棋局往往要十餘回合的推衍方能判斷適當的著手，是故一般的對局搜尋法效果有限。本文提出一個依回溯分析(Retrograde Analysis)[3,5,6]演算法設計的殘局知識庫系統，用來解決一般性的象棋殘局問題。

本系統使用回溯的方式，先建構最基本的殘局資料，依序建構複雜度較高的殘局知識庫。此殘局系統具有完全資訊的特性，即對於知識庫中有記載的殘局，都能明確的知道其正確的勝負狀態與最佳的著手。且知識庫系統是電腦自動建構，故不須要有豐富的專家知識。目前在一般常見的殘局如單傌對單士、炮仕對雙士與單俥對士象全等表現良好。

關鍵詞：Retrograde Analysis, Endgame Database, Computer Chess, Game Theory, 電腦象棋殘局

一、緣起及目的

人們在學習象棋時，研習殘局爲提升棋力不可或缺的課題之一，市面上雖不乏殘局教學的書，然書中對於特定盤面通常僅提及主要變化，其餘變化或許爲棋力中上者視爲理所當然，然初學象棋者研讀起來頗爲艱辛。另如單傌擒孤士等殘局，書中僅提及某些特定盤面之變化，但其餘盤面許多仍具有一定難度，對殘局學習的完備性，實仍不足。此外許多殘局相當艱深，即使是千百年來亦無定論，或是某些殘局的結論會因後人新的發現而被推翻，這些都是現代象棋殘局所面臨的問題。

在科技發達、電腦普及的現代，運用電腦達到象棋殘局教學的目的，已不是夢想。我們可以用今日電腦強大的計算能力與高容量的記憶體，計算並儲存重要的結果，建構完整的殘局知識庫。由於知識庫的完備性，對於知識庫中有記錄的殘局，學習者都可以獲得完善的解答，並可做進一步的統計分析。此外，對於某些高難度的殘局，運用電腦強大的計算能力與豐沛的記體，給予充足的時間仍有辦法獲得欲知的結果，如此，一些難有定論的殘局就能得到解答。

二、研究方法與成果

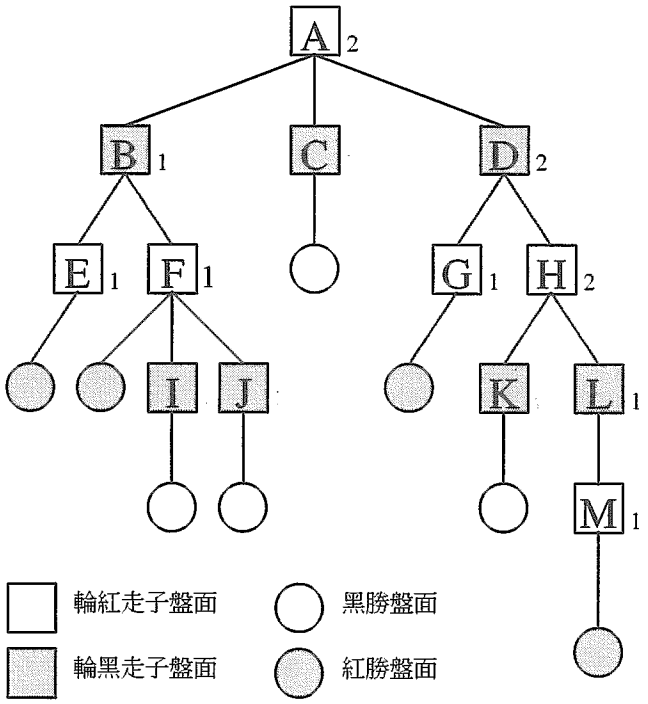
在本殘局知識庫系統裏，對於每個殘局盤面，使用完全雜湊函數將之對映到一個位址，記錄其盤面之勝、負、和的情形，與相對應的最佳著手數，即最短致勝步數或最長延緩步數。在完全雜湊函數的設計上，須使儲存空間達到有效率的運用，本系統對於給定之欲建構的知識庫，先產生其必要的編碼表。首先，對紅方之帥及所有防守子(即相、仕)在棋盤上所有可能出現的排列依序編碼；再者，若紅方有紅俥(無論單俥或雙俥)，則對其所有在盤面上可能出現的排列依序編碼，紅傌、炮、兵的考慮情形亦然；最後，依黑方的子力也做類似的步驟。對於給定之盤面，先依編碼表找出該盤面所有相對應的號碼，每個號碼有其權值，即爲在其之前所有編碼表對映空間大小之積，在本系統實作上，編碼表的順序爲帥仕相、俥、傌、炮、兵、將士象、車、馬、包、卒，最後再加總所有號碼與其權值之積即爲其雜湊函數值。因將子力分組(至多十組)編碼，比逐一考慮各子大幅減少因棋盤上一位置至多僅能擺一子或單一兵種有兩子以上所產生的儲存空間浪費。然象棋棋盤是左右對稱，若兩盤面可因左右翻轉後一致可視爲同一盤面，此二盤面經前述之雜湊函數在知識庫中仍佔有兩份空間，故擇一前述之編碼表修改爲將所有因，並在此兩盤面中擇一爲標準盤面，在求雜湊函數值時若給定之盤面不爲標準盤面先翻轉爲標準盤面再求其值，如此可使知識庫所須的空間減少許多，修改不同的編碼表會造成所須空間減少的比例不一。表(1)爲除四兵(或四卒)及五兵(或五卒)外所有可能的子力組合分類其編碼表對映空間的大小，在雜湊函數設計上選擇空間比率最小的編碼表作爲，如此知識庫所使用的空間爲最少。另由表(1)中可發現所有攻擊子力(即俥、傌、炮)的編碼表末欄比例必大於其餘任何編碼表的比例，而紅黑雙方皆沒有攻擊子力的盤面若非將帥相見即和棋，故將左右翻轉後會一致的兩盤面對映到相同位置的編碼表在程式上不用實作。

假設盤面節點P爲輪紅方走子則紅方必勝，且最短須用n步，則P的子盤面中，若是走向最強攻擊，應該黑方最多只能拖延n-1步，而這樣的盤面可能不只一個，其餘盤面若不是可 讓黑方拖延超過n-1步就是不幸走錯變成和棋甚至致負。再假設Q節點輪紅方走子而紅方輸棋，且最多只能延緩n步，則在Q的子盤面中，黑方最差的狀況亦可在n步取勝，走向這樣的盤面爲最佳防禦著手，其餘盤面只會讓

黑方取勝的所須的著手數更少。若節點R爲輪紅方走子而紅方至多和棋，則在R的子節點中，應有輪黑方走子和棋的盤面，這種盤面通常不只一個，R其餘的子盤面都是黑勝。反之，今若輪黑走子討論情形亦類似。如此，在一個具有封閉性的知識庫中（即在知識庫中的任一盤面之所有子盤面的盤面資料皆可在知識庫中找到）對勝定的盤面可依上述討論循序致勝，對已敗的盤面可延緩的對方的攻擊增加對方出錯的機會。圖1爲每一盤面皆記錄其殘局資料之對局樹範例。

表(1)各種象棋子力組合編碼表對映空間的大小

子力組合	編碼表對映空間的大小		比率 (A)/(B)
	將左右翻轉後一致的兩個盤面視為同一盤面(A)	將左右翻轉後一致的兩個盤面視為不同盤面(B)	
三兵或三卒	13219	26235	50.387%
雙俥(傌、炮等)	2045	4005	51.061%
雙兵或雙卒	765	1485	51.515%
單俥(傌、炮等)	50	90	55.556%
單兵或單卒	31	55	56.364%
帥雙相(將雙象)	120	183	65.574%
帥相(或將象)	41	62	66.129%
帥仕雙相(將士雙象)	537	810	66.296%
帥(或將)	6	9	66.667%
帥仕相全(或將士象全)	948	1410	67.234%
帥仕(或將士)	27	40	67.500%
帥相雙仕(或將象雙士)	326	480	67.917%
帥仕相(將士象)	184	275	66.909%
帥雙仕(將雙士)	48	70	68.571%



1,2,3 ... 表紅勝紅方所須之步數

, , ... 表黑勝黑方所須之步數

圖1 含殘局資料之對局樹

在中局常用的Min-Max與 $\alpha-\beta$ 切捨等演算法所發展出來的對局搜尋技巧並不適用於此殘局知識庫之建構，本文依回溯分析原理，提出一有效之方法建構象棋殘局知識庫以解決一般性的象棋殘局問題。

爲了知識庫之建構與使用之方便，將知識庫以其剩餘子力之種類與多寡分類，並定義子知識庫爲知識庫中盤面吃子後之子盤面其被包含之知識庫。圖2爲一知識庫結構範例。在欲建構一知識庫時，先建構其所有的子知識庫，而最簡單的知識庫僅有將帥兩子之所有盤面資訊，演算法(1)表示這樣的關係。

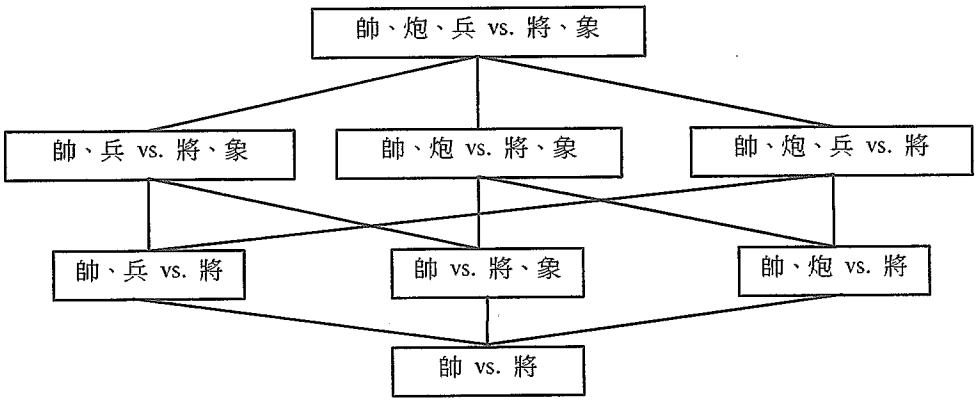


圖2 知識庫結構範例

演算法(1)殘局知識庫系統建構演算法

```
procedure establish(DB as database);
begin
    for i←each unestablished
    supporting database of DB denoted
    by d do
    begin
        if d=null then
        begin
            establish(d);
        end;
    end;
    construct(DB);
end;
```

一盤棋局通常結束於將或帥的被吃或是將帥相見(在象棋棋規上尚有長捉、長照爲禁著等規則)，故對所欲建構之知識庫，先判斷所有盤面是否有將帥相見或有子盤面已造成吃帥或吃將，這樣的分析僅須做一次。之後對於每一個輪紅走子的盤面而言，若所有子盤面均爲黑勝，則此盤面爲黑勝，且最佳防禦著手數爲子盤面中最佳攻擊著手數的最大值。若子盤面中只要有一個紅勝，則此盤面即可判定爲紅勝，最強攻擊著手數爲所有紅勝的子盤面之最佳防禦著手數的最小值加一。在建構知識庫時，不一定知道所有紅勝子盤面之盤面資料，故先將已知之紅勝子盤面之最佳防禦著手數的最小值加一填爲此盤面之最佳攻擊著手數，這個值必定大於或等於真正的最佳攻擊著手數。在程式實作時，爲增加執行效率，可在發現有一紅勝子盤面時，即將原盤面判定爲紅勝，最佳攻擊著手數填爲該子盤面之最佳防禦著手數加一。而若無紅勝子盤面且含有子盤面盤面資訊未知則原盤面仍是無法判別其勝負，如演算法(2)所示，其中*win*(*n*)表在知識庫中盤面節點*n*是否已可得知勝定(即若輪紅走子紅勝，反之若輪黑走子黑勝)，*lose*(*n*)、*draw*(*n*)與*unknown*(*n*)定義亦類似，*ply*表最強攻擊著手數或最佳防禦著手數，*sub*(*n*,*i*)表盤面*n*中第*i*個子盤面。

演算法(2)由子盤面推知盤面狀態

```
record info
begin
    status, ply as integer;
end;
function detect(n as node) as info;
var
    i, ply as integer;
begin
    detect.status←lose;
    for i←1 to child_number(n) do
    begin
        if lose(sub(n,i))=true then
        begin
            detect.status←win;
            detect.pl ←ply(sub(n,i))
            +1;
            return;
        end;
        else if win(sub(n,i))=true then
            detect.ply←max(ply(sub(n
            ,i)),detect.ply);
        else if unknown(sub(n,i))=true
        then
            detect.status←unknown;
        else if draw(sub(n,i))=true
        then
            if detect.status<>unknown
            then
                detect.status←draw;
    end;
end;
```

運用前述之特性，將欲建構之知識庫中所有資訊未知之盤面，反覆地展開一層之對局樹的子盤面判斷該盤面狀態，如此反覆，直到不再有新的盤面資訊產生爲止。考慮任一盤面資訊未知的盤面節點展開其對局樹，以圖3爲例，展開輪紅走子之盤面節點

A，由於迴圈已無法得到新的盤面資訊，故節點A一定沒有紅勝的子盤面，且不全爲黑勝子盤面，即至少有一個或以上的狀態未知盤面或和棋盤面。在正常情形下，紅方不會走向黑勝的盤面，而節點A之子盤面皆不爲紅勝，故會選擇走向和棋盤面或是狀態未知之盤面。若走向狀態未知盤面，如此反覆，在雙方沒有錯著之下，棋局終會成爲和局。故在最後將剩餘的盤面資訊未知之盤面都判定爲和棋，如演算法(3)所示。

面n，在紅勝的子盤面中選擇走向盤面m其最佳防禦著手數爲所有紅勝的子盤面中最小，即子盤面m滿足等式$ply(m)=min\{ply(sub(n,i))|lose(sub(n,i))$ ，此時不等式$ply(n)>ply(m)$必定成立。而若選擇走向盤面m，因m盤面爲輪黑走子紅勝，故盤面m的所有子盤面皆爲紅勝，即$win(sub(m,i))$恆真，且等式$ply(m)=max\{ply(sub(m,i))\}$必定成立。所以$m$無論選擇走向任何一個子盤面$k$，皆滿足不等式$ply(m)\geq ply(k)$。是故，可得不等式$ply(n)>ply(m)\geq ply(k)$，即$ply(n)>ply(k)$。也就是說，以知識庫所記錄的最

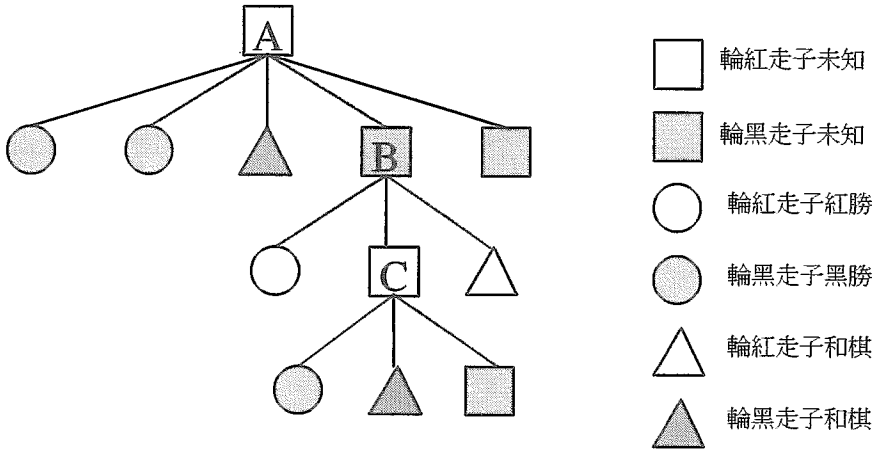


圖3 和棋盤面之判定

演算法(3)殘局知識庫建構演算法

```
procedure construct(DB as database);
begin
    raw_deal(DB) /*判斷棋局是否下一步就結
    束，即判斷最佳攻擊著手數爲1之情形*/
    repeat
      for i←each node in DB denoted
      by n do
        if unknown(n)=true then
          begin
            update n by detect(n);
          end;
    until no more information can be
    generated;
      for i←each node in DB denoted
      by n do
        if unknown(n)=true then
          begin
            set information of n to be
            draw;
          end;
end;
```

對於可勝的盤面，良好的殘局知識庫必能引導其走向勝利[2]，而僅有盤面勝、負、和的 資訊是不夠的，因系統可能在可勝盤面中繞而被認定爲和棋。在一個尚未最佳化的殘局知識庫系統的應用上，對於一個必勝盤面，展開其一層之對局樹，在其所有必敗的子盤面中，選擇走向最佳防禦著手數爲最小之盤面。這個著手雖不一定爲最強攻擊，但依此漸進，最終必定可以致勝。假設一輪紅走子必勝之盤佳著手數而言，經過了一個回合的走子後，紅方離勝利就更近了。如此反覆，紅方必能得勝。

演算法(4)知識庫最佳化演算法

```
procedure optimize(DB as database) ;
var
    n as node;
    i, max_ply as integer;
begin
    max_ply←0;
    for n←each node of DB do
        max_pl ← max(max_ply,ply(n))
        ;
    for i←2 to max_ply do
    begin
        for n←each node of DB do
            if win(n)=true and ply(n)>i
            then
                if
                i-1=max{ply(m)|m=sub(n
                ,j) for each j with
                lose(m)=true} then
                    ply(n) ← i;
        for n←each node of DB do
            if lose(n)=true and ply(n)>i
            then
                if
                i=max{ply(m)|m=sub(n,j
                ) for each j} then
                    ply(n) ← i;
    end;
end;
```

一個完美的殘局系統可引導可勝盤面以最少的著手獲勝且對於必敗的盤面可拖延最多步才敗，這樣的拖延常常可增加對手出錯的機會[4]。在本知識庫建構完成初並不具有這樣的特性，故加上下述最佳化的步驟。在此殘局知識庫系統中，不記錄缺乏將、帥等過於顯易的盤面，故所有有勝負的盤面所儲存著手數之最小值爲1，而所記錄的著手數若有誤差僅會比最佳著手數大，可得知著手數爲1之盤面的資料是一定正確的。對於例勝之盤面，所有一手可獲勝的盤面都可以在最初資料建構時得知，故若資料記錄爲含兩手以上方能獲勝的盤面不可能最佳攻擊著手數爲1；最佳防禦著手數不爲1的盤面必含有最佳攻擊著手數紀錄大於1的子盤面，而所有最佳攻擊著手數爲1的子盤面在建構知識庫初即判斷出，故這樣的子盤面最佳著手數必不爲1，即除了最佳著手數爲紀錄爲1的資料不但是正確的，最佳著手數紀錄大於1的資料盤面其真正的最佳著手數也必不爲1。此時對所有所記錄的著手數大於2之例勝的盤面展開其對局樹，若發現有子盤面最佳防禦著手數爲1則將原盤面最強攻擊之著手數改爲2，再對所有所記錄的著手數大於2之必敗盤面展開其對局樹，若發現子盤面對方最強攻擊著手數之最大值爲2時，則將原盤面最佳防禦著手數改爲2。至此，可得所有著手數應記錄爲2的盤面資訊都是正確的。這樣更新最佳著手數的迴圈可以反覆地執行，直到所有的盤面都是得到真正的最佳值爲止。而在最佳化的一開始就知道所有考慮盤面的最佳著手數之最大值並不容易，然所有盤面一開始所儲存著手數之最大值會大於等於最佳著手數之最大值，故迴圈可考慮至此最大值即可確保所有的資料都已最佳化。上述討論可由演算法(4)表之。

目前對於一般的殘局問題，本系統已可得到良好的解答，並利用知識庫的完備性，做進一步的分析如在所有單傌對單象的殘局盤面中，輪紅走子紅方的勝率爲28.95%，輪黑走子紅方的勝率爲6.07%。而在所有炮仕對雙仕的紅勝盤面中，輪紅走子平均使用17.21個著手方能致勝，輪黑走子則平均要20.45個著手，表(2)爲常見之殘局統計資料(註:表列勝、負機率之樣本空間包含所有可能的盤面但捨棄將帥相見之盤面，並盤面左右反轉可得相同盤面仍視爲兩個相異之盤面，且所有盤面之權重皆相同。其中單俥對士象全的殘局中，在對弈時，進入殘局階段士象通常已連結好，然而在機率空間中各盤面無論士象是否連結好的權重皆相同，故求得之勝率較一般例和殘局高。)

表(2)象棋殘局勝率、平均最佳著手數、最佳著手數最大值列表

殘局種類	紅方勝率		平均最佳著手數		最佳著手數最大值	
	輪紅走子	輪黑走子	輪紅走子	輪黑走子	輪紅走子	輪黑走子
單傌對單士	99.74%	96.42%	13.42	16.20	20	20
單傌對單象	28.95%	6.07%	7.19	10.01	17	16
單俥對雙士	100%	96.88%	4.77	7.06	11	11
單俥對雙象	100%	94.39%	5.03	8.64	13	13
單俥對雙士象	100%	94.51%	7.29	11.06	17	17
單俥對士雙象	100%	93.36%	7.55	12.45	19	19
單俥對士象全	84.60%	51.44%	10.84	16.80	33	33
炮仕對雙士	99.94%	96.67%	17.21	20.45	25	25

三、結論

本論文試圖以回溯分析演算法使用電腦強大的計算能力與豐沛的記憶體解決象棋殘局的問題。此法從基本的殘局著手，逐步建構複雜度高的殘局，並儲存成知識庫的形式。目前發展出來的系統具有下列優點：

1. 知識庫是以回溯法窮舉而得，故不須要豐富的專家知識。另一方面，相較於忠告語言法所建立的殘局系統，可避免因人爲考慮的疏失使系統對於某些盤面產生錯誤判斷。

2. 知識庫利用完全雜湊函數(complete hash function 讀取資料，故不必記錄該盤面子力位置資訊，有效地節省磁碟空間資源。

3. 在使用上，對於給定的一個殘局盤面，展開一層之對局樹即可得正確的著手，所以不因殘局的難度而影響得到解答所須的時間。相較於傳統搜尋法解決殘局的問題，所須時間極少。

4. 知識庫內容具有「完全的知識」(perfect knowledge)的特性，故所得到的解答爲真實解，並不僅爲特定審局函數下的最佳解。即一個可勝的殘局利用此知識庫都能以最少著手獲得勝利，一個必敗盤面可抵擋最多手才負，一個可和的盤面使用此知識庫也不會走出致負的著手。

由於過去甚少有人研究電腦象棋殘局的問題，這套知識庫系統爲電腦在象棋應用上的一大突破。在未來，研究發展主要爲加強電腦象棋殘局能力與殘局知識庫的應用兩個方向。

參考文獻

[1] Barth, W. and Barth, S. "Validating a Range of

Endgame Problems" ICCA Journal, Vol. 15, No. 3, 1992, pp. 132-139.
[2] Barth, W. "Combining Knowledge and Search to Yield Infallible Endgame Problems" ICC Journal, Vol. 18, No. 3, 1995, pp. 148-160.
[3] C. Wirth and J. Nievergelt "Exhaustive and Heuristic Retrograde Analysis of KPPKP Endgame" ICCA Journal, Vol. 22, No. 2, 1999, pp. 67-80.
[4] Dekker, S. T., Herik, H. J. and Herschberg, I. S. "Perfect Knowledge and Beyond" Beal, D. F., Ed., Advances in Computer Chess 5, Elsevier Science Publishers, 1989, pp. 295-312.
[5] Stiller, L. "Some Results from a Massivel Parallel Retrograde Analysis" ICCA Journ al, Vol. 14, No. 3, 1991, pp. 91-93.
[6] Thompson, K. "Retrograde Analysis of Certain Endgames" ICCA Journal, Vol. 9, No. 3, 1986, pp. 131-139.
[7] 許舜欽，黃東輝 "中國象棋知識庫之設計與製作" 民國74年全國計算機會議論文集, 1985, pp. 505-509.
[8] 許舜欽 "電腦西洋棋和電腦象棋的回顧與前瞻" 電腦學刊, Vol. 2, No. 4, 1990.
[9] 張躍騰 "人造智慧在電腦象棋上的應用" 台大電機研究所碩士論文, 1981.
[10] 詹杰文 "一個基於知識的象棋殘局系統之研製" 交通大學資訊科學研究所碩士論文, 1991.
[11] 廖嘉成 "利用計算機下象棋之實驗" 交通大學計算機工程研究所碩士論文, 1982.