運用 Spark 及 Pattern 以提升暗棋盤面等價分類程式之效能

蕭名凱
國立彰化師範大學資訊工程所
shiauyue1017@gmail.com

陳宏光
國立勤益科技大學電子工程學系
hankchentw@gmail.com

賴聯福
國立彰化師範大學資訊工程學系
lflai@cc.ncue.edu.tw

伍朝欽
國立彰化師範大學資訊工程學系
ccwu@cc.ncue.edu.tw

熊原朗
國立彰化師範大學資訊工程學系
bearflorencea@gmail.com

摘要

近年來在人工智慧、大數據等領域中都有許多結合了棋類遊戲的研究，其中，中國暗棋屬於不完全資訊的棋類遊戲。因為在尚未翻開蓋著的棋子時，沒辦法得知所有棋子的位置，故暗棋在研究上比起西洋棋、象棋等遊戲更為困難，空間複雜度也更高。造成儲存盤面資料的資料庫過於龐大，在搜尋樹的查找上也會比較費時。因此，陳志昌教授提出可以將等價的盤面進行合併[7]，以減少資料庫儲存盤面的資料量。本論文改良陳志昌教授的方法，提出透過 Capturing Relation Matrix 運算並且紀錄下每個盤面的 Threat-relation，我們將每個盤面做成 Capturing Relation Pattern，透過我們所提出的 Parallel Refinement Pattern Generating Algorithm 用以取代原作者提出的 Stepwise Refinement Matching Algorithm，以及 Accelerated SRM Algorithm，我們先透過演算法讓每一組盤面產生 Capturing Relation Patter，最後透過合併擁有相同 Pattern 的盤面以達到等價分類。在程式實作上，我們結合 Spark 以及 map-reduce 運算將我們提出的方法其運算過程加以平行化，用以縮短時間，降低複雜度，來改善原方法的程式。

關鍵字：中國暗棋、雲端運算、不完全資訊、等價分類、殘局資料庫

壹、緒論

中國暗棋[4]的研究相當的廣泛，目前已知的研究有從一盤暗棋剛開始時的 Startgame 領域中，透過 open-book 進行[10]，到暗棋中盤 Middlegame 時利用蒙地卡羅搜尋樹[2],[3],[9],[13]和學習[15]輔助 AI 程式判斷要如何下棋，以及對殘局 Endgame, 資料的壓縮跟存放[5],[6],[7],[8], [15]，其中原作者有提到暗棋的殘局數量非常的巨大，而暗棋因為有著在翻開之前不確定是誰的不確定性，往往使得研究困難又有趣，也因為這樣的不確定性，空間複雜度[9]和資料量[7]都十分的龐大，為此，在 2015 年由陳志昌教授提出了針對暗棋龐大的盤面資料作盤面等價的壓縮[7]，作者提出了兩組棋子的盤面在符合 Structural Equivalent 的情況下可以視為等價，並在論文中提出了 Stepwise Refinement Matching Algorithm 找出等價的盤面，也為了改善 SRM 演算法在 239,980,775,397 組含有至少一個蓋著棋子的盤面中找出等價的組合時執行時間太慢的問題，作者也提出了 Accelerated SRM Algorithm 輔助 SRM 演算法，並且成功的化簡了盤面，然而，我們發現原作者的程式有以下問題需要改善：

1. 原作者的程式僅允許兩兩比較，一次只輸入兩組盤面，再透過 SRM 演算法驗證這兩組盤面是否等價，且僅能在單機序列化的環境下執行程式。在找出等價盤面進行合併的過程上，根據作者文章，沒有蓋著棋子的盤面總數有 8,497,176 組，至少含有一個蓋著棋子的盤面則有 239,980,775,397 組，若程式一次只能比較兩組盤面，將花費許多時間才能將所有盤面比較完畢。
2. 原作者的 SRM 演算法在程式的實作上，對於兩組盤面在比較等價的過程中對其中一組盤面使用排列組合，使得時間複雜度非常高，其中 SRM 演算法的時間複雜度為 O(n*n!)。所以原作者在找出完全沒有蓋著的盤面時花了 27.17 小時的時間，若用以處理含有至少一個蓋著的棋子的盤面時，則會耗費更多的時間，遠大於 27.17 小時。
3. 使用 SRM 演算法執行不含蓋著棋子的盤面等價時，因為全部比較完會耗費太多時間，所以作者另外設計 ASRM 演算法先將所有含有蓋著棋子的盤面作處理，再使用 SRM 演算法進行比較。但程式執行上會更複雜，總執行時間也會再拉長。
4. 原作者的實驗環境是在單機的環境上，且程式沒有平行處理，然而棋盤的盤面數量及資料很龐大，僅使用單機執行環境會導致執行時間過長。

為了改善前述原作者在實作上我們所發現的問題，本論文根據原作者的理論提出 Capturing Relation Matrix(簡稱 CRM)，對每一組盤面產生 Material Combination Pattern(縮寫成 MCP)，透過將擁有相同 MCP 的盤面做合併，並使用 hash 表儲存，以利於電腦 AI 程式查詢，相對於原作者對於整個盤面做排列組合，我們參考 CRM 上的值配合適度的排列及交換，可以降低時間複雜度，透過我們的方法可以達到和原作者相同的目的，而在過程中，我們將我們所設計的方法，運用 Spark 將我們的程式執行過程平行化，可以在更進一步的降低時間複雜度。

貳、背景知識

一、暗棋的介紹、玩法及規則及術語

暗棋發源自中國象棋[17]，在 4*8 共 32 個格子中，由兩位玩家遊玩分別代表紅方及黑方，紅方和黑方各 16 個棋子所組成，其中紅方由一個帥、兩個仕、兩個相、兩個俥、兩個傌、兩個炮以及五個兵所組成，黑色部分也由將一個、士、象、車、馬、包各兩個、以及卒五個所組成，在電腦程式裡各個棋子也有屬於自己的英文代號，如圖 1 所示，一開始棋子全數都是蓋著的。

名稱 (紅/黑)	紅方 英文代號	黑方 英文代號	數量	位階
帥/將	K	k	1	1
仕/士	G	g	2	2
相/象	M	m	2	3
俥/車	R	r	2	4
傌/馬	N	n	2	5
炮/包	C	c	2	6
兵/卒	P	p	5	7

圖 1、棋子的英文代號、個數及位階

兩位玩家各執一個顏色輪流進行遊戲，在每一次的行動中，玩家可從以下三種操控中選擇一種進行操作：

(一) 翻牌(flipping action)：
將一個蓋著的棋子翻過來。

(二) 移動/吃棋(moving action)：
將一個翻開的我方棋子移動一格或吃掉位階較小或相等的棋子[1]，兵(卒)也可吃掉將(帥)。

（三）跳吃(jumping action)：
炮或包在直的或橫的方向中，隔著一個棋子有對方的棋子，可以飛越過中間棋子直接吃掉對方的棋子，這個操作不看棋子的位階。

遊戲的勝利條件為其中一方吃掉對方所有的棋子，然而，當雙方連續無吃棋或是翻棋的走步合計到達一個上限，通常為二十五步到五十步之間即為和棋，現行的國際 AI 比賽中則是四十步。而為避免有些玩家徒然浪費時間，也有長捉禁手(或稱為追棋禁手)[1]的規則。

二、Structural Equivalent

在原作者的理論中，首先將要比較的兩個盤面轉換為 Material Combination(簡稱為 MC)，一個 Material Combination 即一個盤面上棋子英文代號的集合，採用紅棋在前面，黑棋在後面的順序，如帥、仕、將的 MC 則寫成 KGk，之後他將其中一組 MC 裡的每一個棋子依照他跟他在同一個 MC 裡敵對的棋子的吃棋關係將每一個棋子加入三個子集合內，分別是 $rel > (x,S)$，即集合裡面的棋子可以被 x 吃掉但是不能吃 x，$rel = (x,S)$，即集合裡棋子和 x 可以互相吃對方，$rel < (x,S)$，即 x 不能吃對方，但是對方可以吃 x，這三個子集合也稱為對該棋子 x 的 Capturing Relation，如果 *S1* 盤面裡的棋子 x 和 *S2* 盤面裡的棋子 y 的三個集合裡元素個數都相等，且符合：

1. x 和 y 同顏色
2. x 和 y 都是開著或都是蓋著的
3. x 和 y 都是炮/包，或都不是炮/包

則 x 和 y 稱為 Capturing Equivalent，如圖 2 所示。

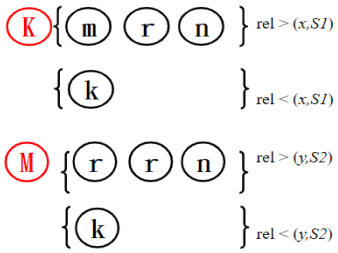

圖 2、K 和 M 符合 Capturing Equivalent

如果所有 *S1* 裡的每一個棋子都可以找到一個在 *S2* 裡面的棋子是符合 Capturing Equivalent 的，即 *S1* 可以透過一對一映成函數找到在 *S2* 裡對應得棋子，則稱這兩組盤面符合 MC-mapping。在原作者所定義的兩個盤面的等價中，兩個盤面除了要符合 MC-mapping 之外，透過映成函數找到在兩個盤面上的所有棋子中，作者還規定，當他們被放在兩個棋盤上面相對應的位置時，在接下來的遊戲中，對於 *S1* 裡面的任一個棋子可以執行的操作，透過一對一映成函數在 *S2* 裡面找到的棋子也要能夠執行，這個過程作者將之定義為 Isomorphic，若兩組盤面符合符合上述 MC-mapping 以及 Isomorphic 的定義，即為作者定義的等價，如圖 3 所示。

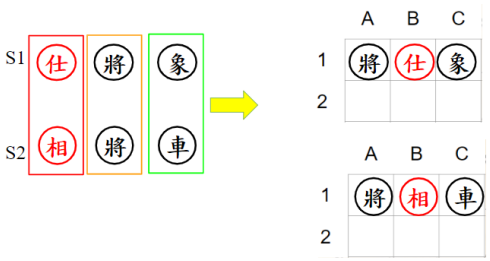

圖 3、符合 Structural Equivalent 的例子

三、Capturing Relation Graph 介紹

Capturing Relation Graph (CRG)是透過圖論中的同構來驗證兩個 MC 是否等價，在 CRG 圖裡，我們將每一個棋子視為一個頂點，頂點中記錄著他的 Attribute Identification Symbol (AIS) 、AIS 可能有以下四種可能，分別是 $\overline{rc}$、$\overline{rc}$、$r\overline{c}$、以及 rc，其中 r 表示這個棋子是開著的，反之則用 $\overline{r}$ 表示，c 則表示該棋子為炮/包，每個頂點會對所有連出去的敵對顏色的頂點延伸出一條 Edge，分別對應前面提到的三種吃棋關係，如果我方可以吃對方，而對方不能吃我方，以→表示，如果雙方可以互吃，則用↔表示，最後，如果只有對方可以吃我方，則用←表示，圖 4 為一個 CRG 的例子。

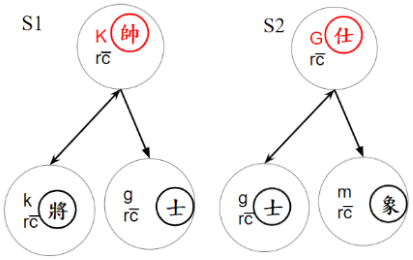

圖 4、S1 和 S2 盤面因 CRG 相同而等價的例子

作者將兩組盤面做成 CRG，並比較在兩個盤面中被一對一映成函數配對的各頂點的 AIS 值以及對敵對所有一樣被一對一映成函數配對的頂點之間的 Edge 是否一

樣，如果一樣就直接將兩組盤面加入一個集合，如果不一樣，作者會固定其中一組盤面，並對另一組盤面做排列組合，再做一次比較，如果排列組合到有一個順序可以跟固定起來盤面的 CRG 對上，則兩組盤面等價，如果所有的盤面的排列組合都已經被比較過但是兩組盤面 CRG 還是不同構，就稱該兩組盤面不等價。

四、Map-Reduce 及 Spark 介紹

Map-Reduce[12]是一個由 google 提出的軟體架構，主要用於大數據平行運算，透過 Map(映射)函數生成多個鍵值對，再交由 Reduce(歸納)函數歸類及合併，本論文藉由 Map-Reduce 函數將我們的方法平行化，可以再更加速我們的程式執行過程，降低時間複雜度。

Spark[16] 是一個開源的叢集運算框架，並延伸了流行的 Map-Reduce 運算框架並提供其他高效率的計算應用，基於記憶體內的計算框架。Spark 在運算時，將中間產生的資料暫存在記憶體中，因此可以加快執行速度。尤其需要反覆操作的次數越多，所需讀取的資料量越大，則越能看出 Spark 的效能。本論文透過利用 Spark 執行 Map-Reduce 運算，希望可以再更進一步將我們的程式執行過程加速。

參、運用 Spark 及 Capturing Relation Pattern 化簡盤面.

一、Capturing Relation Matrix

我們根據原作者提出的 Structural Equivalent 的觀念，將原作者 Capturing Relation 的觀念 [7]改良成 CRM，如圖 5 所示，每一行都代表著一個紅色的棋子，每一列則是黑色棋子，行列所對應的元素則為其對應的紅棋和黑棋之間的 Capturing Relation，如圖 5 中的四個元素分別紀錄的分別為帥對將，帥對象，仕對將以及仕對象的吃棋關係，根據暗棋的規則，如果該棋子只可以被對應的黑棋吃掉，但是自己沒辦法吃對方時(如圖 5 紅仕對應黑將)，記錄為 001，如果這個紅棋可以吃對方，但是也會被對方吃掉(如圖 5 紅帥對應黑將)，則記錄為 010，最後，如果這個紅棋可以吃對方，而對方不能夠吃掉自己時(如圖 5 紅仕對應黑象)，則會記錄為 100。

在所有的對應的格子都記錄完後，我們會將每一行矩陣中紀錄的值往下累加，就可以得到這一行對應的紅色棋子的威脅值，如果將每一列矩陣中的結果往右累加，則會得到該一列對應黑色棋子的威脅值反向，因為在這當中我們是以紅色棋子的吃棋關係紀錄在矩陣中，因此黑色的棋子還需要將我們累加所得到的結果進行倒置處理才會獲得該黑色棋子真正的威脅值，完成的矩陣及累加結果如圖 5 所示。

二、Material Combination Pattern (MCP)

在產生完了 Capturing Relation Matrix 後，我們將每一行紅色的結果累加完後，在每一行結果前面我們多加入兩個位元，第一個字元是這個棋子是不是 Cannon(炮/包)，是則填入 1，反之填入 0，第二個字元是判斷該棋子是不是被翻開的，翻開的棋子我們填入 0，反之則為 1，再對每一列後面黑色棋子的結果也做一樣的動作，這樣做的目的是，因為開著的棋子跟蓋著的棋子在遊戲中能做的操作是不一樣的，故開著的棋子不能等價於蓋著的棋子。此外炮/包在遊戲中吃棋子的方式也和其他棋子不同，因此即便他們有相同的威脅值，也依然不是等價的，我們上面所說的每個元素進行排序，然後將排序後的結果以紅色在前，黑色在後的順序進行串接。而後，為了確實保證該盤面的結果符合 Structural Equivalent 的原則，原作者利用比對兩個 Material Combination 的 CRG，而我們首先在剛才排序後的結果後面加上一個辨識字元，以作為區隔，再依照排序的結果，從第一個紅色開始，對所有黑色的棋子以排序下來的順序依照他們在矩陣內對應的值依序填在辨識的字元後面，如此便可以產生一組 Material Combination Pattern，這個動作即對應原作者論文中的 Edge [7]。

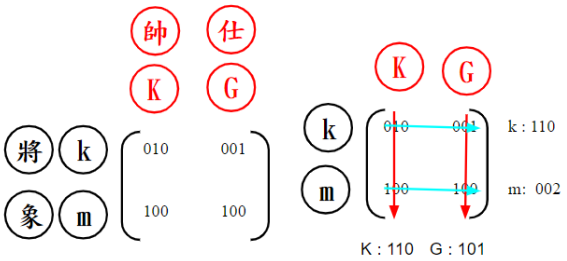

圖 5、(A)以 KGkm 為例產生的 CRM 圖 5、(B)以 CRM 推出 KGkm 的 Threat Relation

以 KG(k)m 為例，在我們的方法當中，我們先依照輸入的資訊產生矩陣，對行列累加後在前面加上位元，如圖 6 所示，之後將紅色的部分排序，得到 0010100110，再將黑色的部分排序得到 0020001011，而後接在一起得到 00101001100020001011，之後補上字元，而在這個例子中，紅色以及黑色的部分分別都沒有重複的值，故我們不進行重複部分的排列組合，而排序之後的結果我們得到 GKm(k)，最後我們再字元的後面依照 Gm、G(k)、Km、K(k)的順序查詢 CRM，得到的結果，如圖 6 所示。

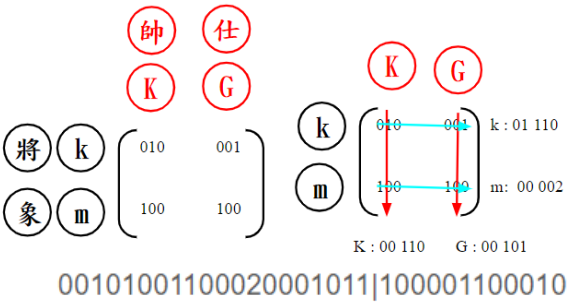

圖 6、以 KG(k)m 為例子產生的 Pattern

若在排序的過程中，發現紅色當中，或是黑色當中，有重複的值出現，我們在特殊字元後面要根據排列順序查表時便會針對有重複的部分，先做排列組合，然後，再將各排列組合後的順序，做威脅值的查表，如此便形成了兩組不一樣的威脅字串，之所以要換位置，其主要的原因是由於雖然兩個元素的開蓋狀態、是否為炮/包以及總威脅值都相同，但其威脅值可能來自於不同的棋子，如圖 7 例子所示，圖中的 G 跟 P 具有一樣的值都是 00101，但是 G 的 100 是來自黑色的 m 而 P 的 100 則

是來自於 k，因此我們會將所具有相同值的元素進行排列組合，在這個例子中，我們會得到兩組 MCS，分別為：
00101001010011000111002 01|001100100001010100 以及
00101001010011000111002 01|100001001100010100。

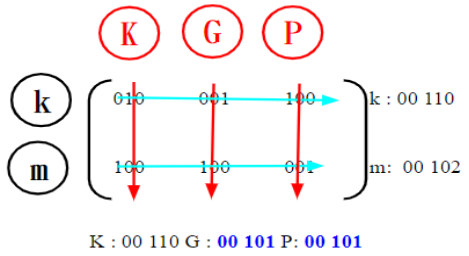

圖 7、以 KGPkm 當作例子所產生的 CRM

我們將所有的 Pattern 產生完後，會將之儲存並記錄該組字串原本是哪一個盤面的，而後我們會將擁有相同 Pattern 的進行合併並且儲存，並且利用特殊字元前面的字串將有相同字串的盤面再做一次過濾，以過濾掉重複的盤面，將資料庫做壓縮。

三、運用 Spark 平行化產生 Pattern

我們將上述產生字串的過程利用數個 Map-Reduce 加以平行處理，以加快產生盤面字串的時間，並且在最後透過 Reduce 將相同 Pattern 的結果合併起來，以達到原作者等價合併的結果。

肆、實驗結果與討論

我們將上述提到的方法套用在暗棋的 Endgame 上做出棋子數量和演算法執行的時間圖，如圖 8 所示，根據我們的方法，透過 Swapping Sequence 相對於全部數量的排列組合，可以讓我們在執行階段減少執行的時間。

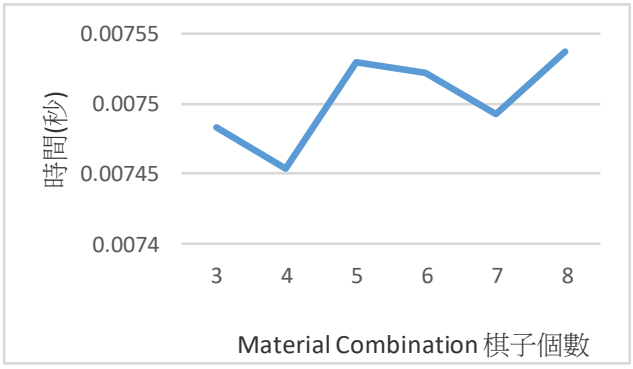

圖 8、使用 Spark 平行在 endgame 各盤面所花費時間

伍、結論與未來展望

本論文提出了 Parallel Refinement Pattern Generating Algorithm 用以改善原作者比較棋盤盤面等價的方法，將 Threat-Table 所獲得的結果加以排序並串接成 Pattern，最後將相同的盤面合併已達成盤面等價化簡的目的，整個過程中我們採用 Spark 搭配 map-reduce 的方法平行化，以加速整個程式執行的過程，原作者在文章中提到在處理暗棋 Endgame 盤面一共 144,158 組資料時花了 11.1 秒，經過我們的方法實測，同樣的盤面等價比較我們花費了 0.045 秒，可以大幅縮短程式的執行時間，化簡之後的結果可以壓縮盤面的資料庫，未來希望可以真正去輔助電腦在執行遊戲搜尋樹的決策過程。

致謝

本論文承蒙科技部專題研究計畫 MOST107-2218-E-018-001 經費補助，特此申謝。

References

[1] 謝政孝，“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，2010。

[2] C. H. Hsueh, I. C. Wu, W. J. Tseng, S. J. Yen, J. C. Chen. An analysis for strength improvement of an MCTS-based program playing Chinese dark chess. Theor. Comput. Sci, vol 644 , pp. 63-75 ,2016

[3] C. Browne, E. Powley, D. Whitehouse, S. Lucas,P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis and S. Colton. A surveyof Monte Carlo Tree Search. IEEE Trans. Comput. Intell. AI in Games,vol. 4, no. 1,pp.1-43, Mar. 2012.

[4] B. N. Chen, B. J. Shen, and T. s. Hsu. Chinese dark chess. ICGA J.,vol. 33, no. 2, pp. 93–106, 2010.

[5] B. N. Chen, H. J. Chang, S. C. Hsu, J. C. Chen, T. S. Hsu. Advanced Meta-knowledge for Chinese Chess Endgame. ICGA Journal, vol 37(1), pp. 17-24 ,2014

[6] H. J. Chang, J. C. Chen, C. W. Hsueh, T. S. Hsu, “Analysis and efficient solutions for 2×4 Chinese dark chess. ICGA Journal, vol. 40, no. 2, pp. 61-76, 2018.

[7] J. C. Chen, T. Y. Lin, B. N. Chen, and T. S. Hsu. Equivalence classesin chinese dark chess endgames. IEEE Trans. Comput. Intell. AI in Games (TCIAIG), vol. 7, no. 2, pp. 109–122, 2015.

[8] B. N. Chen, H. J. Chang, S. C. Hsu, J. C. Chen and T. S. Hsu. Multi-level inference in chinese chess endgame knowledge bases. ICGA Journal, vol. 36, no. 4, pp. 203–214, 2013.

[9] S. J. Yen, C. W. Chou, J. C. Chen , I. C. Wu, K. Y. Kao. Design and implementation of Chinese dark chess programs. IEEE Trans. Computat. Intell. AI Games 7(1), pp. 66–74, 2015.

[10] B. N. Chen, T. S. Hsu, Automatic generation of opening books for dark chess. In: van denHerik, H., Iida, H., Plaat, A. (eds.) CG 2013. LNCS, vol. 8427, pp. 221–232. 2014.

[11] Q. Gao, X. Xu, Research on the Computational Complexity of n x n Chinese Chess. ICGA Journal, vol. 38, no. 1, pp. 47-53, 2015.

[12] J. Dean, S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. OSDI'04: Sixth Symposium on Operating System Design and Implementation, pp. 137-150, 2004.

[13] H. Finnsson, Generalized Monte-Carlo tree search extensions for general game playing. 26th AAAI Conference on Artificial Intelligence. pp. 1550-1556, 2012.

[14] H. J. vanden Herik, J. W. H. M. Uiterwijk and J.van Rijswijck. Games solved: Now and in the future,”Artif. Intell. vol. 134, pp.277–311, 2002.

[15] A. Saffidine, N. Jouabndeau, C. Buron, and T. Cazenave, Material Symmetry to Partition Endgame Tables. Yokohama, Japan: SpringerLNCS, pp. 187–198, 2013.

[16] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica. Spark: cluster computing with working sets. In Proc. Hot Cloud ’10, 2010.

[17] S. J. Yen, J. C. Chen, T. N. Yang and S. C. Hsu. COMPUTER CHINESE CHESS. ICGA. 27(1):3-18 · March 2004.