行政院國家科學委員會補助專題研究計畫 ■ **成 果 報 告**
□**期中進度報告**

六子棋詰棋及開局定石之自動產生系統之研究與設計

計畫類別：■ 個別型計畫　　□ 整合型計畫
計畫編號： NSC 97-2221-E-009 -126 -MY3
執行期間： 97 年 8 月 1 日至 100 年 7 月 31 日

計畫主持人：吳毅成
共同主持人：
計畫參與人員：林秉宏、孫德中、林宏軒、陳柏廷、鄒忻芸、王智功、
陳俊嶧、楊景元、蔡心迪、林正宏、康浩華、陳干越

成果報告類型(依經費核定清單規定繳交)：□精簡報告　■完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
■出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、
列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

執行單位：國立交通大學資訊工程學系
中 華 民 國 一 百 年 十 月 二十五 日

中文摘要

自我們發表全世界第一篇六子棋論文至今僅五年多，六子棋的歷史與其他棋種如象棋、圍棋、西洋棋有數百年、甚至上千年歷史相比，仍屬相當年輕的，因此專家的詰棋題目與開局定石的數量仍相對較為缺乏。

為了加速六子棋的推廣與普及，本計畫（目前正在執行中）除了持續研發新的六子棋搜尋技術外，我們已發展出開局定石自動產生之系統與技術，此系統提供 GUI 介面供棋士便於建立開局庫，並可用來大量產生開局定石，提供六子棋高段棋士研究更深入的各種開局下法。由於開局定石產生系統相當耗時，我們使用平行處理技術來加快處理速度，並計劃將原本的 Proof Number Search 演算法改進成適用於平行化的架構上。此結果發表於 IEEE Transactions on Computational Intelligence and AI Games 期刊，以及本領域最重要的國際會議 International Conference on Computers and Games 2010。

此外，為了瞭解每手棋下 p 子的必勝、必敗、必和的結果，我們發展新的技巧算出許多新的結果，例如每手棋下 2 子，則 11 子棋必和，此結果遠優於現有結果(15 子棋必和)，並發表於 Theoretical Computer Science 期刊。

關鍵字詞：六子棋、詰棋、開局定石、詰棋產生器、開局定石產生器、搜尋技術、迫著策略。

英文摘要

When compared with other games such as Go, Chinese Chess, Chess, the game Connect6 has still been young since we presented the first article about Connect6 four years ago. Therefore, puzzles and openings offered by experts are relatively insufficient.

In order to expedite the promotion of Connect6, this project has developed automatic opening generators, in addition to the search technology for Connect6. Also this project provides GUI interface for Connect6 players to help building the opening. The automatic opening generator generates a large number of openings that players can learn how to play in the openings. Since it takes a huge amount of time in automatic opening generator, we also developed parallel techniques to speed it up. We are planning modifying Proof Number Search to fit the parallel structure.

In addition, in order to understand better about the theoretical values of general Connect games, we investigate those drawn Connect games where p stones are placed in each move. In the case of p=2, we obtain that Connect 11 games is a draw game. The result is far better than the current result (15).

Keywords: Connect6, puzzles, puzzle generator, openings, opening generator, threat-based proof search

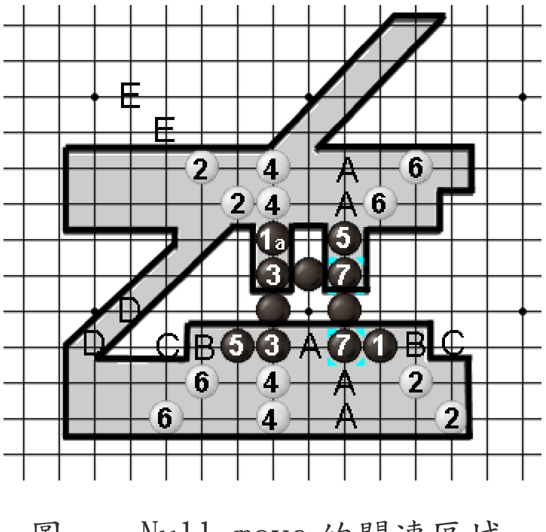
一、計畫說明

本計畫主持人除了第一個正式提出六子棋下法，並研發許多新的六子棋搜尋技術，及棋力強大的六子棋程式「交大六號」。除了獲得國際奧林匹亞六子棋組賽局競賽的冠軍外，最近亦曾有擊敗某次六子棋公開賽冠軍棋士的記錄。

然而自我們發表全世界第一篇六子棋論文至今僅四年多，六子棋的歷史與其他棋種如象棋、圍棋、西洋棋有數百年、甚至上千年歷史相比，仍屬相當年輕的，因此專家的詰棋題目與開局定石的數量仍相對較為缺乏。

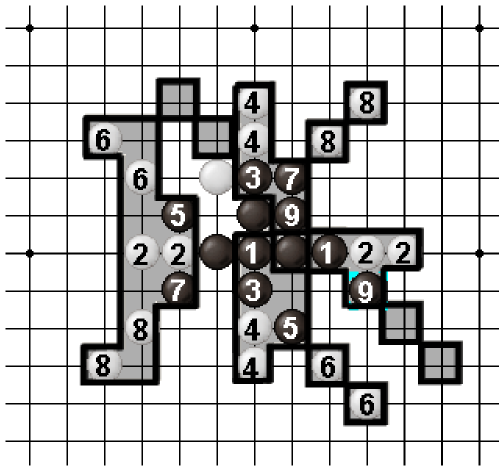
為了加速六子棋的推廣與普及，此計畫（三年期計畫，目前是第二年）除了持續研發新的六子棋搜尋技術外，我們已發展出開局定石自動產生之系統與技術，此系統提供 GUI 介面供棋士便於建立開局庫，並可用來大量產生開局定石，提供六子棋高段棋士研究更深入的各種開局下法。由於開局定石產生系統相當耗時，我們使用平行處理技術來加快處理速度，並計劃將原本的 Proof Number Search 演算法改進成適用於平行化的架構上。

在六子棋搜尋技術方面，我們持續改良我們的六子棋程式「交大六號」，其中一項是，提出一種是以關連區域(Relevance Zones)為主的搜尋技術，這是利用對手下出的空著(Null Move)或半空著(Semi-Null Move)所獲勝後的關連區域，來去做迫著空間搜尋(Threat Space Search)。



圖一　Null move 的關連區域

例如：盤面有黑三子如上圖圖一，很明顯是黑勝，但由於整個棋盤我們假設是非常大或甚至是無窮大，我們無法一一搜尋來證明。於是我們需要先假設對方不下(也就是所謂 null-move)，我們先找到一個活四追四勝（或簡稱追四勝)如上圖，在這下法中找出一個區域(叫做 Relevance-Zone 或簡稱 R-Zone）是對手可能有機會擋住的位置；如在上圖棋譜中，陰影部份是指白若要阻擋黑的追四勝，二子之其中一子必下的 R-Zone，詳見(Wu and Huang, 2005)。

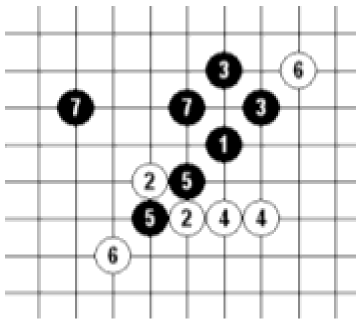


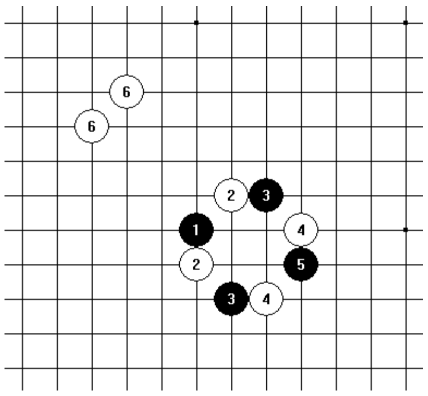
圖二　Semi-null move 的關連區域

設若白之一子下於棋譜一中 1a 的位置，則再假設其另一個子不下(semi-null-move)，從我們追四勝的下法中找出一個 R-Zone 是對手可能有機會擋住的位置，如圖二，陰影部份是指白若要阻擋黑勝，另一子必下的 R-Zone。

然後檢視所有擋法後，全為黑勝。這麼一來，我們可以證明出原始黑三子的盤面，為黑必勝。這就是我們所謂的以關連區域(Relevance Zones)為主的搜尋技術。

在圖一及圖二中，所有阻擋的位置，如白 2、白 4 等，我們都用保守下法（一手下四子)；但有些棋型，若不是保守下法才能獲勝，關連區域會更為複雜。甚至更複雜的是：若需要利用到死四（或稱單迫著）或無四（或稱無迫著）的追四勝的話，則證明會是更為複雜。


圖三　TX-d21 開局

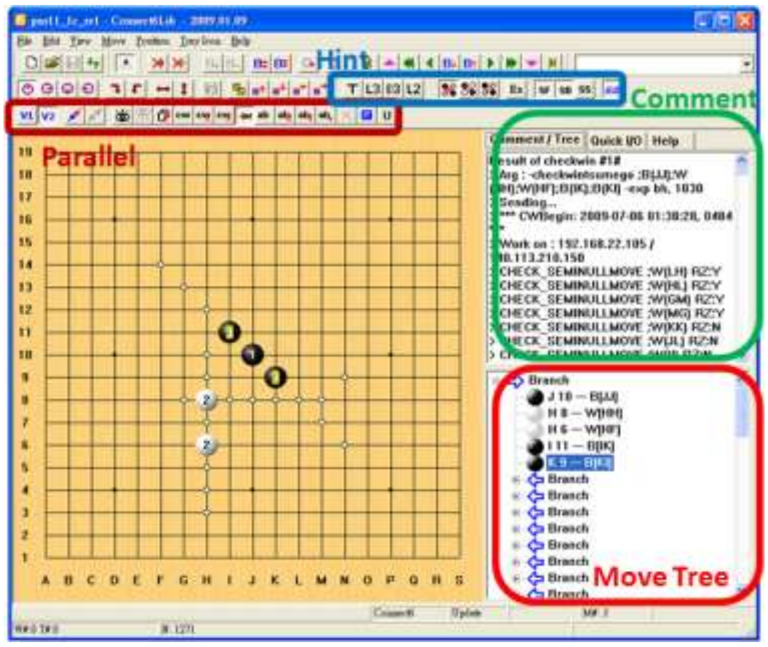

圖四　Semi-null move 的關連區域

例如對上圖圖三，在處理 Semi-null-move 時，若白下一子於白４的右方一格，此時黑無法用雙追著追四勝，必須用單追著追四勝才能獲勝。甚至，如圖四中的棋型（註：此棋型是在討論若黑５只下一子時的狀況)，若黑下於白６右上及右下一格時，白連單追著都無法獲勝，必須是無追著才能得勝。然而單追著追四勝及無追著追四勝的關連區域，並不容易找出。

本計畫發展出一套新的演算法稱為 Threat Proof Search (TPS)，算出這個關連區域。此 TPS 演算法對關連區域，會依照不下的子數來發展出不同的區域。例如，若一子不下的關連區域稱為 Z1; 若二子不下的關連區域稱為 Z2; 若三子不下的關連區域稱為 Z3。利用這些區域，我們可以證明出圖三及圖四的棋型是黑必勝。

在開局定石產生系統方面，我們除了利用 TPS 的搜尋技術外，我們亦使用 Proof Number Search (PNS)技巧來加速開局的驗證。由於我們無法預期各個盤面是必勝或必敗或和，便使用 PNS 技巧找出如何搜尋能最快達成目標。此目標可能是驗證出必勝，也可能是驗證出必敗，甚至是某種程度上證明黑白雙方已經接近和局。目前正在發展中，並已有初步成果。

為了配合開局定石產生系統，我們的 AI 設計出了兩個功能：alpha-beta 搜尋與 Verifier 驗證。在 alpha-beta 搜尋中會利用 AI 找出一個最佳著手，而 Verifier 則是會列出此盤面的所有可能擋法。

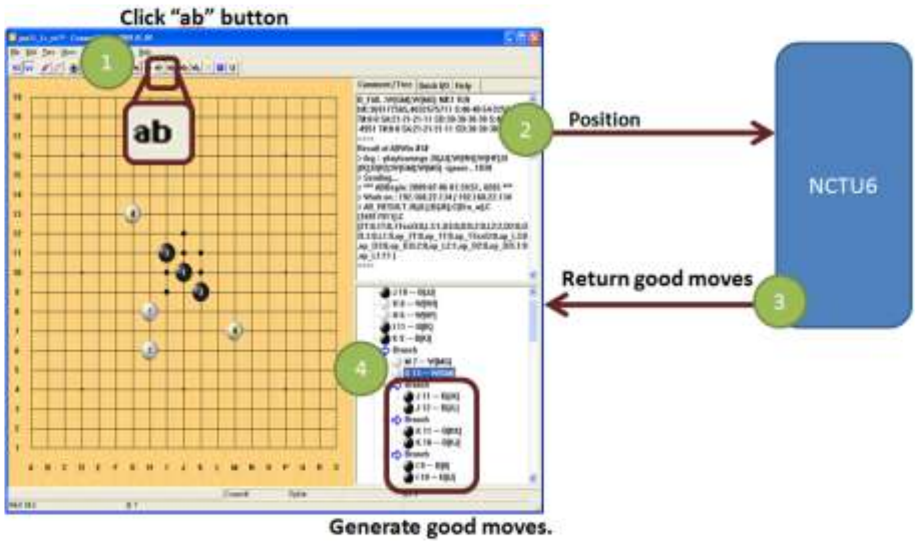


```
Result of checkwin #1#
Arg : -checkwintsumego ;B[JJ];W
[HH];W[HF];B[IK];B[KI] -exp bh, 1030
Sending...
*** CWBegin: 2009-07-06 01:38:28, 0484
**
Work on : 192.168.22.105 /
10.113.210.150
CHECK_SEMINULLMOVE ;W[LH] RZ:Y
CHECK_SEMINULLMOVE ;W[HL] RZ:Y
CHECK_SEMINULLMOVE ;W[GM] RZ:Y
CHECK_SEMINULLMOVE ;W[MG] RZ:Y
CHECK_SEMINULLMOVE ;W[KK] RZ:N
CHECK_SEMINULLMOVE ;W[JL] RZ:N
```

```
Branch
  J 10 — B[JJ]
  H 8 — W[HH]
  H 6 — W[HF]
  I 11 — B[IK]
  K 9 — B[KI]
  Branch
  Branch
  Branch
  Branch
  Branch
  Branch
  Branch
  Branch
```

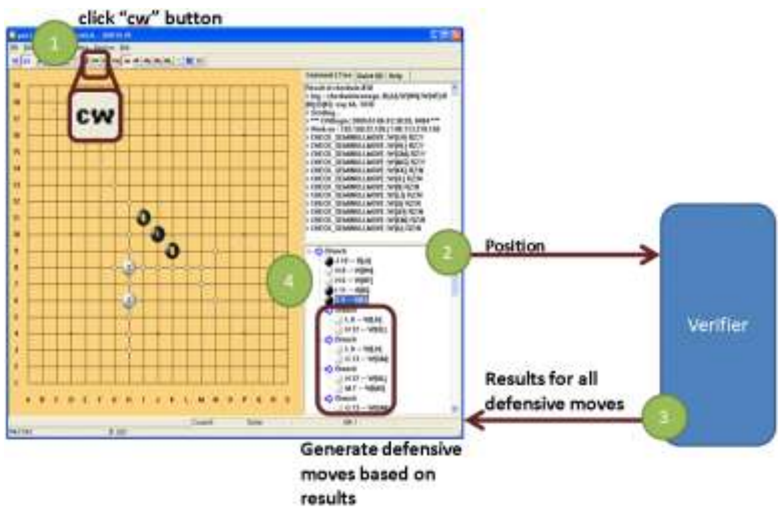
圖五　開局定石系統的 GUI 介面

圖五是我們開局系統的 GUI 介面，此介面程式改良自 Connect6Lib，此介面有許多基本下棋功能，如：盤面展現、記錄下棋的樹狀結構、使用者的註解欄位與迫著的提示等等。利用此介面我們還可連結後端的 AI 程式，啟用 AI 程式的許多功能。



圖六　呼叫 NCTU6 的 AI 搜尋

在開局庫系統中最常使用到的 AI 功能之一即是 AI 搜尋。當我們在建立開局庫的過程中遇到困難的盤面，不知道如何著手時，可以呼叫後端的 NCTU6 程式做運算，如圖六中按下 ab 按紐，此時開局庫系統會將盤面丟到 NCTU6 並啟動 NCTU6 的 alpha-beta 搜尋，當 NCTU6 搜尋完之後會將結果傳回前端並即時地在 GUI 介面中呈現 AI 搜尋的最佳著手(如圖六的第 4 步驟,搜尋結果會以樹狀方式展現)。


click "cw" button
1
cw
2
Position
Verifier
Results for all defensive moves
3
4
Generate defensive moves based on results

圖七 呼叫 Verifier 做驗證

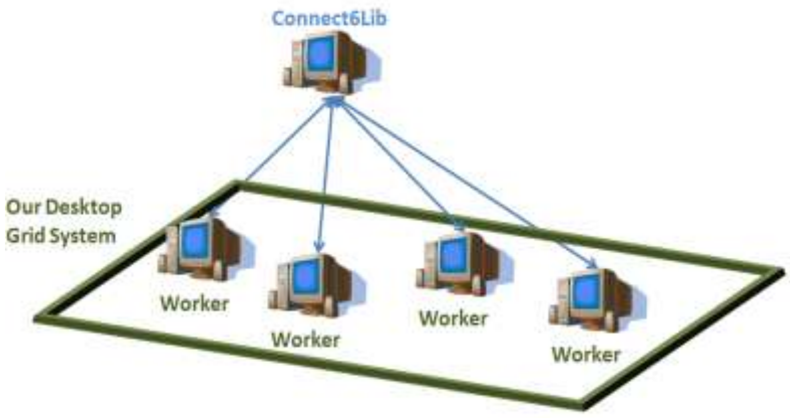
而當我們走到一個盤面，希望驗證此盤面是否已經必勝時(如圖七，黑 3 剛下完，我們希望能驗證此盤面是否是黑必勝)，則可以啟動 Verifier 功能。Verifier 會列出下一步白的所有可能擋法，若白存在有一個以上擋法則代表此盤面黑方仍未必勝，必須繼續深入做驗證。反之，若 Verifier 驗證完沒有顯示任何擋法，則代表此盤面所有白色著手都擋不住，也就是此棋局黑必勝。

因此我們的自動化開局定石產生系統則是利用這兩個功能，當我們需要建立黑方開局庫時，就利用 ab 的功能找出一手黑方最佳著手，在使用 Verifier 驗證白方所有可能的擋法，黑方再針對這些所有可能的擋法，利用 ab 功能找出最佳的一個攻擊著手。如此反覆循環，當任何白方可能的走法黑都可以必勝時，一個黑方的開局庫已建立完成，此開局庫同時也包含了各種黑的必勝下法。

由於建立開局系統非常費時，許多棋局需要花費數十天甚至數個月或數年之久，因此我們使用平行化系統來加速運算。在原本的 GUI 介面上，只要資源充足，我們可以呼叫數個以上的 NCTU6 或 Verifier 程式來做運算。比如說圖七的盤面在經過 Verifier 驗證之後產生了數個白方的擋法，我們可以在同時

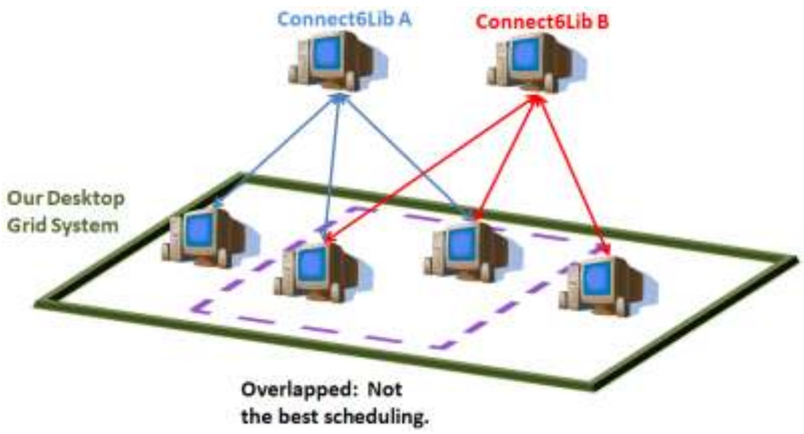
間呼叫多台電腦執行 NCTU6 替每種擋法都產生一個黑方的最佳著手，以達成平行化運算的目的。

為了平行化的順利，我們初期使用的架購圖如下圖圖八所示，每當 GUI 介面(圖八中的 Connect6Lib)需要做運算時，將工作分配給後端的電腦(圖八中的 Workers)，Work 運算完之後會將結果傳回 Connect6Lib 由 Connect6Lib 做整合。



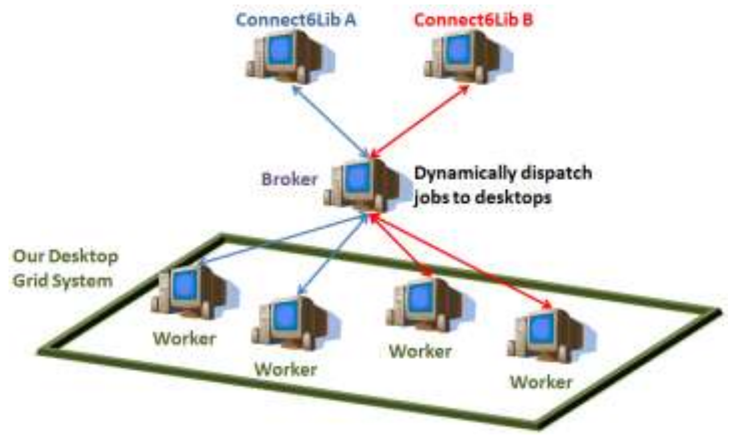
圖八　Desktop grid 架構圖

但是這個架構會產生一個問題，當我們有多個 Connect6Lib 時，會產生如下圖圖 9 的現象，多個 Connect6Lib 會呼叫到同樣的 Worker，也就是資源使用不均的狀況發生。這情形當 Connect6Lib 越多時越嚴重，最慘的狀況是全部 Connect6Lib 都使用到少數同樣的 Workers，而有些 Workers 則完全閒置，這樣會嚴重影響平行化的效率。



圖九　使用到同樣的 Workers

因此最後我們的架構圖如圖十所示，在 Connect6Lib 與 Workers 之間加入 Broker 做協調，Broker 會妥善分配工作，將工作平均分配給每個 Worker 運算，這樣可以解決資源使用不均的狀況。



圖十　使用 Broker 做協調

除了預設由 Broker 自動分配之外，每個 Connect6Lib 可以透過 Broker 看到目前 Worker 的使用情形，也可以透過 Broker 設定自己希望使用幾台電腦做運算，或設定希望固定用哪幾台 Worker 等等。這些設定及運作都會由 Broker 做協調，避免錯誤的發生。

由於原本的 Proof Number Search 只適用在單一台電腦的運算上面，不適用於平形化系統，因此接下來我們要研究的是如何開平行化的架構上發展出一個類似 Proof Number Search 的演算法，來加快我們的平行化效率。

而在詰棋搜尋方面，我們首先利用我們六子棋程式「交大六號」找出大部分必勝棋型的必勝下法。進而利用一般的 Iterative deepening 技巧找出最短的必勝下法。此計畫亦研究分析詰棋的品質，及難易度。例如：即使可以找出某個棋型的必勝法則，然而還需要找出最短必勝下法；即使可以找出最短的必勝下法，還需要找對棋友最感到艱困的必勝下法。此外，若這個棋型可能有很多必勝下法，這樣個棋型，也不一定適合成為詰棋，必須過濾，以確保詰棋品質。

二、研究成果

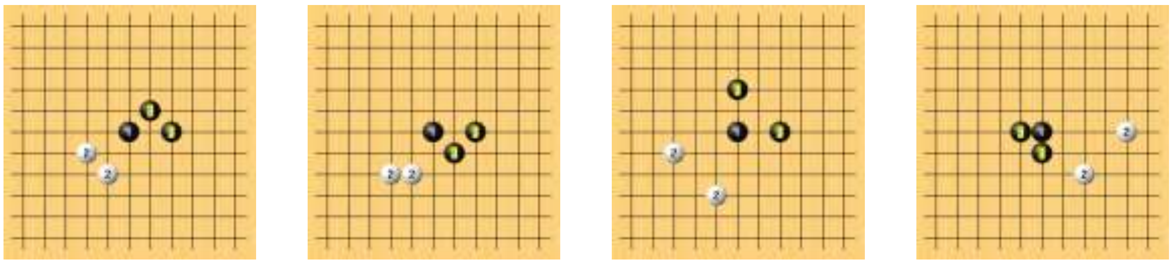
如上所述，我們發展一套新的演算法稱為 Threat Proof Search (TPS)。此 TPS 演算法被用於提升了「交大六號」的搜尋能力，這同時也大幅提升了「交大六號」棋力，戰績如下。

- 在 2008 年 10 月的國際奧林匹亞賽局競賽中的六子棋程式比賽中，在眾多隊伍中脫穎而出，獲得金牌。這為我國獲得唯一的一面金牌及獎牌。

- 在 2008 年第一屆人腦對電腦六子棋大賽中，以 11 勝 1 負，幾乎大獲全勝。此比賽的棋士代表，均為國內一流高手，如下：
  - 第三屆交通大學盃六子棋公開賽前三名：陳威翰、游智翔、林皇羽。
  - CYC Online contest 第一名：林承毅。
  - 國際 Littlegolem.net 網站上一屆總冠軍：黃于峻。
  - 五子棋資深高段棋士（2003 年首屆五子棋亞洲杯大賽亞軍、擔任三屆交大盃裁判及裁判長）：張益豐。
- 在 2009 年第二屆人腦對電腦六子棋大賽中，以 8 勝 0 負，大獲國內高手如下：
  - 第四屆交通大學盃六子棋公開賽第一、三、四名：許紋菁、陳鎮國、陳威翰。
  - 六子棋高段棋士：李士文（六子棋聯誼會會長、五子棋高段棋士）。

以上的戰績，顯示這項研究，獲得相當明確的成果。

另外，我們正在實作一套可以自動產生開局庫的平行化系統，此系統可調用 NCTU6 與 Verifier 做搜尋與驗證。透過此系統，我們已初步可證明出如圖十一所示的幾個盤面皆為黑必勝。也就是說當黑第一手下在天元之後，白色不能下在這幾個地方，否則黑靠第三手的下法就可以贏得棋局。這些都是過去棋士所不能證明出的，令目前高段棋士十分震驚的六子棋開局定石。



圖十一　黑必勝盤面

在詰棋搜尋方面，我們發展一套六子棋的詰棋自動產生系統。目前，此部份計畫仍在持續發展中，但已經收集相當多的題目約 30000 多題，並已初步過濾出 5000 多題。此計畫將持續進一步優化詰棋品質，如找出最短必勝下法、找出最艱困的必勝下法，並避免太多重複解。在 2008, 2009 年的第三、四屆交大盃六子棋公開賽大中，我們已經開始提出一些題庫試用。

三、結論

我們發展一套 TPS 演算法，這除了對程式的搜尋能力有大幅的提升外，亦可用於開局及詰棋自動產生系統。並設計了平行化開局庫自動產生系統，可自動建立開局庫。目前的研究成果顯示此計畫的可行性及效果。我們不僅提出全世界第一篇六子棋玩法的論文，本計畫亦對六子棋發展出許多重要的研究，如開局及詰棋。由於六子棋的規則簡單、遊戲公平、及玩法複雜，有機會成為一項由台灣研究發展出而普及全世界的遊戲。本計畫的成功將有助於六子棋的發展與推廣，及提升國家形象及知名度。

四、計畫成果自評部份

從上述的研究成果，我們可以了解此計畫有相當豐碩的成果如下：

- **獲得第十三屆國際奧林匹亞賽局競賽冠軍，為我國獲得唯一的一面金牌及獎牌，同時也擊敗台灣許多六子棋高段棋士。**
- **我們發展出許多令目前高段棋士十分震驚的六子棋開局定石(如圖十一)，我們預期加上完全平行化的自動開局產生系統，將會有更多前所未見的開局定石。**
- **許多成果發表於重要期刊與會議如: IEEE Transactions on Computational Intelligence and AI Games 期刊, Theoretical Computer Science 期刊, ICGA Journal 期刊, 以及本領域最重要的國際會議 International Conference on Computers and Games 2010.**

因此，我們很確信地自評：**此計畫的執行成果相當優異**。我們附上以下論文:

- I-Chen Wu and Ping-Hung, Lin, "NCTU6-Lite Wins Connect6 Tournament", ICGA Journal (SCI), Vol.31, No.4, December 2008.
- Ping-Hung Lin and I-Chen Wu, "NCTU6 Wins in the Man-Machine Connect6 Championship 2009", ICGA Journal (SCI), vol. 32(4), 2009.
- I-Chen Wu, H.-H. Lin, P.-H. Lin, D.-J. Sun, Y.-C. Chan and B.-T. Chen, "Job-Level Proof-Number Search for Connect6", The International Conference on Computers and Games (CG 2010), Kanazawa, Japan, September 2010.
- I-Chen Wu and Ping-Hung Lin, "Relevance-Zone-Oriented Proof Search for Connect6", the IEEE Transactions on Computational Intelligence and AI in Games (SCI), Vol. 2, No. 3, pp. 191-207, September 2010.
- Sheng-Hao Chiang, I-Chen Wu, Ping-Hung Lin, "Drawn K-In-A-Row Games", Theoretical Computer Science (SCI), Vol. 412, pp. 4558-4569, August 2011.

NCTU6-LITE WINS CONNECT6 TOURNAMENT

I-Chen Wu[1] *and Ping-Hung Lin*[1]

Taiwan

The computer Connect6 tournament was held as part of the 13th Computer Olympiad, which took place in Beijing, China, from September 29th to October 1st, 2008. Ten teams participated in the Connect6 tournament. Table 1 lists the participants and the final standings.

Ranking	Program	Author	Organization	Points
1	NCTU6-LITE	Ping-Hung Lin, Hong-Xuan Lin, Yi-Chih Chan, Ching-Ping Chen and I-Chen Wu	National Chiao Tung University, Taiwan.	17
2	BITSTRONGER	Liang Li, Hao Cui, Ruijian Wang and Siran Lin	Beijing Institute of Technology, China	16
3	NEUCONN6	Chang-Ming Xu	Northeastern University, China	13
4	BEAD CONNECT AND CHESS COMBINE (BCCC)	XiaoChuan Zhang	Chongqing Institute of Technology, China	9
5	KAVALAN	Jung-Kuei Yang and Shi-Jim Yen	National Dong Hwa University, Taiwan	8
6	NEU6STAR	Xinhe Xu, Dongxu Huang, Junjie Tao, Kang Han, XinXing Li	Northeastern University, China	8
7	ML	Jiang Ke	Guilin University of Electronic Technology, China	6.5
8	CV6	Yao Yuping	Guilin University of Electronic Technology, China	5.5
9	DREAM 6	Siwei Liu and Zhenhua Huang	Dalian Jiaotong University, China	4
10	NTNU C6	Shih-Chieh Huang and Yun-Ching Liu	National Taiwan Normal University, Taiwan	3

Table 1: The participants and final standings.

The game Connect6, a kind of six-in-a-row game, was first introduced by Wu and Huang (2005) and then described in more detail by Wu, Huang, and Chang (2005). The rules of Connect6 are very simple. Two players, henceforth represented as B (designated as the first player) and W, alternately place two stones, black and white respectively, on one empty intersection of an 19×19 board, except for that B places one stone initially. The player who first obtains six consecutive stones (horizontally, vertically or diagonally) of his own wins the game. Unlike Renju, a professional variation of five-in-a-row, no extra prohibition rules are imposed on Connect6. When all intersections on the board are occupied without connecting six, the game draws.

In the tournament, the games were played according to a round-robin system in which one program played twice against all the other programs. In each game, every program had to complete all of its moves in 30 minutes. For each game, the winner scored 1 point and the loser scored nothing. However, for a draw game, both scored 0.5.

NCTU6-LITE, a light weight version of NCTU6 that won the gold of the 11th Computer Olympiad, won 17

[1] Dept. of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, Email: icwu@csie.nctu.edu.tw and bhlin@csie.nctu.edu.tw

points and the gold of the 13th Computer Olympiad. BITSTRONGER, the second in 2007 Chinese Computer Games Championship, won 17 points and the silver. NEUCONN6, the first in 2007 Chinese Computer Games Championship, won 13 points and the bronze. Both programs, NCTU6-LITE and BITSTRONGER, were close and won one game against each other. However, BITSTRONGER lost one more game to ML. Hence, NCTU6-LITE obtained one more point, winning the gold. The cross table is listed in Table 2. In the tournament, eight games in total drew. Note that the two games between NTNU C6 and CV6 are both drew.

Program	NCTU	BIT	NEUC	BCCC	KAV	NEUS	ML	CV6	DRM	NTNU
NCTU6-LITE	-	1	2	2	2	2	2	2	2	2
BITSTRONGER	1	-	2	2	2	2	1	2	2	2
NEUCONN6	0	0	-	2	1	2	2	2	2	2
BCCC	0	0	0	-	1	1.5	1.5	1.5	2	1.5
KAVALAN	0	0	1	1	-	0.5	2	1	1	1.5
NEU6STAR	0	0	0	0.5	1.5	-	1	2	1	2
ML	0	1	0	0.5	0	1	-	1	2	1
CV6	0	0	0	0.5	1	0	1	-	2	1
DREAM 6	0	0	0	0	1	1	0	0	-	2
NTNU C6	0	0	0	0.5	0.5	0	1	1	0	-

Table 2: The cross table.

This report comments the two games between NCTU6-LITE and BITSTRONGER, as well as the game that BITSTRONGER lost to ML. Since Connect6 is *draw-ish* due to balancing, strong Connect6 programs should play aggressive in the sense that they do not want to draw too many games to get high points in a tournament. Therefore, both NCTU6-LITE (the gold) and BITSTRONGER (the silver) played aggressively. However, on the other hand, playing aggressively also takes high risks of exposing weakness. Figure 1 shows the record of the first game, NCTU6-LITE (B) vs. BITSTRONGER (W). In this game, NCTU6-LITE made a blunder at 5 and BITSTRONGER immediately caught the blunder by playing at 6, verified as a winning move by NCTU6. Interestingly, the situation was reversed for the second game, BITSTRONGER (B) vs. NCTU6-LITE (W), shown in Figure 2. BITSTRONGER also made a blunder at 5 and NCTU6-LITE immediately caught the blunder by playing at 6, also verified as a winning move by NCTU6. Subsequently, NCTU6-LITE made no similar blunders and won all of the rest of games. In contrast, while playing aggressively, BITSTRONGER sometimes offered some chances for opponents' winning. Figure 3 shows the record of the game that BITSTRONGER lost to ML. In this game, ML defended well without blunders and grew stronger potential outside. For Move 46, W should have played at (E11, L15) to win. On the other hand, for Move 47, B should have played at (E11, G11) to defend. E11 was the key place for both players. Since 47, B had been losing. A similar situation happened in the game, BITSTRONGER played against BCCC. However, BCCC was not able to find some winning moves in that game and lost to BITSTRONGER finally.

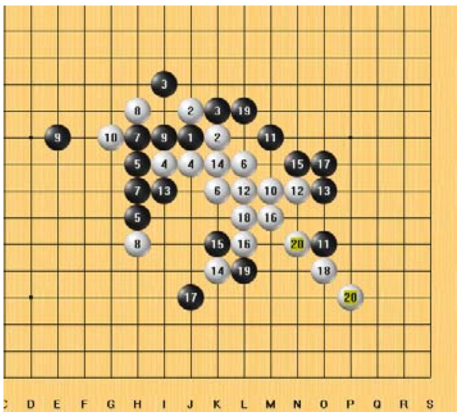


Figure 1: Black: NCTU6-LITE, White: BITSTRONGER, Moves 1 – 20.

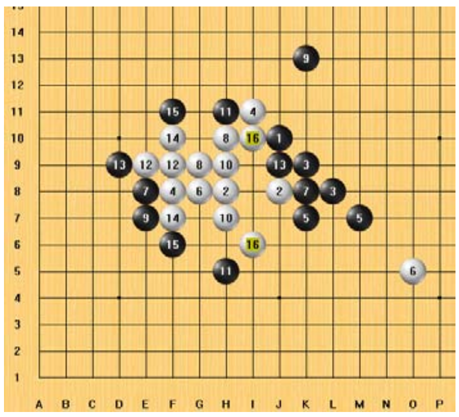


Figure 2: Black: BITSTRONGER, White: NCTU6-LITE, Moves 1 – 16.

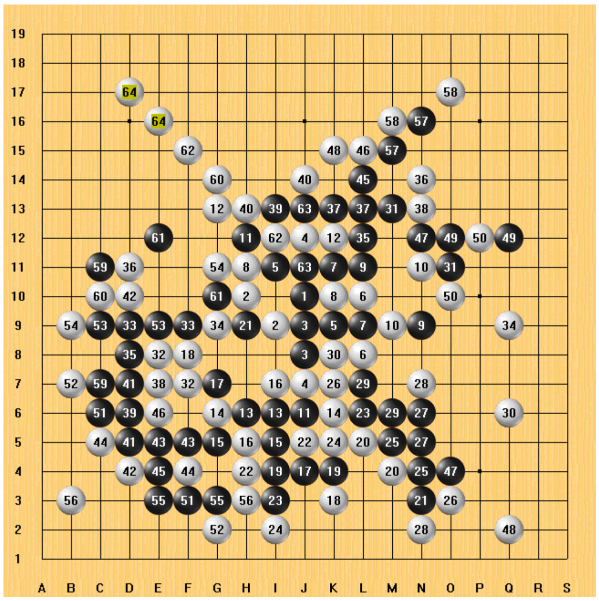


Figure 3: Black: BITSTRONGER, White: ML, Moves 1 – 64.

F.l.t.r. B. H. Lin, I-C. Wu, and H.J. van den Herik.

F.l.t.r. L. Lee (BITSTRONGER) and B. H. Lin (NCTU6-LITE).

References

Wu, I-C. and Huang, D.-Y. (2006) A New Family of k-in-a-row Games. *The 11th Advances in Computer Games* (ACG11) Conference, Taipei, Taiwan. (to be published)

Wu, I-C., Huang, D.-Y., and Chang., H.-C. (2006) Connect6, *ICGA Journal*, Vol. 28, No.4, pp. 234-242.

Wu, I-C., and Yen., S.-J. (2006) NCTU6 Wins Connect6 Tournament, *ICGA Journal*, Vol. 29, No.3, pp. 157-158.

NCTU6 Wins in the Man-Machine Connect6 Championship 2009

Ping-Hung Lin[1] *and I-Chen Wu*[1]

Taiwan

The Man-Machine Connect6 Championship 2009, sponsored by some organizations and industrial companies, was held in Hsinchu, Taiwan, on October 11, 2009. Four of top Connect6 players from Taiwan, listed in Table 1 below, attended this contest and played against NCTU6, the program developed by the team led by I-Chen Wu, including Ping-Hung Lin and Hung-Hsuan Lin.

Player	**Player Points**			**NCTU6 Points**		
	Round 1	**Round 2**	**Total**	**Round 1**	**Round 2**	**Total**
Wen-Ching Hsu	0	0	0	2	2	4
Cheng-Guo Chen	0	0	0	2	2	4
Wei-Han Chen	0	0	0	2	2	4
Shi-Wen Lee	0	0	0	2	2	4

Table 1: The participants and final standings.

The game Connect6, a kind of six-in-a-row game, was first introduced by Wu and Huang (Wu and Huang, 2005) and then described in more detail by Wu, Huang, and Chang (Wu, Huang, and Chang, 2005). The rules of Connect6 are very simple. Two players, henceforth represented as Black (designated as the first player) and White, alternately place two stones, black and white respectively, on one empty intersection of an 19×19 board, except for that Black places one stone initially. The player who first obtains six consecutive stones (horizontally, vertically or diagonally) wins the game. When all intersections on the board are occupied without connecting six, the game is drawn.

In this contest, four of top Connect6 players from Taiwan, Wen-Ching Hsu, Cheng-Guo Chen, Wei-Han Chen and Shi-Wen Lee, were invited to play against NCTU6. The first three are the top four in the Fourth Annual NCTU-Cup Connect6 Open Tournament (whose web pages are in www.connect6.org), which were held on August 23, in 2009. NCTU-Cup Connect6 Open Tournament is the most important annual Connect6 tournament held in Taiwan that usually attracts about a hundred players each year. The winners in this tournament are usually ones of the top players in Taiwan. Shi-Wen Lee is the head of Taiwan Connect6 Club, who could also be the first player who posted Connect6 openings and puzzles over the Internet (namely posted in November, 2005).

The Connect6 program, NCTU6, attended the 11th and 13th Computer Olympiad in both 2006 (Wu and Yen, 2006) and 2008 (Wu and Lin, 2008), and won gold both. NCTU6 also beat Chou Chun-Hsun (also transliterated as Zhou Junxun), an ever Go Champion, in invited competition events between NCTU6 and Chou, sponsored by National Science Council in Taiwan and some other organizations.

In this championship, the games were played in two rounds. NCTU6 played first against each human player in the first round, while playing second in the second round. In each game, every player freely played during the initial 80 minutes. After the period, each player had at most 10 times to play moves that took more than one minute, or lost the game. For each game, the winner scored 2 points and the loser scored nothing. For a draw game, both scored 1. NCTU6 won all games and the final points for human players are listed in Table 1. The winner was awarded NT$6000, roughly US$180.

This report comments four games between NCTU6 and human players. First, two of the games in the first round are commented. Figure 1 shows the record of the game, NCTU6 (Black) vs. Shi-Wen Lee (White). Moves 2 to 5 are a popular opening that is also played in the game shown in Figure 2. Lee made a good shape at Move 8. However, NCTU6 also successfully made a counter move at 9 that forced White back to

[1] Dept. of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, Email: bhlin@csie.nctu.edu.tw and icwu@csie.nctu.edu.tw.

defend. After Move 9, NCTU6 continuously played aggressively, but Lee also defended well and grew better outside. Finally, NCTU6 won the game when Lee made a blunder at Move 26. Figure 2 shows the record of the game, NCTU6 (Black) vs. Wei-Han Chen (White), in the same round. In this game, Moves 2 to 5 are the same opening. Chen made a very good shape at Move 14. Fortunately, NCTU6 played well at Moves 15, 17 and 19 to resolve the good shape. NCTU6 won the game after Chen made a blunder at Move 22 which should have defended at G9 and H10.

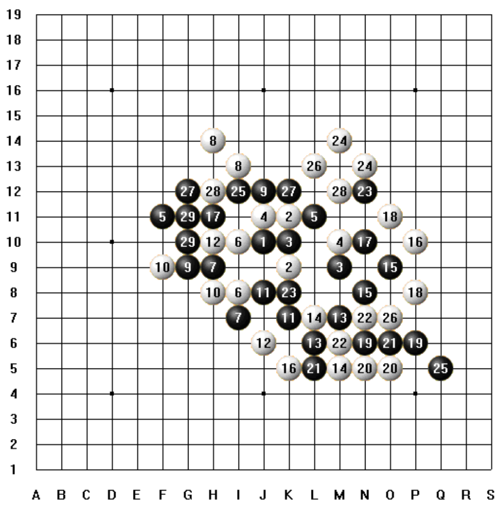


Figure 1: Black: NCTU6, White: Shi-Wen Lee, Moves 1 – 29.

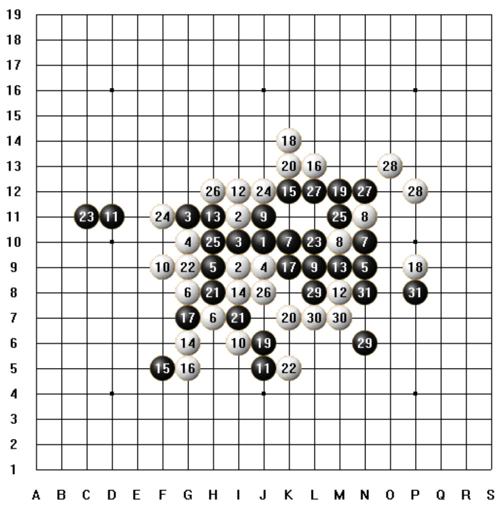


Figure 2: Black: NCTU6, White: Wei-Han Chen, Moves 1 – 31.

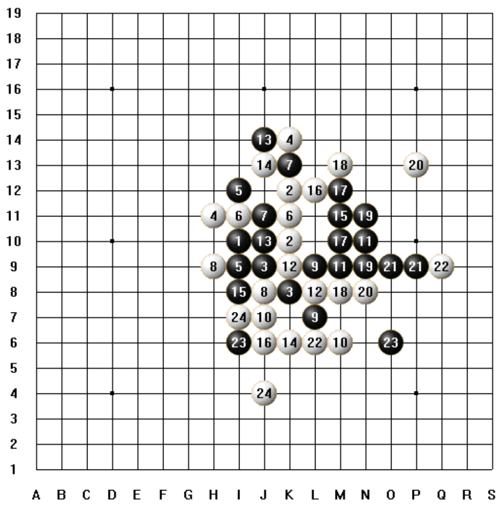


Figure 3: Black: Wen-Ching Hsu, White: NCTU6, Moves 1 – 24.

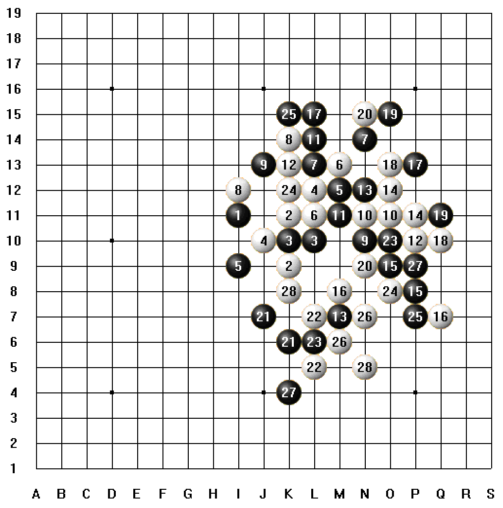


Figure 4: Black: Cheng-Guo Chen, White: NCTU6, Moves 1 – 28.

For the two games in the second round, we first comment the record of the game, Wen-Ching Hsu (Black) vs. NCTU6 (White), as shown in Figure 3. Although the opening played by White is not common and slightly better for Black, NCTU6 still chose this opening since this opening was more robust if opponents did not play starting at the center. From Move 3 to Move 19, Hsu played very aggressively and controlled this game. However, Hsu made a blunder at Move 21 that should have defended at L6, and NCTU6 thus won this game. In another game, Cheng-Guo Chen (Black) vs. NCTU6 (White) shown in Figure 4, NCTU6 made a bad shape at Move 18. NCTU6 still fortunately won when Chen neglected a winning move by White at Move 24.

In general, human players are good at making good shapes to win games. NCTU6 still needs to be improved to make better shapes. On the other hand, human players do not search winning moves as accurately as NCTU6 does, especially under time pressures. NCTU6 can easily catch any blunders made by human players. Unanimously, the four players thought that they might be able to win some games, if played with more times. After the games, a common agreement that was reached is to allow much more times in the championship next year.

F.l.t.r. Ping-Hung Lin, Shi-Wen Lee, Cheng-Guo Chen, I-Chen Wu, Wei-Han Chen, Shun-Ji Guo (the referee), and Wen-Ching Hsu.

References

Wu, I-C. and Huang, D.-Y. (2005) A New Family of k-in-a-row Games. *The 11th Advances in Computer Games* (ACG11) Conference, Taipei, Taiwan.

Wu, I-C., Huang, D.-Y., and Chang, H.-C. (2005) Connect6, *ICGA Journal*, Vol. 28, No.4, pp. 234-241.

Wu, I-C., and Yen, S.-J. (2006) NCTU6 Wins Connect6 Tournament, *ICGA Journal*, Vol. 29, No.3, pp. 157-158.

Wu, I-C., and Lin, P.-H. (2008) NCTU6-Lite Wins Connect6 Tournament, *ICGA Journal*, Vol. 31, No. 4, pp. 244-247.

JOB-LEVEL PROOF-NUMBER SEARCH FOR CONNECT6

I-Chen Wu[1], Hung-Hsuan Lin[1], Ping-Hung Lin[1], Der-Johng Sun[1], Yi-Chih Chan[1], and Bo-Ting Chen[1]

[1] Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
{icwu, bhlin, stanleylin, derjohng, nick314, qqting}@java.csie.nctu.edu.tw

Abstract. This paper proposes a new approach for proof number (PN) search, named job-level PN (JL-PN) search, where each search tree node is evaluated or expanded by a heavy-weight job, which takes normally over tens of seconds. Such JL-PN search is well suited for parallel processing, since these jobs are allowed to be performed by remote processors independently. This paper applies JL-PN search to solving automatically several Connect6 positions including openings on desktop grids. For some of these openings, none of human experts had been able to find the winning strategies before. Our experiments also showed the speedups for solving these positions are roughly linear, fluctuated from *sublinear* to *superlinear*. Hence, JL-PN search appears to be a very promising approach to solving games.

Keywords: Connect6, proof-number search, job-level proof-number search, threat-space search, desktop grids.

1 Introduction

Proof-number (PN) search, proposed by Allis et al. [1,3], is a kind of best-first search algorithm that was successfully used to prove or solve theoretical values [9] of game positions for many games [1,2,3,8,17,18,19,23], such as Connect-Four, Gomoku, Renju, Checkers, Lines of Action, Go, Shogi. Like most best-first search, PN search has a well-known disadvantage, the requirement of maintaining the whole search tree in memory. Therefore, many variations [5,11,14,15,19,23] were proposed to avoid this problem, such as PN^2, DF-PN, PN*, PDS, and parallel PN search [10,17] were also proposed. For example, PN^2 used two-level PN search to reduce the size of the maintained search tree.

This paper proposes a new approach, named *job-level proof-number* (*JL-PN*) *search*, where the PN search tree is maintained by a process, the *client* in this paper, and search tree nodes are evaluated or expanded by *heavy-weight jobs*, which can be executed remotely in a parallel system. Heavy-weight jobs take normally tens of seconds or more (perhaps up to one day).

In JL-PN search, we leverage the well-written programs as the heavy-weight jobs. In this paper, NCTU6 and NCTU6-Verifier (abbr. Verifier) are used as the heavy-weight jobs for Connect6. NCTU6 is a Connect6 program which won the gold of Connect6 Tournaments in Computer Olympiad [26,27,30] in 2006 and 2008 and also won 8 games and lost nothing against top Connect6 players [12] in Taiwan in 2009, and is used to generate a move (a node) and also evaluate the generated node. Verifier is a verifier modified from NCTU6, and is used to generate all the defensive moves (on the other hand, the moves not generated are proved to be losing). The JL-PN approach has the following advantages.

- Develop jobs (well-written programs) and the JL-PN search independently, except for a few efforts required to support JL-PN search from these jobs. As described in this paper, these required efforts are relatively low.
- Dispatch jobs to remote processors in parallel. Such JL-PN search is well suited for parallel processing, since these jobs are allowed to be performed by remote processors independently.
- Maintain the JL-PN search tree inside the client memory without much problem. Since well-written programs also support accurate domain-specific knowledge to a certain extent, the search trees require less nodes to solve the game positions (when compared with PN search). In our experiments for Connect6, the search tree usually contains no more than one million nodes, which can fit process (client) memory well. Assume that it takes one minute (60 seconds) to run NCTU6. Then, a parallel system with 60 processors takes about 11 days to build a tree up to one million nodes. In such cases, we can manually split one JL-PN search into two.
- Easily monitor the search tree. Since the maintenance cost for the search tree is relatively low when compared with the heavy-weight jobs, the client that maintains the JL-PN search tree can support more GUI utilities to let users easily monitor the running of the whole JL-PN search tree real time. For example, let users look into the search tree during the running time.

Using JL-PN search with the two jobs NCTU6 or Verifier on desktop grids, a kind of volunteer computing systems [4,7,20,25], this paper solved several Connect6 positions including several 3-move openings as shown in Figure 6 (below). For some of these openings, none of professionals had been able to find the winning strategies before. These solved openings include the popular one as shown in Figure 6 (i), named *Mickey-Mouse Opening* [21] (since White 2 and Black 1 together look like a face of the Mickey Mouse).

This paper is organized as follows. Section 2 reviews Connect6 applications including the jobs for Connect6. Section 3 describes JL-PN search and discusses some related issues. Section 4 does experiments for JL-PN search. Section 5 makes concluding remarks.

2 Connect6 Applications

Connect6 [28,29] is a kind of *six-in-a-row* game that was introduced by Wu et al. Two players, named Black and White in this paper, alternately place *two* black and white stones respectively on empty intersections of a Go board (a 19×19 board) in each turn. Black plays first and places *one* stone initially. The player who gets *six* consecutive stones of his own first horizontally, vertically or diagonally wins.

One issue for Connect6 is that the game lacks openings for players since the game is still young when compared with other games such as Chess, Chinese Chess and Go. Hence, it is important for Connect6 player community to investigate more openings quickly. For this issue, Wu et al. in [25] designed a *desktop grid* to help human experts build and solve openings. The desktop grid is also used as our parallel system for JL-PN search. Processors in the grid are called *workers*.

In the desktop grid, two programs, NCTU6 and Verifier, are embedded as jobs. NCTU6 is a Connect6 program, written by some of the authors, as also described in Section 1. According to [27], NCTU6 included a solver that was able to find *Victory by Continuous Four* (*VCF*), a common term for winning strategies in the Renju community. More specifically, VCF for Connect6, also called VCST in [27], is to win by making continuously moves with at least one four (that threat the opponent to defend) and ending with connecting up to six in all subsequent variations.

From the viewpoint of lambda search [22,27], VCF or VCST is a winning strategy in the second order of threats according to the definition in [27], that is, a Λ_a^2-tree (similar to a λ_a^2-tree in [22]) with value 1. Lambda search defined by Thomson [22] is a threat-based search method, formalized to express different orders of threats. Wu and Lin [27] modified the definition to fit Connect6 as well as a family of k-in-a-row games and changed the notation from λ_a^i to Λ_a^i.

Verifier is a verifier modified from NCTU6, and is used to verify whether the player to move loses in the position, or list all the defensive moves that may prevent from losing in the order Λ_a^2. If a move is not listed, Verifier is able to prove that the move is losing [27]. In some extreme cases, Verifier may report up to tens of thousands of moves. Generating such a large number of moves in PN search is resource-consuming (either computation or memory resources).

NCTU6 jobs usually take about one minute and NCTU6-Verifier jobs take a wide variety of times, from one minute up to one day, depending on the number of defensive moves. In the research [25], human experts solve positions by submitting jobs to free workers (in a desktop grid) manually. This paper is to use JL-PN search to submit jobs automatically. In order to support the automation, two additional functionalities are also added into NCTU6 as follows.

1. Given a list of exclusive moves as input, NCTU6 generates the best move among all the non-exclusive moves (those not in the list).
2. For the above functionality, if all the non-exclusive moves cannot prevent from losing, NCTU6 needs to report a sure loss message. After supporting this functionality, NCTU6 is able to replace Verifier in some cases. This functionality is critical in JL-PN search, described in Section 3.

3 Job-Level Proof Number (JL-PN) Search

This section presents job-level proof number (JL-PN) search that is used to solve Connect6 positions automatically. For simplicity of discussion about proof-number (PN) search, we follow in principle the definitions and algorithms in [1,3]. PN search is based on an AND/OR search tree where each node *n* is associated with proof/disproof numbers, *p*(*n*) and *d*(*n*), which represent the minimum numbers of nodes to be expanded to prove/disprove *n*. The values *p*(*n*)/*d*(*n*) are 0/∞ if the node *n* is proved, and ∞/0 if it is disproved. PN search repeatedly chooses a leaf called *the most-proving node* (*MPN*) to expand, until the root is proved or disproved. The details of choosing MPN and maintaining the proof/disproof numbers can be found in [1,3] and therefore is omitted in this paper. If the selected MPN is proved (disproved), the proof (disproof) number of the root of the tree is decreased by one.

Our JL-PN search is parallel PN search with the following two features. First, well-written programs such as NCTU6 and Verifier are used to expand and generate MPNs. These programs are viewed as jobs, sent to and done by free workers in a desktop grid. Second, multiple MPNs are allowed to be chosen simultaneously and therefore can be done by different workers in parallel.

In the rest of this section, Subsection 3.1 briefly describes the initializations of the proof/disproof numbers that help guide the search. Subsection 3.2 discusses the first feature, node expansion and generation, using NCTU6 and Verifier. Subsection 3.3 describes a very important algorithm of choosing the next MPN for parallelism for the second feature.

3.1 Proof/Disproof Number Initialization

This subsection briefly describes how to apply the domain knowledge given by NCTU6 to initialization of the proof/disproof numbers. Since it normally takes one minute or even more to execute a NCUT6 or Verifier job, it becomes critical to choose a *good* MPN carefully to expand, especially when there are many candidates with 1/1 as the standard initialization. In [1], Allis suggested several methods such as the use of the number of nodes to be expanded, the number of moves to the end of games, or the depth of a node.

Status	Bw	B4	B3	B2	B1	W1	W2	W3	W4	Ww	stable	unstable1	unstable2
p(n)/d(n)	0/∞	1/18	2/12	3/10	4/8	8/4	10/3	12/2	18/1	∞/0	6/6	5/5	4/4

Table 1: Game status and the corresponding initializations.

Our approach is simply to trust NCTU6 and use its evaluations on nodes (positions) to initialize the proof/disproof numbers in JL-PN search as shown in Table 1. The status Bw indicates that Black has a sure win, so the proof/disproof numbers of a node with Bw are 0/∞. The status B1 to B4 indicates that the game favors Black with different levels, where B1 indicates to favor Black least and B4 most (implicitly Black has a very good chance to win for B4) according to the evaluation by NCTU6. Similarly, the status W* are for White. The status stable indicates that the game is stable for both players, while both unstable1 and unstable2 indicate unstable, where unstable2 is more unstable than unstable1.

Surely, there are many different kinds of initializations other than those in Table 1. Our philosophy is simply to pass the domain-specific knowledge from NCTU6 to JL-PN search. Different programs or games surely have different policies on initializations from practical experiences.

3.2 Node Expansion and Generation

In JL-PN search, NCTU6 and Verifier are used to expand and generate nodes. Given a node (a position) *n* and a list of its children (exclusive moves), NCTU6 expands *n* by generating from *n* a new child (the best among all the moves outside the list) and evaluating the new child. Given a node *n*, Verifier expands *n* by generating all the children (that may prevent from losing in the order Λ_a^2). In our current version, Verifier does not evaluate these children (as described above), and provides no domain-specific knowledge (about how good these moves are).

In our earliest scheme of JL-PN search, we assumed in advance whom to win and then used NCTU6 to expand OR nodes and Verifier to expand AND nodes asymmetrically. Although it seems straightforward to prove positions in this scheme, this scheme has the following three drawbacks.

1. When expanding an AND node *n*, Verifier may generate a large number of moves as mentioned above. In the case that *n* is not proved but one sibling of *n* is proved, it may waste resources to generate all the children of *n*, especially when the number of children is very large. The dilemma is that it is hard to decide when Verifier should terminate node expansion.
2. Verifier provides no domain-specific knowledge so that these moves are not ordered for search. When Verifier generates a large number of moves from an AND node, this problem is even more serious. It is hard to choose which child to select for MPNs.
3. In many cases, it is hard to decide whom to win in advance. For example, White wins at 4 in Figure 6 (f). However, at the first glance, we and even some human experts thought that Black won or had an advantage at 3, and therefore spent time in proving whether Black wins, but unfortunately failed to prove at 4.

In order to cope with the above drawbacks, we developed several techniques and also successfully solved several positions and openings based on this scheme. However, since these techniques are too complicated and this scheme outperforms the next scheme in a few cases only, the above scheme is discussed no longer in this paper.

This paper uses the following scheme, instead: NCTU6 is used to expand all nodes. However, one issue raised from this scheme is when to generate siblings of nodes. Since chosen MPNs must be leaves, expanding chosen MPNs alone implies expanding leaves only. For this issue, we propose a method called *postponed sibling generation* as follows.

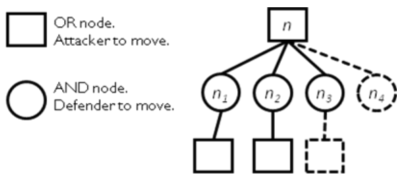


Figure 1: Expanding *n* (to generate n_4) and n_3 simultaneously.

- Assume that for a node *n* NCTU6 already generates the *i*-th move, n_i, but not yet for the (*i*+1)-st, n_{i+1}. When the node n_i is chosen as the MPN for expansion, expand n_i and generate n_{i+1} simultaneously. For generating n_{i+1}, NCTU6 expands *n* with an exclusive list of moves, n_1, n_2, …, n_i (using the first functionality as described in Section 2). For example, when the node n_3 is chosen as the MPN, expand n_3 and expand *n* (to generate n_4) simultaneously. On the other hand, if the branch n_1 or n_2 is chosen, do not generate n_4 yet. In addition, assume that the move to n_4 is a sure loss, reported by NCTU6. From the second functionality as described in Section 2, all the moves except for n_1, n_2 and n_3 lose. Then, the node *n* is no longer expanded. In this case, n_4 behaves as a stopper.

The postponed sibling generation method fits parallelism well, since both generating n_4 and expanding n_3 can be performed simultaneously. Some more issues are described as follows.

One may ask what if we choose to generate n_4 before expanding n_3. Assume that one player, say Attacker, is to move in the OR node *n*. Let Defender indicate the opponent. From the first additional functionality described in Section 2, the move n_3 is supposed to be *better* for Attacker than n_4 (according to the evaluation of NCTU6). Assume that it is indeed. Then, the condition $p(n_3) \le p(n_4)$ holds. Thus, the node n_3 must be chosen as the MPN to expand earlier than n_4. Thus, it becomes insignificant to generate n_4 before expanding n_3. In addition, the above condition also implies that the proof numbers of all the ancestors of node *n* remains unchanged. As for the disproof numbers of all the ancestors of *n*, these values are the same as or higher. Unfortunately, higher disproof numbers discourage the JL-PN search to choose n_3 as MPNs to expand. Thus, the behavior becomes awkward, especially if the node n_3 will be proved eventually.

One may also ask what if we expand n_3, but generate n_4 later. In such a case, it may make the proof number of *n* fluctuated. An extreme situation is that the value becomes infinity when all nodes, n_1, n_2 and n_3, are disproved.

3.3 Most Proving Nodes in Parallelism

This subsection discusses the key issue, choosing the MPNs to expand in parallel. When no MPNs are being expanded yet, we simply follow the traditional PN search to find an MPN and then use the method of postponed sibling generation (described in the previous subsection) to expand the MPN and generate its new sibling, if necessary. The node expansion and sibling generation form jobs which are respectively dispatched to free workers in the desktop grid. Whenever jobs are completed in workers, the results are returned back to the client. Then, the client updates the proof/disproof numbers of all nodes in the tree accordingly.

When some more free workers in the desktop grid are available, more MPNs are chosen for execution on these workers. However, if we do not change the proof/disproof numbers of the chosen MPNs being expanded, named the *active MPNs* in this paper, we would choose the same node obviously, as shown in Figure 2 (a) below. The issue is solved by the following policies.

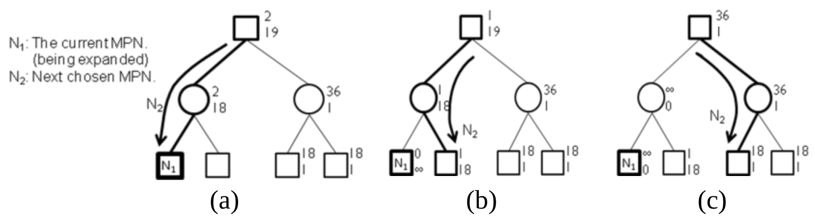


Figure 2: (a) Remaining unchanged. (b) Virtual win. (c) Virtual loss.

One policy of preventing from choosing the same node is to assume a virtual win [6] on the active MPNs. The idea of the virtual-win policy is to assume that the active are all proved. Thus, their proof/disproof numbers are all set to $0/\infty$, as illustrated in Figure 2 (b). When the proof number of the root is zero, we stop choosing more MPNs. The reason is that the root is already proved if the active are all proved.

Another policy is to assume a virtual loss on the active MPNs. Thus, the proof/disproof numbers of these nodes are set to $\infty/0$ as shown in Figure 2 (c). Similarly, when the disproof number of the root is zero, we stop choosing more MPNs. Similarly, the root is disproved, if all the active are disproved.

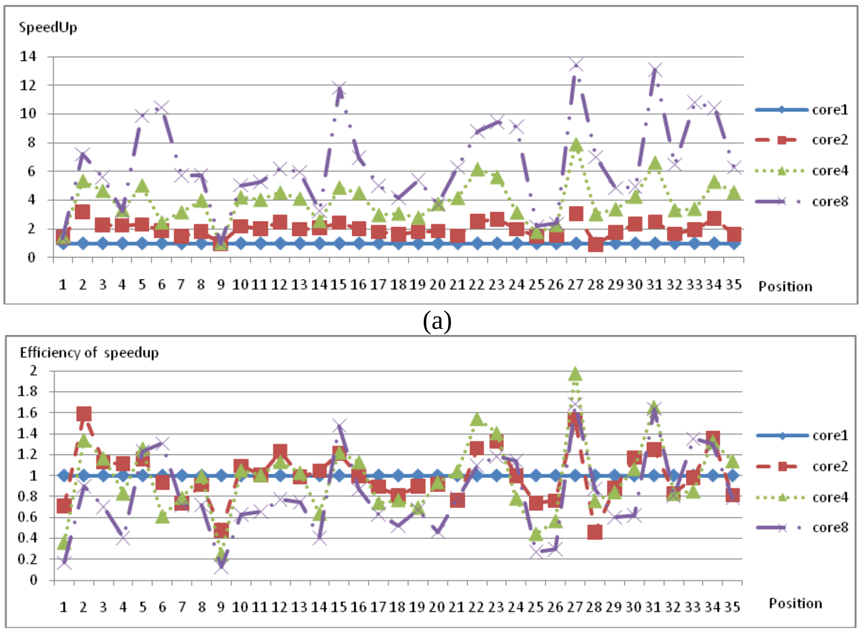
We also propose another policy, named a *greedy policy*, which chooses virtual-loss when the chosen nodes favor the disproof according to the evaluation of NCTU6, and chooses virtual-win otherwise. As described above, we may not be able to decide whom to win in advance in some cases such as the one in Figure 6 (f). This policy is used to see whether it makes differences.

4 Experiments

In our experiments of JL-PN search, the benchmark included 35 Connect6 positions (available in [31]), among which the last 15 positions are won by the player to move, while the first 20 are won by the other. The first 20 and the last 15 are ordered according to the computation time in the desktop grid [25] with 8 workers, actually 8 cores on four Intel Core2 Duo 3.33 GHz machines. In our experiments, the client is located on another host. Note that the time for maintaining the JL-PN search tree is negligible, since it is relatively low when compared with those of NCTU6 and Verifier (normally taking 1 minute or more).

Figure 3 (below) shows the speedups and the speedup efficiencies of the 35 positions using JL-PN search with the virtual-loss policy and with 1, 2, 4 and 8 cores respectively. Let the speedup $S_k = T_1/T_k$, where T_k is the computation time for solving a position with the virtual-loss policy with k cores. Also, let the speedup efficiency $E_k = S_k/k$. The efficiencies are one for ideal linear speedups.

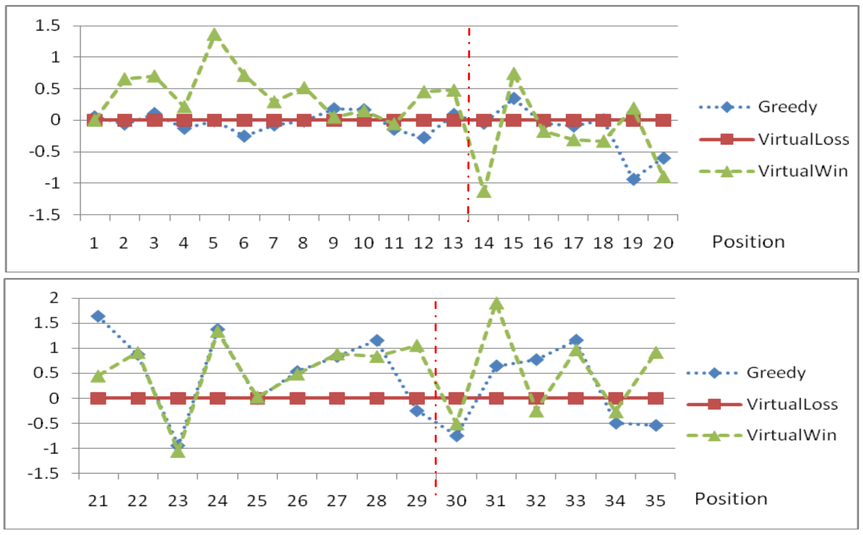
From Figure 3, the speedups for our JL-PN search are roughly linear, but are fluctuated from *sublinear* to *superlinear* due to the high uncertainty of parallel state-space search [13,16]. The phenomenon of superlinear speedups for parallel state-space search has been discussed in [16] in greater detail. Since PN search is a kind of state-space search, it fits the phenomenon. Although fluctuated, the speedups are close to linear speedups. Such a result shows that JL-PN search is a very promising approach to solving games.



(a)

(b)

Figure 3: (a) Speedups S_k and (b) efficiencies E_k of the 35 positions for k = 1, 2, 4 or 8.



(a)

(b)

Figure 4: Normalized logarithmic time scales with 8 cores (a) for the first 20 positions and (b) for last 15.

Our next experiment is to investigate the three policies, virtual-win, virtual-loss and greedy policies, as proposed in Subsection 3.3. For these policies, we measured their computation times with 8 cores only, normalized to those for the virtual-loss policy in the following way. Let $T_{i,vwin}$, $T_{i,vloss}$ and $T_{i,grd}$ be the times for solving the *i*th position in the benchmark with virtual-win, virtual-loss and greedy policy. The normalized logarithmic time scales are $N_{i,type} = \log(T_{i,type}/T_{i,vloss})$, where *type* is *vwin*, *vloss* or *grd*. Clearly, the scales for the virtual-loss policy are all zeros. Figure 4 shows all $N_{i,vwin}$, $N_{i,vloss}$ and $N_{i,grd}$. The higher the scales are, the less efficient the performances are with respect to the virtual-loss policy.

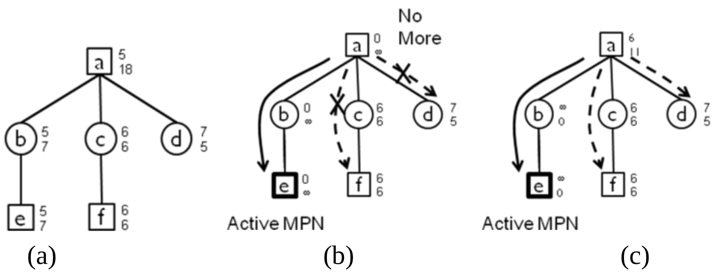


Figure 5: (a) A JL-PN search tree. (b) The virtual-win policy. (c) The virtual-loss policy.

From Figure 4, the normalized logarithmic time scales are fluctuated due to the same reason, the high uncertainty of parallelizing a search tree. In general, none of the policies has clear advantages over any others, except that we observe the following phenomenon: The virtual-loss policy seems slightly better in the positions where the tree size is relatively smaller. These positions are in the left hand side of dashed lines in Figure 4 (note that the positions are ordered according to the computation times as mentioned above).

The phenomenon is explained and illustrated in the following example. Consider a JL-PN search tree in Figure 5 (a). After choosing e as MPN, the virtual-win policy sets the proof/disproof numbers of both b and a (the root) to 0/∞ as shown in Figure 5 (b), and therefore chooses no more MPNs until the job for e is completed. In contrast, the virtual-loss policy sets the proof/disproof numbers of b to ∞/0, sets those of a to 6/11, and therefore chooses f and d as next MPNs, as shown in Figure 5 (c).

From the above observation, the virtual-win policy tends to choose MPNs from the first (or the first several) branch of the root, while the virtual-loss policy tends to choose MPNs from all branches. Thus, the virtual-loss policy tends to spread computations better and utilize parallelism better at the early stage. However, in the case that the tree size is large, the above advantage of the virtual-loss policy becomes less significant.

As for the greedy policy, it is in-between. If it follows the virtual-win policy at the very beginning, then the above phenomenon is also observed (see the left hand side of Figure 4 (b)). Otherwise, it is similar to the virtual-loss policy.

All in all, since the speedups are fluctuated seriously, it is hard to conclude which policy is the best, especially when the search tree is large. In our real experiences, we tend to use the virtual-loss policy due to the above phenomenon.

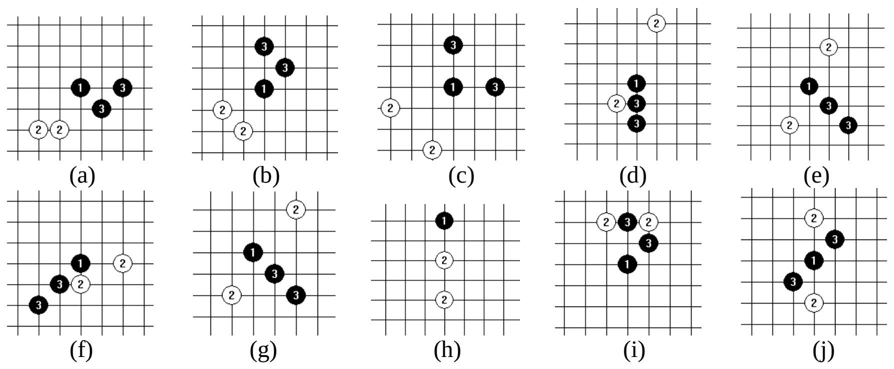


Figure 6: Ten openings in our benchmark.

Now, re-investigate the 35 positions in the benchmark. Among them, ten are 3-move openings shown in Figure 6. The winning strategies for the first three were also found in [27]. In these openings, White wins in the sixth one, while Black wins in others. For many of them, their winning strategies were not found before. Especially, the Mickey-Mouse Opening (the ninth one) had been one of popular openings before we solved it. The tenth one, also called Straight Opening, is another difficult one.

5 Conclusion

The contributions of this paper are summarized as follows.

- This paper proposes a new approach, JL-PN (job-level proof-number) search, to help solve the openings of Connect6. In this approach, some techniques are used, such as the method of postponed sibling generation and the policies of choosing MPNs. In this paper, JL-PN search was successfully used to solve several positions of Connect6 automatically, including several 3-move openings, such as Mickey-Mouse Opening and Straight Opening, which none of Connect6 human experts had been able to solve before.
- Our experiments also demonstrated roughly linear speedup, even superlinear speedups in some cases. Based on JL-PN search, we expect to solve and develop more Connect6 openings.
- From our experiments, we observed that the virtual-loss policy seemed slightly better for small tree sizes. However, for large tree sizes, we observed that none of the policies had clear advantages over any others.

In addition, the approach of JL-PN search has several advantages as indicated in Section 1. We expect to apply it to many other games in the near future.

Acknowledgments.

The authors would like to thank anonymous reviewers for their valuable comments, and thank the National Science Council of the Republic of China (Taiwan) for financial support of this research under contract numbers NSC 95-2221-E-009-122-MY2 and NSC 97-2221-E-009-126-MY3.

References

[1] Allis, L.V., *Searching for solutions in games and artificial intelligence*, Ph.D. Thesis, University of Limburg, Maastricht, The Netherlands, 1994.

[2] Allis, L.V., Herik, H. J. van den, and Huntjens, M. P. H., Go-Moku Solved by New Search Techniques. *Computational Intelligence*, Vol. 12, pp. 7-23, 1996.

[3] Allis, L.V., Meulen, M. van der, and Herik, H. J. van den, Proof-number search, *Artificial Intelligence*, Vol. 66(1), pp. 91-124, 1994.

[4] Anderson, D. P. Boinc: A system for public-resource computing and storage. In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, IEEE CS Press, Pittsburgh, USA, pp. 4-10, 2004.

[5] Breuker, D. M., Uiterwijk, J., and Herik, H. J. van den, The PN2-search algorithm, in H. J. van den Herik, B. Monien (Eds.), *Advances in Computer Games*, Vol. 9, IKAT, Universiteit Maastricht, Maastricht, The Netherlands, pp. 115-132, 2001.

[6] Chaslot, G. M., Winands, M. H. M., and Herik, H. J. van den, Parallel Monte-Carlo Tree Search. *The 6th International Conference on Computers and Games (CG2008)*, Beijing, China, 2008.

[7] Fedak, G., Germain, C., Neri, V., and Cappello, F., Xtremweb: A generic global computing system. In *Proceedings of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2001): Workshop on Global Computing on Personal Devices*, IEEE CS Press, Brisbane, Australia, pp. 582-587, 2001.

[8] Herik, H. J. van den, and Winands, M. H. M., Proof-Number Search and its Variants. In *Oppositional Concepts in Computational Intelligence*, pp. 91-118, 2008.

[9] Herik, H. J. van den, Uiterwijk, J. W. H. M., and Rijswijck, J. V., Games solved: Now and in the future. *Artificial Intelligence*, Vol. 134, pp. 277-311, 2002.

[10] Kishimoto, A., and Kotani, Y., Parallel AND/OR tree search based on proof and disproof numbers. In *5th Games Programming Workshop*, Vol. 99(14) of IPSJ Symposium Series, pp. 24-30, 1999.

[11] Kishimoto, A., and Müller, M., Search versus Knowledge for Solving Life and Death Problems in Go, Twentieth National Conference on Artificial Intelligence (AAAI-05), pp. 1374-1379, 2005.

[12] Lin, P.-H., and Wu, I.-C., NCTU6 Wins Man-Machine Connect6 Championship 2009, *ICGA Journal*, Vol. 32(4), pp. 230–232, 2009.

[13] Manohararajah, V. *Parallel alpha-beta search on shared memory multiprocessors*. Master's thesis, Graduate Department of Electrical and Computer Engineering, University of Toronto, Canada, 2001.

[14] Nagai, A., *Df-pn Algorithm for Searching AND/OR Trees and Its Applications*. PhD thesis, University of Tokyo, Japan, 2002.

[15] Pawlewicz, J., and Lew, L., Improving depth-first pn-search: $1+\varepsilon$ trick. In H. J. van den Herik, P. Ciancarini, and H.H.L.M. Donkers, editors, *5th International Conference on Computers and Games*, Vol. 4630 of LNCS, pp. 160-170. Computers and Games, Springer, Heidelberg, 2006.

[16] Rao, V. N., and Kumar, V., Superlinear Speedup in State-space Search. In *Proceedings of the 1988 Foundation of Software Technology and Theoretical Computer Science*, number 338 of LNCS, pp. 161-174, Springer-Verlag, 1988.

[17] Saito, J. T., Winands, M. H. M., and Herik, H. J. van den, Randomized Parallel Proof-Number Search. *Advances in Computer Games Conference (ACG'12), Lecture Notes in Computer Science* (*LNCS 6048*), pp. 75-87, Palacio del Condestable, Pamplona, Spain, 2009.

[18] Schaeffer, J., Burch, N., Björnsson, Y., N., Kishimoto, A., Müller, M., Lake, R., Lu, P., and Sutphen, S., Checkers is solved. *Science*, Vol. 5844(317), pp. 1518-1552, 2007.

[19] Seo, M., Iida, H., and Uiterwijk, J., The PN*-search algorithm: Application to Tsumeshogi. *Artificial Intelligence*, Vol. 129(1-2), pp. 253-277, 2001.

[20] SETI@home Project. available at *http://setiathome.ssl.berkeley.edu*.

[21] Taiwan Connect6 Association, Connect6 homepage, available at *http://www.connect6.org/*.

[22] Thomsen, T., Lambda-search in game trees - with application to Go. *ICGA Journal*, Vol. 23(4), pp. 203-217, 2000.

[23] Winands, M. H. M., Uiterwijk, J. W. H. M., and Herik, H. J. van den, PDS-PN: A new proof-number search algorithm: Application to Lines of Action. In J. Schaeffer, M. Müller, and Y. Björnson, editors, *Computers and Games 2002*, Vol. 2883 of LNCS, pp. 170-185. Computers and Games, Springer, Heidelberg, 2003.

[24] Wu, I-C., Hsu, S.-C., Yen, S.-J., Lin, S.-S., Kao, K.-Y., Chen, J.-C., Huang, K.-C., Chang, H.-Y., and Chung, Y.-C., *A Volunteer Computing System for Computer Games and its Applications*, an integrated project proposal submitted to National Science Council, Taiwan, 2010.

[25] Wu, I.-C., Chen, C.-P., Lin, P.-H., Huang, K.-C., Chen, L.-P., Sun, D.-J., Chan, Y.-C., and Tsou, H.-Y., "A Volunteer-Computing-Based Grid Environment for Connect6 Applications", *the 12th IEEE International Conference on Computational Science and Engineering* (*CSE-09*), August 29-31, Vancouver, Canada, 2009.

[26] Wu, I.-C., and Lin, P.-H., NCTU6-Lite Wins Connect6 Tournament, *ICGA Journal*, Vol. 31(4), pp. 240–243, 2008.

[27] Wu, I.-C., and Lin, P.-H., Relevance-Zone-Oriented Proof Search for Connect6, to appear in *the IEEE Transactions on Computational Intelligence and AI in Games*, 2010.

[28] Wu, I.-C., Huang, D.-Y., and Chang, H.-C., Connect6. *ICGA Journal*, Vol. 28(4), pp. 234-242, 2006.

[29] Wu, I.-C., and Huang, D.-Y., A New Family of k-in-a-row Games. *The 11th Advances in Computer Games Conference (ACG'11)*, pp. 180-194, Taipei, Taiwan, 2005.

[30] Wu, I.-C., and Yen, S.-J., NCTU6 Wins Connect6 Tournament, *ICGA Journal,* Vol. 29(3), pp. 157-158, September 2006.

[31] Wu, I.-C., et al. Benchmark for Connect6, available at http://www.connect6.org/articles/JL-PNS/.

Relevance-Zone-Oriented Proof Search for *Connect6*

I-Chen Wu, *Member, IEEE*, and Ping-Hung Lin

***Abstract*—Wu and Huang (*Advances in Computer Games*, pp. 180–194, 2006) presented a new family of k-in-a-row games, among which *Connect6* (a kind of six-in-a-row) attracted much attention. For *Connect6* as well as the family of k-in-a-row games, this paper proposes a new threat-based proof search method, named relevance-zone-oriented proof (RZOP) search, developed from the lambda search proposed by Thomsen (*Int. Comput. Games Assoc. J.*, vol. 23, no. 4, pp. 203–217, 2000). The proposed RZOP search is a novel, general, and elegant method of constructing and promoting relevance zones. Using this method together with a proof number search, this paper solved effectively and successfully many new *Connect6* game positions, including several *Connect6* openings, especially the Mickey Mouse opening, which used to be one of the popular openings before we solved it.**

***Index Terms*—Board games, *Connect6*, k-in-a-row games, lambda search, threat-based proof search, threat-space search.**

I. Introduction

A generalized family of k-in-a-row games, named *Connect*(m, n, k, p, q) [30], [31], was introduced and presented by Wu *et al.* Two players, named *Black* and *White*, alternately place p stones on empty *squares*[1] of an $m \times n$ board in each turn. Black plays first and places q stones initially. The player who first gets k consecutive stones of his own horizontally, vertically, and diagonally wins. Both players tie the game when the board is filled up with neither player winning. Games in this family are also called *Connect* games[2] in this paper. For example, *Tic-tac-toe* is *Connect(3,3,3,1,1)*, *Go-Moku* in the free style (a traditional five-in-a-row game) is *Connect(15,15,5,1,1)*, and *Connect6* played on the traditional *Go* board is *Connect(19,19,6,2,1)*. For simplicity, let *Connect*(k, p, q) denote the game *Connect*$(\infty, \infty, k, p, q)$, played on infinite boards. For example, when played on infinite boards, *Go-Moku* becomes *Connect(5,1,1)* and *Connect6* becomes *Connect(6,2,1)*.

Among these *Connect* games, *Connect6* attracted much attention due to three merits: fairness, simplicity of rules, and high game complexity as described in [30] and [31]. Since *Connect6* was introduced, hundreds of thousands of *Connect6* games have been played on web sites, such as littlegolem.net [14] and cygame.com [21]. Since 2006, several *Connect6* open tournaments [20] for human players have been held, such as NCTU Open, ThinkNewIdea Open, Russian Open, and World Open. *Connect6* has also been included as one of the computer game tournaments at the Computer Olympiad [24] and Chinese Computer Games Contest [9], since 2006 and 2007, respectively.

For *Connect6*, researchers in [30] and [31] mentioned a simple threat-based proof search method for solving *Connect(6,2,3)*. Section II shows that many more winning positions cannot be solved by such a method. This paper proposes a new threat-based proof search method, named relevance-zone-oriented proof (RZOP) *search*, developed from the lambda search proposed by Thomsen [22]. Section IV presents this novel, general, and elegant method of constructing and promoting relevance zones for *Connect6*. The proposed method is also generalized to all *Connect* games in the Appendix. Together with a proof number search [3], [28], it solved effectively and successfully many new *Connect6* game positions, including several *Connect6* openings, especially the Mickey Mouse opening, as described in Section V. This opening used to be one of the popular openings before we solved it. All definitions and notations used in this paper are given in Section III. Concluding remarks are made in Section VI.

II. Motivation

When *Connect6* was first introduced by Wu *et al.* [30], [31], they mentioned that threats are the key to winning *Connect6* as well as other *Connect* games, like *Renju*. According to the definitions by [30] and [31], one player has t and only t threats, if and only if t is the smallest number of stones that the opponent needs to place to prevent from losing the game in the next move. A move is called a single-threat move if the player who makes the move has one and only one threat after the move, a double-threat move if two, a triple-threat move if three, and a nonthreat move if none. In *Connect6*, one player clearly wins by a triple-threat-or-more move (a move with at least three threats).

In [30] and [31], Wu *et al.* showed a type of winning strategy, called victory by continuous double-threat-or-more moves (VCDT) in this paper. It is similar to victory by continuous four (VCF), a common term for winning strategies in the *Renju* community [15]. More specifically, the type of VCDT strategy is to win by making continuously double-threat moves and ending with a triple-or-more-threat move or connecting up to six in all variations, for example, in Fig. 1, White's VCDT 12–18 (18 is a triple-threat move) moves.

Soon after the introduction of *Connect6*, many experts found another type of winning strategy in which additional single-threat moves are involved, i.e., single-threat and double-threat

Manuscript received February 11, 2010; revised June 11, 2010; accepted July 12, 2010. Date of publication July 23, 2010; date of current version September 15, 2010. This work was supported in part by the National Science Council of the Republic of China (Taiwan) under contract numbers NSC 95-2221-E-009-122-MY2 and NSC 97-2221-E-009-126-MY3.

The authors are with the Department of Computer Science, National Chiao Tung University, Hsinchu 30050, Taiwan (e-mail: icwu@csie.nctu.edu.tw; bhlin@csie.nctu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2010.2060262

[1]Practically, stones are placed on empty intersections of *Renju* or *Go* boards. In this paper, by squares, we mean intersections.

[2]The term of connect games defined in [10] covers the games such as *Hex*, *Connect Four*, etc. In this paper, *Connect* are capitalized to indicate all the games in the family of *Connect*(m, n, k, p, q).

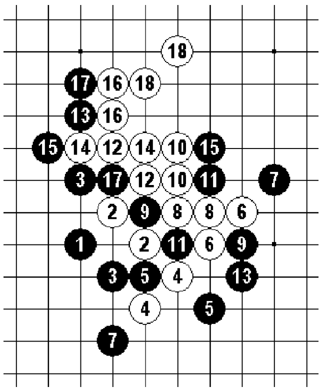


Fig. 1. Sequence of winning moves by White.

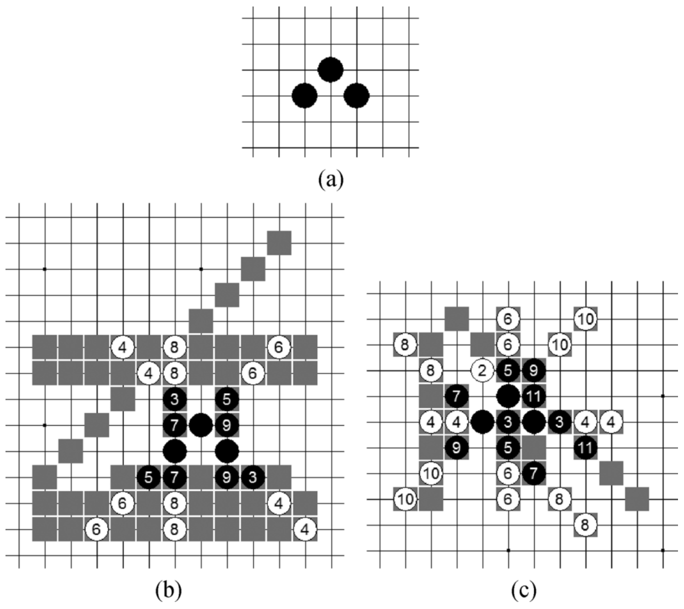


Fig. 2. (a) Black's winning move in *Connect(6,2,3)*. (b) VCDT for a null move in (a). (c) VCDT for a seminull move 2.

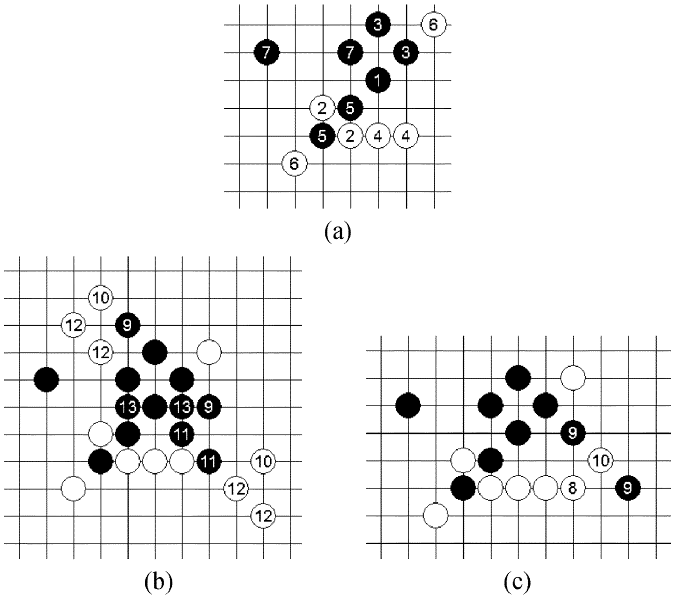


Fig. 3. (a) Position with Black winning. (b) VCDT for the null move in (a). (c) Winning single-threat move 9 for the seminull move 8.

moves are mixed (before ending with a triple-or-more-threat move). This type of winning strategy is herein called victory by continuous single-threat-or-more moves (VCST). For example, Lee [13], a *Renju* 3-dan player, found and claimed in late 2005 that White won starting from move 8 (both 8 and 10 are single-threat moves) in the game as shown in Fig. 1. Similarly, the type of winning strategy with additional non-threat moves involved is called victory by continuous nonthreat-or-more moves (VCNT).

Although VCST was unknown then, Wu *et al.* [30], [31] were already able to solve a simple VCNT case, when Black wins *Connect(6,2,3)*. This clearly is a case of VCNT, since Black's first winning move, as shown in Fig. 2(a), must be a nonthreat move. To solve it, they used a simple threat proof search method involving null or seminull moves and relevance zones, as briefly described in the following. Let White place no stones, called a null move in [30] and [31]. Obviously, Black wins by VCDT 3–9 as shown in Fig. 2(b). Then, a relevance zone Z, the area of gray squares in Fig. 2(b), can be derived to indicate that White must place at least one of the two stones inside this zone, or Black wins by simply replaying the same VCDT. Next, all squares s in Z are verified as follows. Let White place one stone on s only, called a seminull move in [30] and [31]; for example, move 2 in Fig. 2(c). Again, Black is able to win by another VCDT 3–11. Thus, another relevance zone Z', the gray area in Fig. 2(c), can be derived again to indicate that White must place another stone inside Z', or Black wins by replaying the same VCDT. Finally, all s are verified such that Black wins over all moves placed at s and s', where s' is in the Z' corresponding to the seminull move at s. Hence, Black was proved to win.

In the above search method for solving the case *Connect(6,2,3)* with VCNT, both winning strategies for the null move [3–9 in Fig. 2(b)] and the seminull move [3–11 in Fig. 2(c)] must be VCDT. However, with more and more winning *Connect6* positions investigated, we found that winning strategies for null and seminull moves may be VCSTs or even VCNTs, thus making these positions much more difficult to solve.

For example, consider the two winning nonthreat moves (proved in this paper): moves 7 in Fig. 3(a) and 6 in Fig. 4(a), respectively. The former, found in 2006 [20], was the key used to help prove that Black wins at move 3 in Fig. 3 [see also the opening in Fig. 22(a)]; that is, the opening move 2 is solved. In this case, for the null move in Fig. 3(a), Black wins by a VCDT as shown in Fig. 3(b). However, for the seminull move 8 in Fig. 3(c), Black has no double-threat moves to win by a VCDT, though Black wins by a VCST starting at 9 in Fig. 3(c).

The latter, the position in Fig. 4(a) found by Huang [11], was investigated to see whether the seminull move 5 was safe enough, since the position at 5 was popular in the following sense. Among all the first-five-move positions of *Connect6* games played by the players ranked above 1800 in [14], about 2% covered (or superset) the position according to the statistics discussed in [20]. The proof for this position is extremely complicated. Even for a null move by Black, White has no

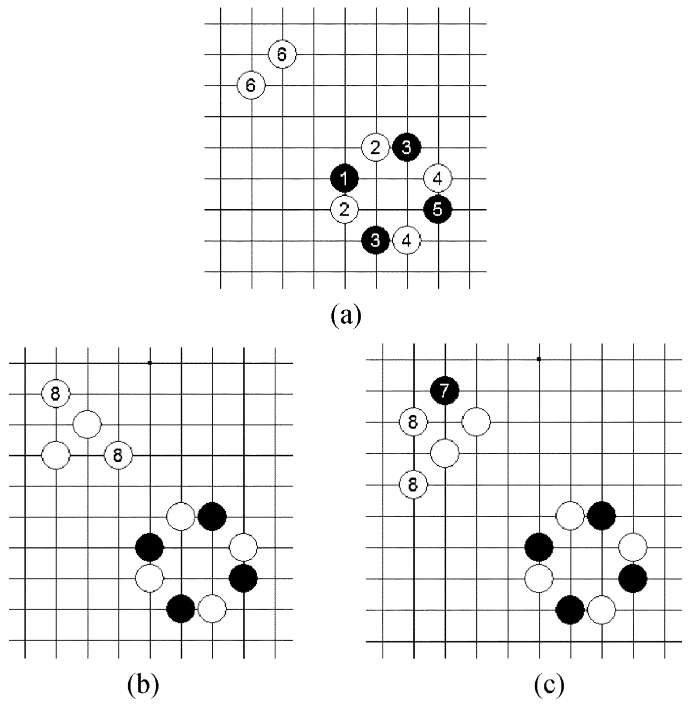


Fig. 4. (a) Position with White winning. (b) Winning single-threat move 8 for a null move in (a). (c) Winning nonthreat move 8 for a seminull move 7.

double-threat moves to win by a VCDT, but can actually win by a VCST starting at 8 as shown in Fig. 4(b). In addition, if a seminull move is made at 7 in Fig. 4(c), White cannot win by a VCDT or even a VCST, thus making the position in Fig. 4(a) much more complicated to solve.

In order to solve these as well as other positions shown in Section V, this paper proposes a new threat-based proof search method, named relevance-zone-oriented proof (RZOP) search, developed from the lambda search proposed by Thomsen [22]. In the past, many researchers [1]–[3], [6], [7], [22] have proposed threat-based search methods. Lambda search is to formalize the search trees with null moves and to solve positions of games such as *Go* and *Chess*. In lambda search, null moves are involved with different orders of threat sequences, also called lambda trees.

From the viewpoint of lambda search, a VCDT is a typical λ^1-tree with value 1 (cf., [22]). However, the definition of lambda search cannot be directly applied to *Connect6* or *Connect* games with $p \geq 2$. For *Connect* games, this paper modifies the definition of lambda search in Section III-D, and replaces the notation λ^i by Λ^i. Under the new definition, a VCST is a Λ^2-tree with value 1, the winning strategy for the position in Fig. 3(a) is a Λ^3-tree with value 1, while that in Fig. 4(a) is a Λ^4-tree with value 1. The Λ search formalized in this paper is able to solve Λ^1-trees to Λ^4-trees with value 1 for *Connect6*.

III. Definitions and Notation

This section gives definitions and notation related to *Connect* game positions, search trees, threats, lambda search, and relevance zones in Sections III-A–III-E, respectively.

A. Game Positions

In *Connect* games, a game position P includes the information of all the stones and their occupied squares on the board and the turn of whom to play. The player to be proved to win, either Black or White, is called the attacker and the other defender in this paper. Let $\sigma_A(s)$ denote the information of an attacker stone placed on the unoccupied square s, and $P + \sigma_A(s)$ denote the position after placing an attacker stone on s in position P without changing the turn. $\sigma_D(s)$ and $P + \sigma_D(s)$ are similarly defined for the defender. From the strategy stealing argument by Nash (cf., [4] and [30]), we obtain the following. If the attacker wins in P, he wins in $P + \sigma_A(s)$ as well; and if the attacker wins in $P + \sigma_D(s)$, he wins in P as well.

In this paper, $P \oplus M$ denotes the position after one player makes move M and before the other makes the next move. In *Connect6*, let $M_A(s_1, s_2)$ denote an attacker move where two attacker stones are placed on both unoccupied squares s_1 and s_2. $M_D(s_1, s_2)$ and $P \oplus M_D(s_1, s_2)$ are similarly defined for the defender. Note that in contrast to $P + \sigma_A(s_1) + \sigma_A(s_2)$, the position $P \oplus M_A(s_1, s_2)$ indicates changing the turn from the attacker to the defender.

In *Connect6*, one player, say an attacker, is allowed to make a null move, $M_{A,\phi\phi}$, that is, to place no stones; and a seminull move, $M_{A,\phi}(s_1)$, that is, to place one stone only on square s_1 in P. Thus, the position $P \oplus M_A(s_1, s_2)$ is equivalent to $(P \oplus M_{A,\phi}(s_1)) + \sigma_A(s_2)$ and $(P \oplus M_{A,\phi\phi}) + \sigma_A(s_1) + \sigma_A(s_2)$. From another viewpoint, null or seminull moves are to place some null stones while placing normal stones. In *Connect*(m, n, k, p, q), we place p null stones for a null move, while placing one to $p - 1$ null stones for seminull moves.

In *Connect6*, a segment is defined to be a set of six consecutive squares horizontally, vertically, or diagonally on the board, while in *Connect*(m, n, k, p, q), a segment is a set of k consecutive squares. A segment is called an empty segment if all the squares on it are unoccupied yet. A segment is called an active segment of one player, if none of the squares are occupied by the opponent's stones. An active segment of one player is called a win segment of the player, if all the squares on it are occupied by the player. Obviously, one player wins if the player makes a win segment. From the definition of *Connect* games, a game ends when one makes some win segment or all the squares of the board are already occupied. According to this definition, it is impossible for both players to have win segments simultaneously.

B. Search Trees

This paper basically follows the definitions of search trees in [5] and [17]. A search tree is shown in Fig. 5(a), where rectangle and circle nodes indicate the positions in the attacker's and defender's turns,[3] respectively. The value of a leaf is 1, if the attacker makes a win segment, and 0, otherwise. The value of a search tree is the minimax value of the tree. The attacker wins in the root position if the search tree has value 1 and all the internal circles expand all defender's legal moves.

[3]When we say that a position P is in the attacker's (defender's) turn, we mean that the attacker (defender) is to move next in P.

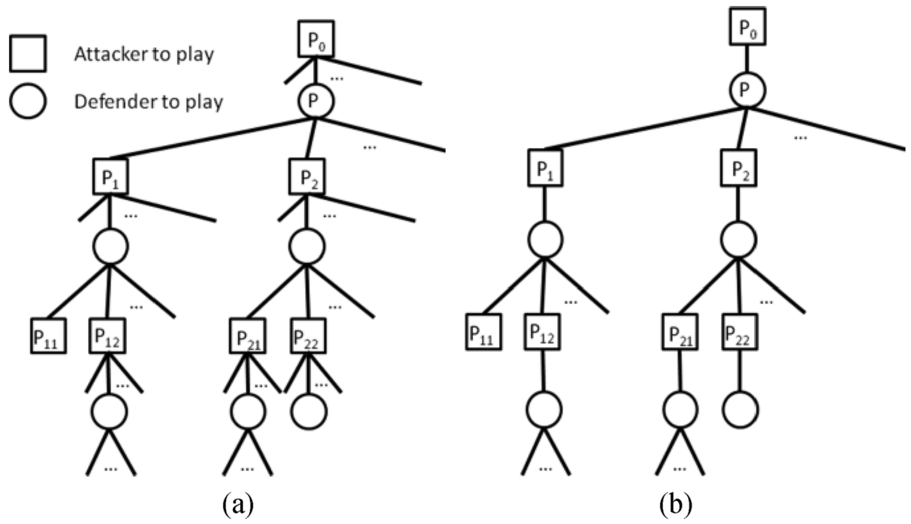


Fig. 5. (a) Search tree. (b) Solution tree.

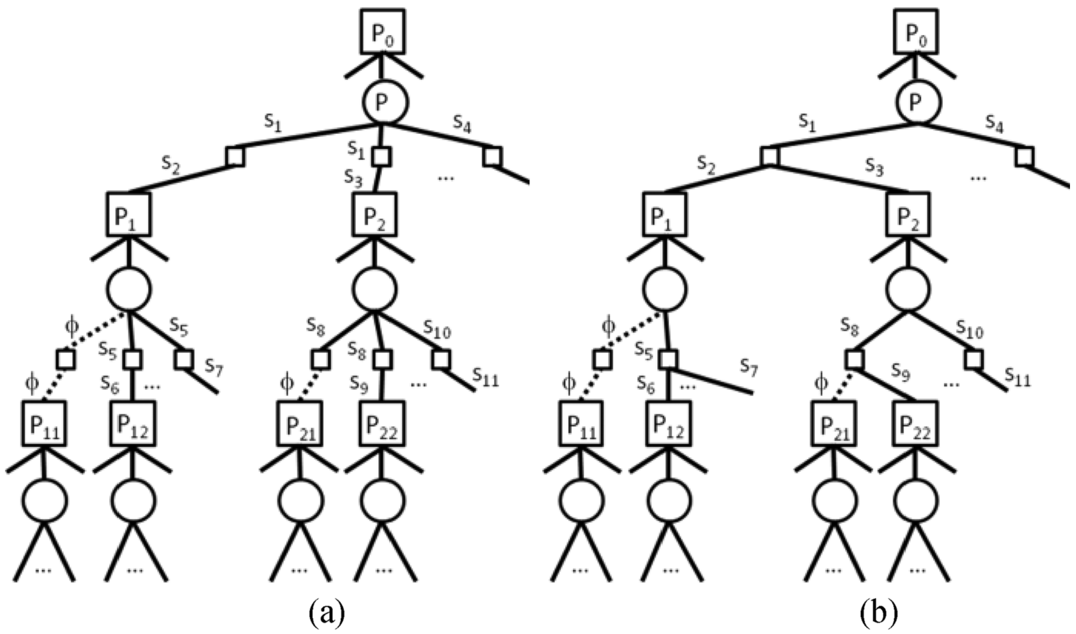


Fig. 6. (a) Marking squares of moves by inserting small boxes. (b) Combining the same edges from (a).

A strategy S of the attacker is viewed as a move-generating function of positions P that are in the attacker's turn. Namely, $S(P)$ indicates the move that the attacker chooses to make according to the strategy S. In a search tree following S, each position P expands at most one move $S(P)$. A strategy S of an attacker is called a winning strategy for position P, if and only if the value of the search tree rooted at P is 1 following S and all defender's legal moves are generated in the tree. Thus, we obtain Corollary 1. A tree as shown in Fig. 5(b) is called a solution tree in [5] and [17].

Corollary 1: The attacker wins in a position P if and only if there exists at least one winning strategy of the attacker in P. ■

In order to investigate more closely squares of defensive moves, insert small rectangles onto the corresponding edges that are broken into two, marked s_1 and s_2, respectively, as shown in Fig. 6(a). Furthermore, the edges are combined with the same s_1, as shown in Fig. 6(b). Note that null stones are marked as ϕ and the corresponding edges are indicated by dashes.

A verifier V (for the attacker) is to verify whether the attacker wins in a position P by following a strategy S. Specifically, if $V(P, S)$ returns the value 1, then the attacker wins in P and S is a winning strategy for P. A straightforward verifier is to verify it by traversing exhaustively the whole solution tree. Clearly, it is infeasible in most cases, especially in case of very large boards or even infinite boards. Fortunately, in *Connect* games, the traversal of the search tree for proof can be greatly reduced according to threats, as described in Section III-C. The traversed search tree for proof by a verifier is called a proof search tree.

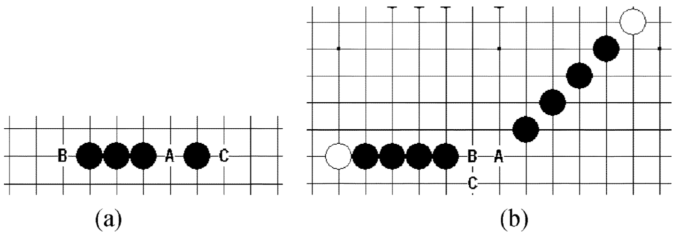


Fig. 7. (a) Normal critical defense. (b) Relaxed critical defense.

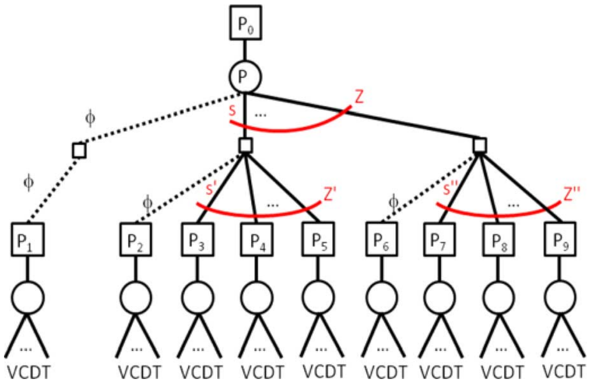


Fig. 8. Proof search tree for solving *Connect(6,2,3)*.

C. Threats

In *Connect6* (other *Connect* games are similar), threats are the key to great reduction of the proof search tree. An active segment in which the attacker occupied four or five squares is called a threat segment of the attacker. The segment poses a threat and the defender has to block it, or the attacker wins by making a win segment in the next move.

Section I has already presented the definition of threat numbers. Examples of the line patterns with one, two, and three threats can be found in [30] and [31]. The defensive moves that block all the threats are called critical defenses, while removing any stones in the moves unblocks some threats. For example, White's seminull moves $M_{D,\phi}(A)$ and moves $M_D(B,C)$ in Fig. 7(a) and (b) are critical defenses, while moves $M_D(A,B)$ are not, because the threats are still blocked without B. (Note that null moves are also critical defenses in positions without any threats according to the above definition.) Critical defenses are said to be normal if the numbers of stones in the defenses are the same as the numbers of threats; and relaxed, otherwise. For example, in Fig. 7, seminull moves $M_{D,\phi}(A)$ are normal, while moves $M_D(B,C)$ are relaxed. In *Connect6*, relaxed critical defenses are not played frequently due to their inefficiency (using two stones to block only one threat).

As described above, threats are the key to great reduction of the proof search tree without going through the entire defensive search tree. For example, for double-threat moves, there are at most four defensive moves. In addition, even for a nonthreat move such as the game *Connect(6,2,3)* described in Section I, Wu *et al.* [30], [31] were able to solve it by using a much smaller proof search tree through considering those defenses in the gray areas shown in Fig. 2. Fig. 8 shows the proof search tree [for solving *Connect(6,2,3)*] that expands $M_{D,\phi\phi}$ first, then $M_{D,\phi}(s)$ for all $s \in Z$, and $M_D(s,s')$ for all $s' \in Z'$, where Z' is the zone derived from $M_{D,\phi}(s)$.

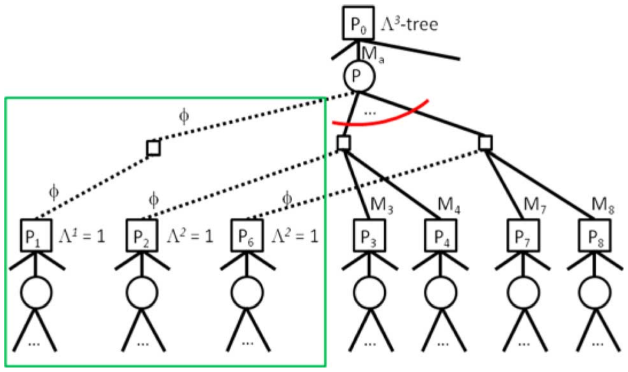


Fig. 9. A Λ^3-tree.

D. Lambda Search

In [22], Thomsen proposed using the lambda search to express how a direct attacker can achieve a goal. In *Connect* games, the goal is normally to make a win segment. The formalization of lambda search is modified for *Connect* games as follows.

Definition 1: In *Connect* games, a Λ^r-tree is a search tree which comprises all legal Λ^r-moves. If a Λ^r-move is an attacker move, the following condition holds. For all subsequent null moves or seminull moves M_D made by the defender, if M_D have exactly u null stones, where $1 \le u \le p$, there exists at least one subsequent Λ^i-tree with value 1, where $0 \le i \le r-u$ or $i=0$ if $r<u$. If a Λ^r-move is a defender move, the following condition holds. There exist no subsequent Λ^i-trees with value 1, where $0 \le i \le r-1$. In a Λ^r-tree, a node is a leaf (without any children) if there are no Λ^r-moves following it. The value of a leaf is 1 if the defender is to move, and 0 if the attacker is to move. The value of a Λ^r-tree is either 1 (indicating that the attacker wins) or 0 (otherwise), derived using minimax calculation. The value of a Λ^0-tree (where the attacker is to move) is simply 1 if the attacker makes a win segment in the next move. ■

In case of $p=1$, the definition of Λ^r is the same as that of λ^r (the goal is to win) in [22]; that is, a Λ^r-tree is a λ^r-tree and a Λ^r-move is a λ^r-move, and *vice versa*. In case of $p=2$, such as *Connect6*, a Λ^3-tree is illustrated in Fig. 9 and move M_a in the tree is a Λ^3-move, since the values of Λ^1-tree and all Λ^2-trees in the left box are all 1. In addition, moves M_3, M_4, M_7, and M_8 are Λ^3-moves, if the attacker has no subsequent Λ^0-moves, Λ^1-moves, or Λ^2-moves. By following the proof of Theorem 1 in [22], we derive the following theorem (whose proof is omitted).

Theorem 1: For a Λ^r-tree rooted in a position P, if a minimax search on it returns value 1, the attacker wins in P. ■

Definition 2: A winning strategy is called a Λ^r-strategy for a position P, if the subsequent nonnull moves following the strategy are all Λ^i-moves, where $0 \le i \le r$. ■

From the above definition, a VCDT is a Λ^1-strategy, while a VCST is a Λ^2-strategy. For example, there exists a Λ^2-strategy for winning position 7 in Fig. 1 (the attacker is White), where moves 8–10 are all Λ^2-moves. VCNTs are Λ^3-strategies or strategies of higher orders, as illustrated in the following. In Fig. 2(a), move M_{623} is a Λ^3-move, and the rest of the attacker

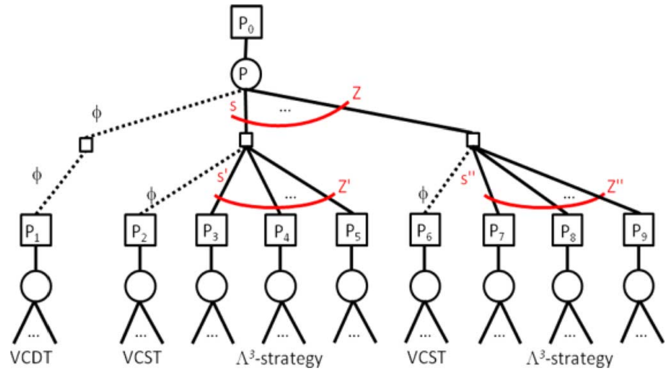


Fig. 10. A Λ^3-strategy.

moves are Λ^1-moves, so it is a Λ^3-strategy for *Connect6(6,2,3)*. In Fig. 3(a), move 7 is a Λ^3-move, and the rest of the attacker moves are Λ^1-moves or Λ^2-moves, so it is a Λ^3-strategy. Fig. 10 shows a general Λ^3-strategy. However, it is more complicated in Fig. 4(a), where move 6 is a Λ^4-move. Section V shows that it is a Λ^4-strategy.

From Definition 2, a Λ^r-strategy, $r \geq 1$, also implies that for a move with u null stones the attacker has a Λ^{r-u}-strategy. For example, in the Λ^3-strategy in Fig. 10, the attacker has a Λ^1-strategy for the null move and Λ^2-strategies for all the seminull moves.

E. Relevance Zones

As seen in Section III-E, the lambda search is a powerful method for proving the winning positions with different orders of threat sequences. The next important issue for lambda search is to construct relevance zones to reduce greatly the search space. In general, different applications construct relevance zones in different ways. In *Connect* games, it is critical to construct relevance zones in order to propagate relevance zones across different orders of threat sequences. For example, in Fig. 10, the relevance zones derived in the VCDT (Λ^1-strategy) or VCSTs (Λ^2-strategies) can be used in the whole search tree (Λ^3-strategy).

This section defines such relevance zones, which are elegantly employed to solve *Connect* games. A set of squares on the board is called a zone. A sequence of zones with size r, $\Psi = \langle Z_1, Z_2, \dots, Z_r\rangle$, is incremental, if the condition $Z_1 \subseteq Z_2 \subseteq \cdots \subseteq Z_r$ holds. In the rest of this paper, sequences of zones with different sizes are all incremental and are thus not explicitly specified. In addition, these zones usually indicate the squares to be chosen for stones to be placed on, so only unoccupied (or empty) squares are of interest.

In a position P, its unoccupied zone, denoted by $Z_{un}(P)$, is the zone that comprises all the unoccupied squares. That is, $Z_{un}(P) = Z_{\text{board}} \backslash Z_P$, where Z_{board} is the zone for the whole board and Z_P is the set of all occupied squares in P. Let $\neg_P(Z)$ denote $Z_{un}(P)\backslash Z$ and indicate the set of unoccupied squares outside Z. Consider a sequence of zones $\Psi = \langle Z_1, Z_2, \dots, Z_r\rangle$ in P. A sequence of unoccupied squares $\varphi = \langle s_1, s_2, \dots, s_{r'}\rangle$, where $r' \leq r$, is said to be outside Ψ or irrelevant to Ψ, if all $s_i \notin Z_i$ or $s_i \in \neg_P(Z_i)$. Let $\varphi \in \neg_P(\Psi)$ denote the relation that φ is irrelevant to Ψ in P.

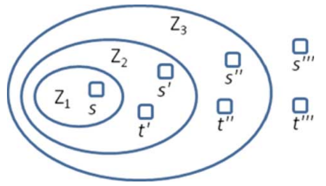


Fig. 11. Sequence of zones $\langle Z_1, Z_2, Z_3\rangle$.

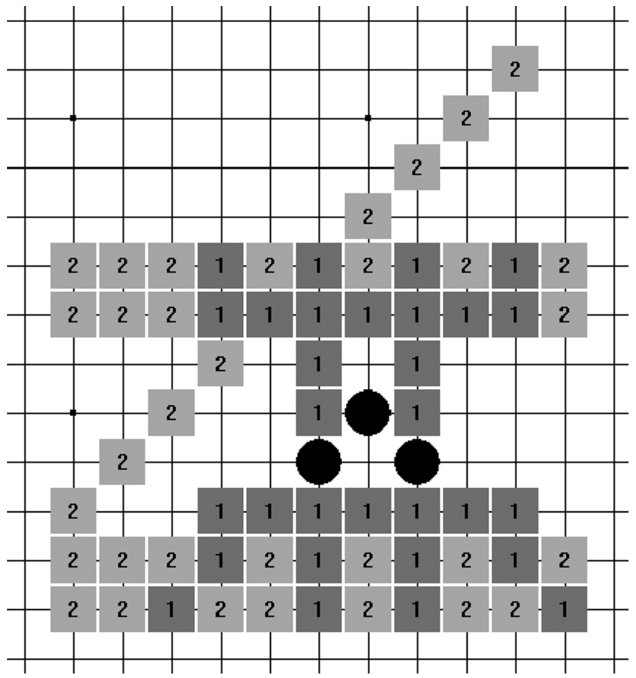


Fig. 12. Sequence of relevance zones $\Psi = \langle Z_1, Z_2\rangle$ for the winning position in Fig. 2(a).

Implicitly, $\neg_P(\Psi)$ denotes $\langle \neg_P(Z_1), \neg_P(Z_2), \dots, \neg_P(Z_r)\rangle$. For example, in Fig. 11, $\langle s', s'', s'''\rangle$, $\langle s', s''\rangle$, $\langle s'', s'''\rangle$, $\langle s'\rangle$, $\langle s'''\rangle$, and even the empty sequence $\langle\rangle$ are all irrelevant to $\langle Z_1, Z_2, Z_3\rangle$, while $\langle s\rangle$, $\langle s', t'\rangle$, $\langle s', s'', t''\rangle$, $\langle s', s'', s''', t'''\rangle$, $\langle s'', s'\rangle$, and $\langle s, s', s''\rangle$ are not. For simplicity, let $\sigma_A(\varphi)$ denote $\sigma_A(s_1) + \sigma_A(s_2) + \cdots + \sigma_A(s_{r'}) = \Sigma_{1\leq i\leq r'}\sigma_A(s_i)$. Similarly, $\sigma_D(\varphi) = \Sigma_{1\leq i\leq r'}\sigma_D(s_i)$.

Definition 3: A sequence of zones Ψ is called a sequence of relevance zones for the attacker in a position P, if and only if the attacker wins in $P + \sigma_D(\varphi)$ for all irrelevant φ; that is, $\varphi \in \neg_P(\Psi)$. Let $RZ(P)$ denote the set of all the sequences of relevance zones for the attacker in P. (Use the notation $RZ(P)$ instead of $RZ_A(P)$, since only relevance zones for the attacker are discussed in this paper). ■

From Definition 3, if $RZ(P)$ is not empty, there must exist some Ψ in $RZ(P)$. This implies that the attacker wins in P by choosing the empty sequence of squares $\langle\rangle$ for φ, since φ is irrelevant to Ψ as described above. Thus, Corollary 2 is obtained.

Corollary 2: If there exists at least one sequence of zones Ψ in $RZ(P)$, then the attacker wins in P. ■

For the winning sequence in Fig. 2(b), Fig. 12 illustrates relevance zones $\Psi = \langle Z_1, Z_2\rangle$, where Z_1 is the set of empty squares marked with a small "1," and Z_2 marked "1" and "2." Note that in the rest of this paper, a sequence of zones is shown in this manner. Interestingly, Z_2 is the same as Z in Fig. 2(b). From observation, Black still wins over all irrelevant $\varphi \in \neg_P(\Psi)$. That is, if White places one in $\neg_P(Z_1)$ and the other in $\neg_P(Z_2)$, Black still wins by replaying the winning sequence in Fig. 2(b). The result is slightly stronger than that in [30] and [31].

Lemma 1 shows an important property that appending extra Z_{board} to a sequence of relevance zones is still in $RZ(P)$. Note that we use Z_{board}, instead of $Z_{un}(P)$, in order to be independent of the position P, for simplicity. For example, in Fig. 12, $\langle Z_1, Z_2, Z_{\text{board}}\rangle$ is also in $RZ(P)$.

Lemma 1: Assume that $\Psi = \langle Z_1, Z_2, \ldots, Z_r\rangle$ is in $RZ(P)$. Then, $\Psi' = \langle Z_1, Z_2, \ldots, Z_r, Z_{\text{board}}\rangle$ is also in $RZ(P)$.

Proof: Consider all irrelevant $\varphi \in \neg_P(\Psi')$. For this lemma, it suffices to prove that the attacker wins in $P + \sigma_D(\varphi)$. Since $\neg_P(Z_{\text{board}})$ is empty, φ must not have the $(r+1)$th item. From the definition, we also obtain $\varphi \in \neg_P(\Psi)$. Since Ψ is assumed to be in $RZ(P)$, the attacker wins in $P + \sigma_D(\varphi)$ due to $\varphi \in \neg_P(\Psi)$. ■

From Lemma 1, two sequences of relevance zones with different sizes can be adjusted to those with the same size by appending extra Z_{board} or removing Z_{board} at the end. For simplicity of the discussion, this paper uses some more notations for operations on sequences of zones with the same size in P, say $\Psi = \langle Z_1, Z_2, \ldots, Z_r\rangle$ and $\Psi' = \langle Z'_1, Z'_2, \ldots, Z'_r\rangle$, as follows.

- Let $\Psi \subseteq \Psi'$ indicate that Ψ is contained in Ψ' pairwise; that is, $Z_i \subseteq Z'_i$ over all $1 \le i \le r$.
- Let $\Psi \cup \Psi' = \langle Z_1 \cup Z'_1, Z_2 \cup Z'_2, \ldots, Z_r \cup Z'_r\rangle$.
- Let $\Psi \cup Z = \langle Z_1 \cup Z, Z_2 \cup Z, \ldots, Z_r \cup Z\rangle$ and $\Psi\backslash Z = \langle Z_1\backslash Z, Z_2\backslash Z, \ldots, Z_r\backslash Z\rangle$, where Z is a zone.
- Let $\Psi \ll 1$ denote $\langle Z_2, Z_3, \ldots, Z_r, Z_{\text{board}}\rangle$ and indicate promotion of the zones in Ψ (that is, shifting zones to the left by 1) with extra Z_{board}. Similarly, let $\Psi \ll 2$ denote $(\Psi \ll 1) \ll 1$, and $\Psi \ll i$ denote $(\Psi \ll (i-1)) \ll 1$, where $i \ge 2$.

From the above notation and definitions, more properties are shown in Lemmas 2 and 3 as follows.

Lemma 2: Assume that Ψ is in $RZ(P)$ and $\Psi \subseteq \Psi'$. Then, Ψ' is also in $RZ(P)$.

Proof: Let $\Psi = \langle Z_1, Z_2, \ldots, Z_r\rangle$ and $\Psi' = \langle Z'_1, Z'_2, \ldots, Z'_r\rangle$. Consider all irrelevant $\varphi \in \neg_P(\Psi')$. It suffices to prove that the attacker wins in $P + \sigma_D(\varphi)$. Since $\Psi \subseteq \Psi'$, the condition $\varphi \in \neg_P(\Psi')$ also implies $\varphi \in \neg_P(\Psi)$. Since Ψ is in $RZ(P)$, the attacker wins in $P + \sigma_D(\varphi)$ due to $\varphi \in \neg_P(\Psi)$. ■

Lemma 3 shows important properties that are employed to improve the verifiers in Section IV.

Lemma 3: Assume that $\Psi = \langle Z_1, Z_2, \ldots, Z_r\rangle$ is in $RZ(P)$. The following two properties are satisfied.

1) Assume that $\neg_P(Z_1)$ is not empty. Let the unoccupied square be $s \in \neg_P(Z_1)$. Then, $\Psi \ll 1$ is in $RZ(P+\sigma_D(s))$.
2) Let φ be a sequence of unoccupied squares $\langle s1, s2, \ldots, sr'\rangle$ in $\neg_P(\Psi)$, where $r' \le r$. Then, $\Psi \ll r'$ is in $RZ(P + \sigma_D(\varphi))$.

Proof: It suffices to prove the first property, since the first implies the second by induction.

Let $\Psi' = \Psi \ll 1$ and consider all irrelevant $\varphi' = \langle s_2, \ldots, s_{r'}\rangle \in \neg_P(\Psi')$, where $r' \le r$. For the first property, it suffices to prove that the attacker wins in $(P+\sigma_D(s))+\sigma_D(\varphi')$.

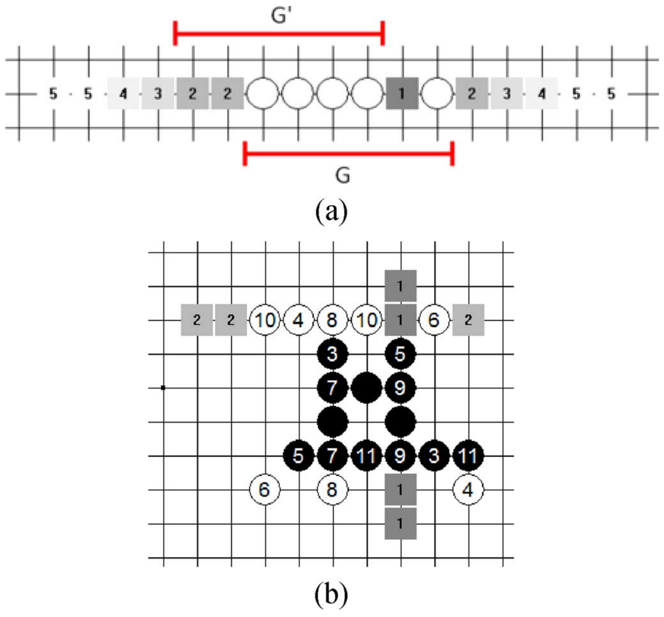


Fig. 13. Relevance zones (a) in a line and (b) in a board, upon winning with a win segment.

Let $\varphi = \langle s, s_2, \ldots, s_{r'}\rangle$. Then, the condition $\varphi \in \neg_P(\Psi)$ holds due to $s \in \neg_P(Z_1)$. Since Ψ is in $RZ(P)$, the attacker wins in $P + \sigma_D(\varphi)$ due to $\varphi \in \neg_P(\Psi)$; that is, the attacker wins in $(P + \sigma_D(s)) + \sigma_D(\varphi')(= P + \sigma_D(\varphi))$. ■

IV. RZOP Search for *Connect6*

For solving positions in *Connect6*, this section investigates a verifier $V(P, S)$ that also constructs recursively a sequence of zones $\Psi(P) = \langle Z_1(P), Z_2(P), \ldots, Z_r(P)\rangle$ with the following property.

Property RZV: In the case that $V(P, S)$ returns value 1, the sequence of zones $\Psi(P)$ constructed by $V(P, S)$ is in $RZ(P)$.

This section presents such a verifier, named $V_{C6}(P, S)$, with a new proof search method for *Connect6*. This method will be generalized to all *Connect* games in the Appendix. The verifier $V_{C6}(P, S)$ is described in Sections IV-B–IV-D respectively for three distinct kinds of P, namely, endgame positions, positions in the attacker's turn, and positions in defender's turn. Finally, Section IV-E concludes with Theorem 2, showing that the verifier satisfies Property RZV in all cases.

A. Endgame Positions

If the attacker does not win in the endgame position P, the verifier simply returns the value 0. If the attacker wins in P (i.e., the attacker has a win segment in P), the verifier returns 1 and constructs $\Psi(P)$ in the following operation.

EP-1) For each active segment G of the defender containing exactly i unoccupied squares, these squares in G are all added into $Z_i(P)$ or higher order zones; that is, $Z_j(P)$ for all $j \ge i$. In other words, for each active segment G of the defender containing at most i unoccupied squares, add all of these squares in G into $Z_i(P)$.

Let us illustrate the above operation by the line shown in Fig. 13(a), where the defender is White. Following the operation, the square marked with "1" is in Z_1, those marked with

"1" or "2" are in Z_2, and so on. For example, segment G has only one unoccupied square that is in Z_1 or higher order zones, while segment G' has two unoccupied squares that are in Z_2 or higher order zones. It is observed that placing one white stone on the square in Z_1 forms a counter win segment (e.g., G) or an inversion that may prevent the attacker from winning. Note that if the defender has an inversion, this position P is unreachable since neither can have win segments simultaneously (as described in the previous section); who wins first is thus unknown. On the other hand, the attacker still wins if one white stone is placed in square s_1, where $s_1 \notin Z_1$. Similarly, the attacker still wins if one white stone is placed on s_1, where $s_1 \notin Z_1$, and the other on s_2, where $s_2 \notin Z_2$. The above can be generalized to higher orders, and to all lines (or segments) on a board. An example of constructing zones $\langle Z_1, Z_2\rangle$ on a board is illustrated in Fig. 13(b). Note that move 10 in the figure is simply one of all the defenses and is chosen for an illustration. In addition, since move 9 clearly wins already, Section IV-D will describe how to speed up the establishment of relevance zones.

From the above observation, it can be derived that the constructed $\Psi(P)$ in operation EP-1 is in $RZ(P)$. This implies that $V_{C6}(P,S)$ satisfies Property RZV in the case of endgame P, as shown in Lemma 4.

Lemma 4: Assume P to be an endgame position. Property RZV is satisfied for $V_{C6}(P,S)$.

Proof: Omitted. ■

In *Connect6*, all $Z_i(P)$ with $i \geq 6$, are nearly the same as $Z_{un}(P)$, except for those unoccupied squares covered by none of the active segments of the defender. For example, if an unoccupied square is surrounded by the attacker's squares, it is clearly covered by none of the active segments of the defender and is not included in these $Z_i(P)$. However, there are normally not many such squares, especially when board sizes are large and only a small number of stones are in positions. Practically, we simply ignore all $Z_i(P)$ with $i \geq 6$ or use $Z_{un}(P)$ whenever needed.

B. Positions in the Attacker's Turn

In such positions, the attacker simply follows strategy S to make the move $S(P)$ in P. Let P_A denote $P \oplus S(P)$. This verifier first performs $V_{C6}(P_A,S)$ recursively. If $V_{C6}(P_A,S)$ returns the value 0, this verifier $V_{C6}(P,S)$ also returns 0. On the other hand, if $V_{C6}(P_A,S)$ returns 1, this verifier $V_{C6}(P,S)$ returns 1 as well, and constructs $\Psi(P)$ in the following operation.

AT-1) Let $\Psi(P) = \Psi(P_A) \cup Z_S$, where $Z_S = \{s | s \in S(P)\}$.

Intuitively, placing any stones on the squares in Z_S by the defender in advance may block the attacks and prevent the attacker from winning. In this sense, the squares in Z_S are relevant and are therefore contained in all $Z_i(P)$ (or $\Psi(P)$).

In fact, the above operation AT-1 also implies the property $\neg_P\Psi(P) = \neg_{P_A}\Psi(P_A)$ for the following reason. From the operation, the condition $Z_i(P) = Z_i(P_A) \cup Z_S$ holds for all i. In addition, since $P_A = P \oplus S(P)$, it is clear that $Z_{un}(P_A) = Z_{un}(P) \backslash Z_S$ or $Z_{un}(P) = Z_{un}(P_A) \cup Z_S$. Thus, for all i, we derive

$$\begin{aligned}\neg_P Z_i(P) &= Z_{un}(P) \backslash Z_i(P)\\ &= (Z_{un}(P_A) \cup Z_S) \backslash (Z_i(P_A) \cup Z_S)\\ &= Z_{un}(P_A) \backslash Z_i(P_A) = \neg_{P_A} Z_i(P_A).\end{aligned}$$

From this property, Lemma 5 shows that this verifier $V_{C6}(P,S)$ satisfies Property RZV if $V_{C6}(P_A,S)$ satisfies Property RZV.

Lemma 5: Assume a position P in the attacker's turn. From the above, assume that $V_{C6}(P_A,S)$ satisfies Property RZV, where $P_A = P \oplus S(P)$. This verifier $V_{C6}(P,S)$ satisfies Property RZV.

Proof: Assume that this verifier $V_{C6}(P,S)$ returns the value 1. For this lemma (this verifier satisfies Property RZV), it suffices to prove that the constructed $\Psi(P)$ is in $RZ(P)$. From the above operation, $V_{C6}(P_A,S)$ must also return 1. Since $V_{C6}(P_A,S)$ satisfies Property RZV from the lemma, $\Psi(P_A)$ is in $RZ(P_A)$.

Consider all irrelevant φ, where $\varphi \in \neg_P\Psi(P)$. It suffices to prove that the attacker wins in $P + \sigma_D(\varphi)$. Since the property $\neg_P\Psi(P) = \neg_{P_A}\Psi(P_A)$ is satisfied as described above, the condition $\varphi \in \neg_{P_A}\Psi(P_A)$ holds as well. Since $\Psi(P_A)$ is in $RZ(P_A)$ from the above, the attacker wins in $P_A + \sigma_D(\varphi)$ due to $\varphi \in \neg_{P_A}\Psi(P_A)$. Since the attacker wins in $P_A + \sigma_D(\varphi) = (P + \sigma_D(\varphi)) \oplus S(P)$, the attacker wins in $P + \sigma_D(\varphi)$ by choosing the move $S(P)$. ■

C. Positions in the Defender's Turn

For positions in the defender's turn, Lemma 6 shows a very important property used in this section as well as in the Appendix.

Lemma 6: Assume a position P in the defender's turn. For a given sequence of zones Ψ, assume that for all defender moves M_D there exists some Ψ_D such that $\Psi_D \subseteq \Psi$ and Ψ_D is in $RZ(P \oplus M_D)$. Then, Ψ is in $RZ(P)$.

Proof: Consider all irrelevant $\varphi \in \neg_P\Psi$. For this lemma, it suffices to prove that the attacker wins in $P + \sigma_D(\varphi)$.

Now, consider all defender moves M_D in $P + \sigma_D(\varphi)$. From this lemma, there exists some Ψ_D such that $\Psi_D \subseteq \Psi$ and Ψ_D is in $RZ(P \oplus M_D)$. Since $\Psi_D \subseteq \Psi$, the condition $\varphi \in \neg_P\Psi$ implies $\varphi \in \neg_P\Psi_D$. Since squares in M_D and $\sigma_D(\varphi)$ are mutually exclusive, $\varphi \in \neg_P\Psi_D$ also implies $\varphi \in \neg_{P\oplus M_D}\Psi_D$. Since Ψ_D is in $RZ(P \oplus M_D)$ from the above, the attacker wins in $(P \oplus M_D) + \sigma_D(\varphi)$ due to $\varphi \in \neg_{P\oplus M_D}\Psi_D$. Since $(P \oplus M_D) + \sigma_D(\varphi) = (P + \sigma_D(\varphi)) \oplus M_D$, the attacker also wins in $(P + \sigma_D(\varphi)) \oplus M_D$. From the above, since the attacker wins in $(P + \sigma_D(\varphi)) \oplus M_D$ over all defender moves M_D, the attacker wins in $P + \sigma_D(\varphi)$. ■

A straightforward verifier is to verify whether the attacker wins for all defender moves, as follows. The verifier $V_{C6}(P,S)$ returns value 1, if the recursive $V_{C6}(P \oplus M_D, S)$ returns 1 for all defender moves M_D; otherwise, it returns 0. In the case that this verifier $V_{C6}(P,S)$ returns 1, the zones $\Psi(P)$ are constructed in the following operation.

DT-1) Initialize all zones in $\Psi(P)$ to be empty. Then, for all defender moves M_D, let $\Psi(P) = \Psi(P) \cup \Psi(P \oplus M_D)$.

From the above operation, the condition $\Psi(P \oplus M_D) \subseteq \Psi(P)$ clearly holds for all M_D. Assume that all the recursive $V_{C6}(P \oplus M_D, S)$ satisfy Property RZV. Then, all $\Psi(P \oplus M_D)$ are in $RZ(P \oplus M_D)$ for all defender moves M_D. From Lemma 6, we obtain that $\Psi(P)$ is in $RZ(P)$; and therefore, the verifier satisfies Property RZV. By induction, the above straightforward verifier satisfies Property RZV in all cases.

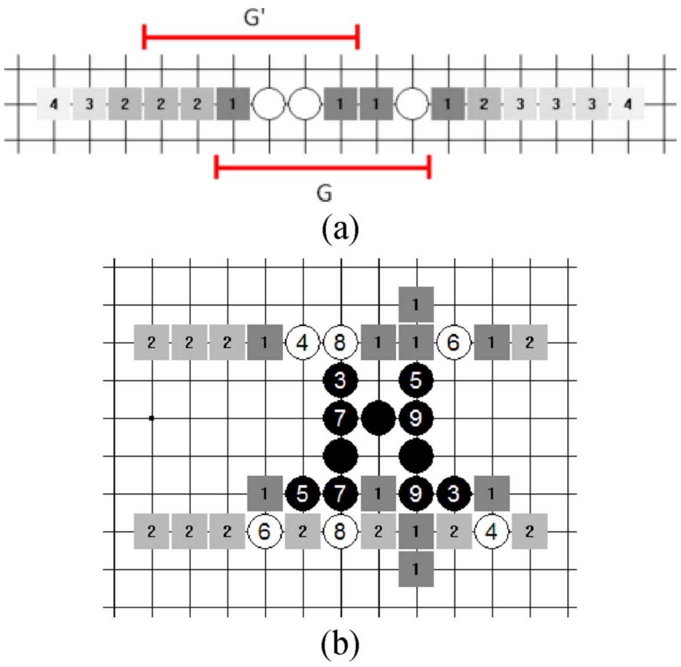


Fig. 14. Relevance zones (a) in a line and (b) in a board, upon winning with three or more threats.

However, the above straightforward verifier is apparently inefficient, since it searches exhaustively all defender moves, even when the attacker moves have some threats. The situation is even worse in the case that the board size is very large or infinite. In this section, an efficient and elegant verifier is devised to reduce the search space by making use of both threats and relevance zones. In *Connect6*, the position P (in the defender's turn) can be classified into the following four cases. The number of the attacker threats in P is 1) three or more, 2) two, 3) one, and 4) zero. The four cases are discussed, respectively, in the following.

1) Three Threats or More: In this case, the attacker is sure to win by simply following the strategy S_{3T} as follows. For each defender move, since the move must leave some threat segments unblocked, the attacker wins simply by making a win segment from the unblocked one. Since the strategy is a sure win, the verifier returns value 1 and constructs the zones (initialized to be empty) in the following operations.

T3-1) Add all unoccupied squares s on threat segments into all $Z_i(P)$.

T3-2) For each active segment G of the defender containing exactly $i + 2$ unoccupied squares, all these squares in G are added into all $Z_j(P)$ or higher order zones. In other words, for each active segment G of the defender containing at most $i + 2$ unoccupied squares, add all these squares in G into $Z_i(P)$.

Let us illustrate the above operations by the line shown in Fig. 14(a), where the defender is White. Zones in the line are marked in a way similar to that in Fig. 13(a). It is observed that placing one white stone in G or Z_1 results in a counter threat segment or an inversion that may threaten the attacker to defend in some of his earlier moves and prevent the attacker from winning. On the other hand, the attacker still wins if one white stone is placed on other squares s_1, where $s_1 \notin Z_1$. Similarly, the attacker still wins if one white stone is placed on s_1, where $s_1 \notin Z_1$, and the other on s_2, where $s_2 \notin Z_2$. The above can be generalized to higher orders, and to all lines (or segments) on the board. An example of constructing two zones $\langle Z_1, Z_2\rangle$ on a board is illustrated in Fig. 14(b). Lemma 7 shows that in this case the verifier satisfies Property RZV; that is, $\Psi(P)$ is in $RZ(P)$.

Lemma 7: Assume that the defender is to move and the attacker has three or more threats in P. The verifier described above satisfies Property RZV.

Proof: For this lemma, it suffices to prove that the constructed $\Psi(P)$ is in $RZ(P)$. Consider all defender moves M_D. The attacker simply follows a strategy S_{3T} to connect six from an unblocked threat segment. Let $P_D = P \oplus M_D$ and $P_6 = P_D \oplus S_{3T}(P_D)$. From Lemmas 4 and 5, $\Psi(P_6)$ and $\Psi(P_D)$ are in $RZ(P_6)$ and $RZ(P_D)$, respectively.

To prove that $\Psi(P)$ is in $RZ(P)$, it suffices to prove from Lemma 6 that $\Psi(P_D) \subseteq \Psi(P)$, since $\Psi(P_D)$ is already in $RZ(P_D)$. From Section IV-C, $\Psi(P_D) = \Psi(P_6) \cup Z_S$, where $Z_S = \{s | s \in S_{3T}(P_D)\}$. From operation T3-1, all squares in Z_S are added into $\Psi(P)$. Thus, it suffices to prove that $\Psi(P_6) \subseteq \Psi(P)$.

Since the attacker connects six in P_6, operation EP-1 (in Section IV-B) is employed to construct zones $\Psi(P_6)$. The operation is restated as follows. For each active segment G of the defender containing at most i unoccupied squares in P_6, all the squares in G are added into $Z_i(P_6)$. Since one move has at most two squares, at most two occupied squares in G were occupied by move M_D. Therefore, G contains at most $2 + i$ unoccupied squares back in P (before making move M_D). From operation T3-2, all these unoccupied squares are also added into $Z_i(P)$. For example, let both lines in Figs. 13(a) and 14(a) be, respectively, in positions P_6 and P, where move M_D is placed on the two leftmost squares marked "1" in segment G in Fig. 14(a). Thus, the two squares marked "2" in segment G' in Fig. 13(a) are also added into $Z_2(P)$ in Fig. 14(a). From the above observation, we can derive $\Psi(P_6) \subseteq \Psi(P)$. ■

Since all active segments G of the defender contain at most $6 (= 4 + 2)$ unoccupied squares in *Connect6*, all these squares in G are added into all $Z_i(P)$ from operation T3-2, where $i \geq 4$. Thus, these $Z_i(P)$ are nearly the same as $Z_{un}(P)$, except for the unoccupied squares not covered by any active segments of the defender, e.g., the unoccupied squares surrounded by all the attacker squares. Similar to the argument in Section IV-C, we construct zones with size three, and simply use $Z_{un}(P)$ for those higher order zones, whenever needed.

2) Two Threats: When the attacker has two threats in P, the defender must defend by blocking the two threats. In this case, the verifier performs the following operations.

T2-1) For each defender move M_D that blocks the two threats, perform the following.

a) Return value 0 if the recursive $V_{C6}(P_D, S)$ returns value 0, where $P_D = P \oplus M_D$.

b) Let $\Psi(P) = \Psi(P) \cup \Psi(P_D)$.

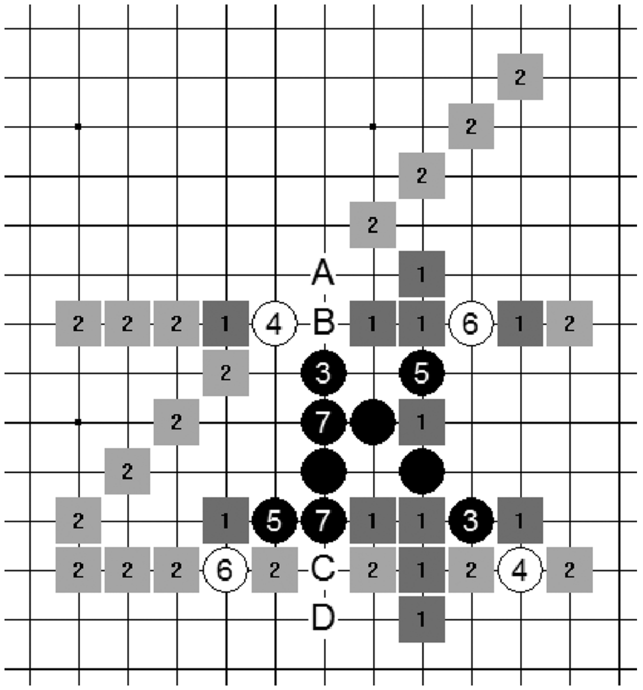


Fig. 15. Winning position with two threats for Black (the attacker) and the constructed $\Psi(P)$.

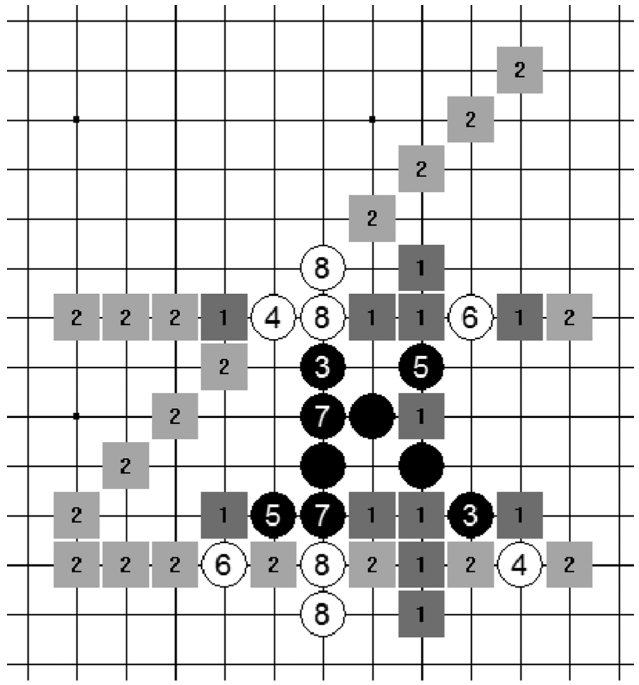


Fig. 16. Combining three defensive moves into one with four stones.

T2-2) Continue to construct zones by both operations T3-1 and T3-2, and return 1.

For example, for position P in Fig. 15 [the grandparent of the position in Fig. 14(b)] where Black has two threats, White has three defensive moves at (B,C), (A,C), and (B,D). Obviously, since Black still wins for each of the three moves, Black wins in P. From the above operations, this verifier returns value 1 and constructs $\Psi(P)$ as shown in Fig. 15. Lemma 8 shows that this verifier satisfies Property RZV if the verifier satisfies Property RZV for all the defensive moves as well. From this lemma, $\Psi(P)$ in Fig. 15 is in $RZ(P)$.

Lemma 8: From the above, assume that the defender is to move and the attacker has two threats in P. Assume that all the recursive $V_{C6}(P_D, S)$ in operation T2-1 satisfy Property RZV. Then, the verifier $V_{C6}(P, S)$ satisfies Property RZV as well.

Proof: Assume that this verifier $V_{C6}(P, S)$ returns 1. For this lemma (this verifier satisfies Property RZV), it suffices to prove that the constructed $\Psi(P)$ is in $RZ(P)$. Since $V_{C6}(P, S)$ returns 1, all the recursive $V_{C6}(P_D, S)$ in operation T2-1 must return 1. Since these $V_{C6}(P_D, S)$ satisfy Property RZV from this lemma, all constructed $\Psi(P_D)$ are in $RZ(P_D)$.

To prove $\Psi(P) \in RZ(P)$, it suffices to prove from Lemma 6 the following. For all defender moves M_D, there exists some Ψ_D such that Ψ_D is in $RZ(P \oplus M_D)$ and $\Psi_D \subseteq \Psi(P)$. All defender moves M_D are classified into the following cases.

1) All defender moves M_D that block both threats. From the above, $\Psi(P_D)$ are in $RZ(P_D)$. In addition, since these $\Psi(P_D)$ are merged into $\Psi(P)$ in operation T2-1b, we obtain $\Psi(P_D) \subseteq \Psi(P)$. Thus, $\Psi(P_D)$ is the Ψ_D.
2) All defender moves M_D that leave some threat segment unblocked. The attacker wins by connecting six on the segment, like strategy S_{3T}. Since operation T2-2 follows those steps in T3-1 and T3-2, we simply follow the proof of Lemma 7 to prove that there exists some Ψ_D such that $\Psi_D \subseteq \Psi(P)$ and Ψ_D is in $RZ(P_D)$. ∎

Assume that the subsequent winning moves of the attacker are the same for all the defensive moves. Then, we can optimize the construction of zones by combining these defensive moves together. For example, in Fig. 15, the three defensive moves, (B,C), (A,C), and (B,D), can be combined into a macromove (A, B, C, D) as shown in Fig. 16. Since the subsequent winning sequences of the attacker are the same, the sizes of relevance zones are relatively smaller and the threat-based search is also greatly reduced. However, note that the segment containing both A and B (the same for C and D) in Fig. 15 should be considered to have one white stone only for zone construction. Since the winning sequences in Fig. 2(b) are the same for all defensive moves, the relevance zones are constructed as shown in Fig. 12.

3) One Threat: When the attacker has one threat, the defender must defend by blocking the threat. In this case, the verifier performs the following operations.

T1-1) For each normal critical defense (defined in Section III-C), $M_{D,\phi}(s)$, where square s blocks the threat, perform the operation of seminull-move proof search as follows.
  a) Return value 0, if the recursive $V_{C6}(P_s, S)$ returns 0 where $P_s = P \oplus M_{D,\phi}(s)$.
  b) Let $\Psi(P) = \Psi(P) \cup (\Psi(Ps) \ll 1)$.
  c) For each defensive move $M_D(s, s')$, where $s' \in Z_1(P_s)$, perform both operations T2-1a and T2-1b.

T1-2) For all relaxed critical defenses $M_D(s, s')$, perform both operations T2-1a and T2-1b.

T1-3) Perform both operations T3-1 and T3-2, and return 1.

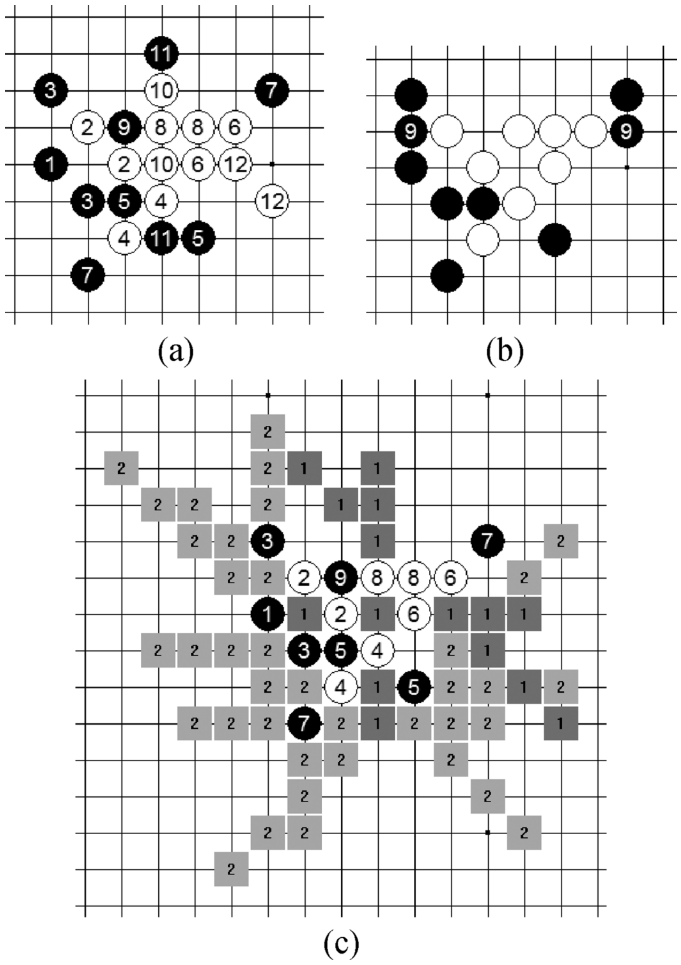


Fig. 17. (a) VCDT for the seminull move 9. (b) Relaxed critical defense at 9. (c) Constructed zones for the seminull move 9 in (a).

Consider a position P, 8 in Fig. 17(a) (the same as 8 in Fig. 1), and another P_s, with a seminull move added at 9. White (the attacker) wins in P_s by the winning sequence in Fig. 17(a). The above operations construct the zones $\Psi(P_s) = \langle Z_1(P_s), Z_2(P_s), Z_3(P_s)\rangle$, with the first two zones shown in Fig. 17(c). According to operation T1-1b, both zones $Z_2(P_s)$ and $Z_3(P_s)$ are shifted and merged into $Z_1(P)$ and $Z_2(P)$, respectively. For all defensive moves $M_D(s, s')$, where $s' \in Z_1(P_s)$, operation T1-1c follows both T2-1a and T2-1b to construct zones and verify whether $V_{C6}(P \oplus M_D(s, s'), S)$ return 1. In addition, operation T1-2 also performs the same for all relaxed critical defenses, such as the one in Fig. 17(b). From Fig. 17(c), since the number of squares in $Z_1(P_s)$ is only 15, the number of recursive V_{C6} is relatively small, even in very large or infinite boards.

Lemma 9 shows that the verifier satisfies Property RZV if all the recursive V_{C6} satisfy Property RZV.

Lemma 9: From the above, assume that the defender is to move and the attacker has one threat in P. Assume that all the recursive V_{C6} in both operations T1-1 and T1-2 satisfy Property RZV. Then, the verifier $V_{C6}(P, S)$ satisfies Property RZV as well.

Proof: Assume that this verifier $V_{C6}(P, S)$ returns 1. For this lemma, it suffices to prove that the constructed $\Psi(P)$ is in $RZ(P)$. Since $V_{C6}(P, S)$ returns 1, all the recursive V_{C6} in both operations T1-1 and T1-2 must also return 1. Since all the recursive V_{C6} satisfy Property RZV from this lemma, all $\Psi(P_s)$ constructed from T1-1a are in $RZ(P_s)$ and all $\Psi(P_D)$ from T1-1c and T1-2 are in $RZ(P_D)$.

To prove $\Psi(P) \in RZ(P)$, it suffices to prove from Lemma 6 the following. For all defender moves M_D, there exists some Ψ_D such that Ψ_D is in $RZ(P \oplus M_D)$ and $\Psi_D \subseteq \Psi(P)$. All defender moves M_D are classified into the following cases.

1) All defender moves $M_D(s, s')$ where s blocks the threat as described in T1-1. Let $P_s = P \oplus M_{D,\phi}(s)$. Furthermore, this case is separated into the following two subcases.
   a) $s' \in Z(P_s)$. Let P_D denote $P \oplus M_D(s, s')$. The zone $\Psi(P_D)$ is constructed in operation T1-1c, and is in $RZ(P_D)$ according to the first paragraph of this proof. Since $\Psi(P_D)$ is merged into $\Psi(P)$ in T1-1c, we obtain $\Psi(P_D) \subseteq \Psi(P)$. Thus, $\Psi(P_D)$ is the ΨD.
   b) $s' \in \neg_{P_s}(Z_1(P_s))$. From the above, $\Psi(P_s)$ is in $RZ(P_s)$. Since $s' \in \neg_{P_s}(Z_1(P_s))$, Lemma 3 shows that $\Psi(P_s) \ll 1$ is in $RZ(P_s + \sigma_D(s'))$, meaning $RZ(P \oplus MD(s, s'))$. From operation T1-1b, $(\Psi(P_s) \ll 1) \subseteq \Psi(P)$. Thus, $\Psi(P_s) \ll 1$ is Ψ_D.
2) All defender moves $M_D(s, s')$ in operation T1-2 are relaxed critical defenses. The proof is similar to that in case 1a and therefore omitted.
3) All defender moves $M_D(s, s')$ that do not block the threat. The attacker wins by connecting six on some unblocked threat segments, like strategy S_{3T}. Find Ψ_D by following the proof of Lemma 7. ■

4) No Threats: When the attacker has no threats, it becomes more complicated since the defender has much more freedom to move. In this case, the verifier makes use of the constructed relevance zones to minimize the search space in the following operations.

T0-1) Return value 0 if $V_{C6}(P_\phi, S)$ returns 0, where $P_\phi = P \oplus M_{D,\phi\phi}$.
T0-2) Let $\Psi(P) = \Psi(P_\phi) \ll 2$.
T0-3) For each square s in $Z_2(P_\phi)$, perform the seminull move proof search, as in operations T1-1a to T1-1c.
T0-4) Return 1.

Let us illustrate the above operations by the example in Fig. 2. From the winning moves in Fig. 2(b), operation T0-1 constructs relevance zones $\Psi(P_\phi) = \langle Z_1(P_\phi), Z_2(P_\phi), Z_3(P_\phi)\rangle$, with only the first two zones shown in Fig. 12. Similarly, zone $Z_2(P_\phi)$ is the same as Z in Fig. 2(b). According to operation T0-2, zone $Z_3(P_\phi)$ is shifted and merged into $Z_1(P)$. Then, in operation T0-3, one square s in $Z_2(P_\phi)$ is chosen to perform the seminull move proof search. In the case that 2 in Fig. 2(c) is chosen, the seminull move proof search in T0-3 constructs the relevance zones $\Psi(P_s) = \langle Z_1(P_s), Z_2(P_s), Z_3(P_s)\rangle$, where $P_s = P \oplus M_{D,\phi}(s)$. Zone $Z_1(P_s)$ is actually the same as Z' in Fig. 2(c). After verifying that White wins for all $s \in Z_2(P_\phi)$ and all $s' \in Z_1(P_s)$, the verifier confirms that White wins in P, as shown in Lemma 10. For the position in Fig. 2, the number of the recursive V_{C6} in T0-1–T0-3 is 2656, relatively small when compared with the number of legal moves.

Lemma 10: Assume that the defender is to move and the attacker has no threats in P. From the above, assume that all recursive V_{C6} in both operations T0-1 and T0-3 satisfy Property RZV. Then, the verifier $V_{C6}(P, S)$ also satisfies Property RZV.

Proof: Assume that this verifier $V_{C6}(P, S)$ returns 1. For this lemma, it suffices to prove that the constructed $\Psi(P)$ is in $RZ(P)$. Since $V_{C6}(P, S)$ returns 1, all the recursive V_{C6} in both operations T0-1 and T0-3 must also return 1. Since these recursive V_{C6}, say for position P', satisfy Property RZV from this lemma, the constructed zones $\Psi(P')$ are in $RZ(P')$.

To prove $\Psi(P) \in RZ(P)$, it suffices to prove from Lemma 6 the following: for all defender moves M_D, there exists some Ψ_D such that Ψ_D is in $RZ(P \oplus M_D)$ and $\Psi_D \subseteq \Psi(P)$. All defender moves M_D are classified into the following cases.

1) All defender moves $M_D(s, s')$ where $s \in \neg_{P_\phi}(Z_2(P_\phi))$ and $s' \in \neg_{P_\phi}(Z_2(P_\phi))$. From the first paragraph in this proof, $\Psi(P_\phi)$ is in $RZ(P_\phi)$. Since $s \in \neg_{P_\phi}(Z_2(P_\phi))$ and $s' \in \neg_{P_\phi}(Z_2(P_\phi))$, $\Psi(P_\phi) \ll 2$ is in $RZ(P_\phi + \sigma_D(s) + \sigma_D(s'))$ from Lemma 3. Since $P\phi + \sigma_D(s) + \sigma_D(s') = P \oplus M_D(s, s')$, $\Psi(P_\phi) \ll 2$ is also in $RZ(P \oplus M_D(s, s'))$. In addition, $(\Psi(P_\phi) \ll 2) \subseteq \Psi(P)$ from operation T0-2. Thus, $\Psi(P_\phi) \ll 2$ is Ψ_D.
2) All defender moves $M_D(s, s')$ where $s \in Z_2(P_\phi)$. By following the proof for case 1 (including subcases 1a and 1b) in Lemma 9, we obtain that there exists some Ψ in $P \oplus M_D(s, s')$ for all s' such that $\Psi \subseteq \Psi(P)$. The details are omitted. ■

D. Conclusion

Theorem 2 concludes that the verifier $V_{C6}(P, S)$ in all cases satisfies Property RZV. Therefore, if $V_{C6}(P, S)$ returns value 1, the constructed $\Psi(P)$ is in RZ(P), and the attacker wins in P from Corollary 2.

Theorem 2: The verifier $V_{C6}(P, S)$ satisfies Property RZV in all cases.

Proof: By induction, the verifier $V_{C6}(P, S)$ satisfies Property RZV in all cases from Lemma 4 to Lemma 10. ■

V. Solving *Connect6* Positions

In Section IV, we present a verifier $V_{C6}(P, S)$ to verify whether the attacker wins in a *Connect6* position P by following strategy S. However, in order to solve positions, we still need to provide the verifier with winning strategies S. Winning strategies can be provided in the following three ways.

1) Let human experts offer the winning strategies manually.
2) Let programs find the winning strategies automatically.
3) Find the winning strategies by mixing the above two.

Traditionally, experts used the first way to claim that some positions are winning, e.g., *Go-Moku* and *Renju* [18]. However, it becomes complicated and tedious for human players to traverse all positions to prove it thoroughly. Hence, it is more feasible to solve these positions by programs using the second way. However, programs may not be smart enough sometimes to find the correct winning moves. Therefore, some researchers chose the third way by following experts' suggestions for some opening moves and then letting programs solve the subsequent moves. For example, Allis [1], [2] solved *Go-Moku* in the free style, and Wágner and Virág [23] solved *Renju*. In Section V-A, we developed some assistant programs to help find the winning strategies for *Connect6*. In Section V-B, we illustrate our new proof search method in Section IV by solving the positions in Figs. 3(a) and 4(a). Finally, we give more results in Section V-C.

A. Assistant Programs

This section describes some assistant programs developed for solvers and verifiers. Given a position P in the attacker's turn, a solver is to return a winning move as well as the relevance zones, if found; and, otherwise, a null move is returned to indicate failure of finding a winning move. A solver of finding a VCDT strategy, denoted by S_{VCDT}, is described as follows.

1) If there exist connect-six moves or triple-threat-or-higher moves, simply choose one of them to win.
2) Evaluate all the double-threat moves and choose some *good* ones for further expansion (according to the evaluations).
3) For each chosen move M, return M if $V_{C6}(P \oplus M, S_{\text{VCDT}})$ returns 1.
4) Return the null move to indicate failure of finding a winning move.

A solver of finding a VCST (VCNT) is similar to the above, except that single-threat (nonthreat) moves are also evaluated and chosen at step 2. Actual solvers are implemented in a more complicated way to reduce the size of a search tree and control the timing. For example, the techniques of iterative deepening and transposition table are normally incorporated.

In this paper, we implemented a solver with VCDT, named VCDT-Solver, and another solver with VCST, named VCST-Solver. More accurately, the VCDT-Solver is to find a Λ^1-strategy, while the VCST-Solver is to find a Λ^2-strategy. Our VCST-Solver also tends to find VCDTs, if any, unless some single-threat moves are evaluated to be much better. Currently, this solver is able to find a Λ^2-strategy up to depth 25 where the size of the longest path with Λ^2-moves is 13. This solver was also incorporated into our *Connect6* program NCTU6, which won the gold at the 11th and 13th Computer Olympiads [26], [33] in 2006 and 2008, respectively; and also won eight games and lost none against top *Connect6* players in Taiwan in 2009 [12]. From our experience, VCST-Solver is able to find Λ^2-strategies, if any, in most cases accurately.

Regarding solvers for Λ^3-strategies or strategies of higher orders, the time complexities become much higher, since the numbers of defensive moves to be verified grow much higher. Therefore, we did not implement it directly.

First, we implemented a verifier, named NCTU6-Verifier, to verify whether the attacker wins for all defender moves. In other words, given a position P in the defender's turn as shown in Fig. 18(a), the verifier uses VCDT-Solver for null moves and VCST-Solver for all seminull moves and nonnull moves. If null and seminull moves are all solved, then move M (from the parent of P to P) in Fig. 18(a) is an attacker Λ^3-move. If some nonnull moves are not solved by VCST-Solver, these moves are reported or generated. Note that the defender Λ^3-moves must be reported. Since our VCST-Solver can find Λ^2-strategies accurately in most cases, most reported moves are the defender Λ^3-moves in our experiments.

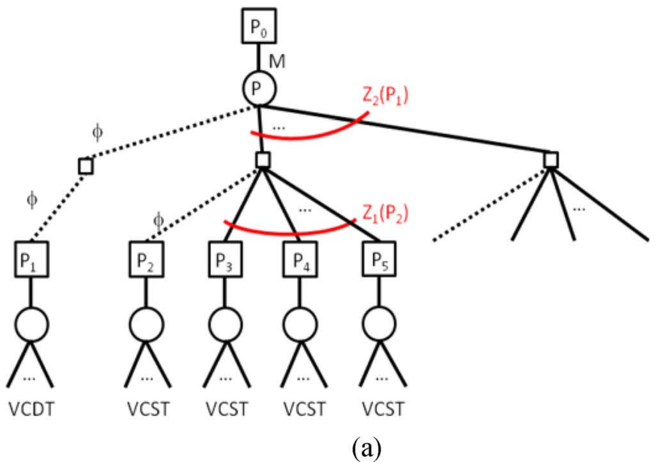


Fig. 18. Proof search tree of (a) NCTU6-Verifier and (b) the verifier of one higher order.

When our *Connect6* program NCTU6 mentioned above cannot find Λ^2-strategies (VCSTs), NCTU6 then chooses some promising moves including nonthreat moves using heuristic evaluations. The details of heuristic evaluations are beyond the scope of this paper and therefore omitted.

Since NCTU6 may not be able to find winning moves all the time, experts are allowed to help find winning moves. (As [1], [2], and [23], expert knowledge was utilized to help solve *Go-Moku* and *Renju*.) Hence, the above programs, such as NCTU6 and NCTU6-Verifier, were integrated into a *Connect6* editor named Connect6Lib [8], modified from Renlib [16], in order to accommodate hints from human experts. In the integrated system [25], [32], the users (human experts) are allowed to suggest some attacker moves directly or let NCTU6 suggest possibly good moves in a designated position. Then, for suggested moves, users invoke NCTU6-Verifier to verify and report all the defensive moves (most are Λ^3-moves). Then, users repeat the above for the subsequent moves, until a Λ^3-strategy is found.

Second, for Λ^4-strategies, the integrated system (on top of the editor Connect6Lib) needs to maintain a global verifier and modify the search by incrementing the order by one as shown in Fig. 18(b).

B. Illustration of Solving Positions

In this section, we illustrate the proof search method in Section IV by solving the two positions in Figs. 3(a) and 4(a). First, consider the one in Fig. 3(a). The position is solved by simply running NCTU6-Verifier. In the proof search tree shown in Fig. 19, P indicates the position at 7 in Fig. 3(a); P_0, the position at 6; P_1, the position after a null move; P_2, the position after the seminull move 8 in Fig. 3(c); and P_{21}, the position after another seminull move at 10 in Fig. 3(c). As can be seen, the attacker wins in a Λ^3-strategy.

Second, consider the position in Fig. 4(a), which is much more complicated than the previous one. This position is solved via the integrated system supporting Λ^4-trees, as described in Section V-A. In the proof search tree shown in Fig. 20, P indicates this position, P_1 does the position after a null move, and

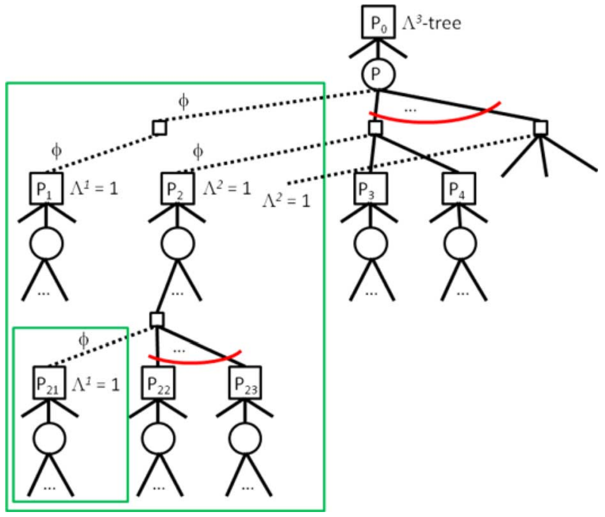


Fig. 19. Proof search tree for the position in Fig. 3(a).

P_2 does the position after a seminull move at 7 in Fig. 4(c). Initially, let NCTU6-Verifier of one higher order run in P. Since VCST-Solver is able to find the winning move for P_1, the defender (Black) should place at least one stone in zone $Z_2(P_1)$. Consider one square s in $Z_2(P_1)$, say square 7 in Fig. 4(c). For the seminull move at 7, choose move 8 and then use NCTU6-Verifier (without raising one order) to derive that the attacker wins at 8. Thus, move 8 is a Λ^3-move. By verifying all null and seminull moves in P, we show that move 6 in Fig. 4(a) is a Λ^4-move (from Definition 1).

Furthermore, the attacker is shown to win at 6 in a Λ^4-strategy as follows. In our experiment, the attacker wins for all defensive (nonnull) moves by finding Λ^3-strategies. For example, for move 7 in Fig. 21, NCTU6-Verifier is recursively employed to find a Λ^3-strategy, where moves 8–12 are shown to be Λ^3-moves.

In the proof search tree shown in Fig. 20, we found three seminull moves that are Λ^3-moves with value 1 [like P_2 which is also 7 in Fig. 4(c)], and 569 defender Λ^3-moves in total. Move 12 in Fig. 21 is the deepest Λ^3-move. In this experiment, experts

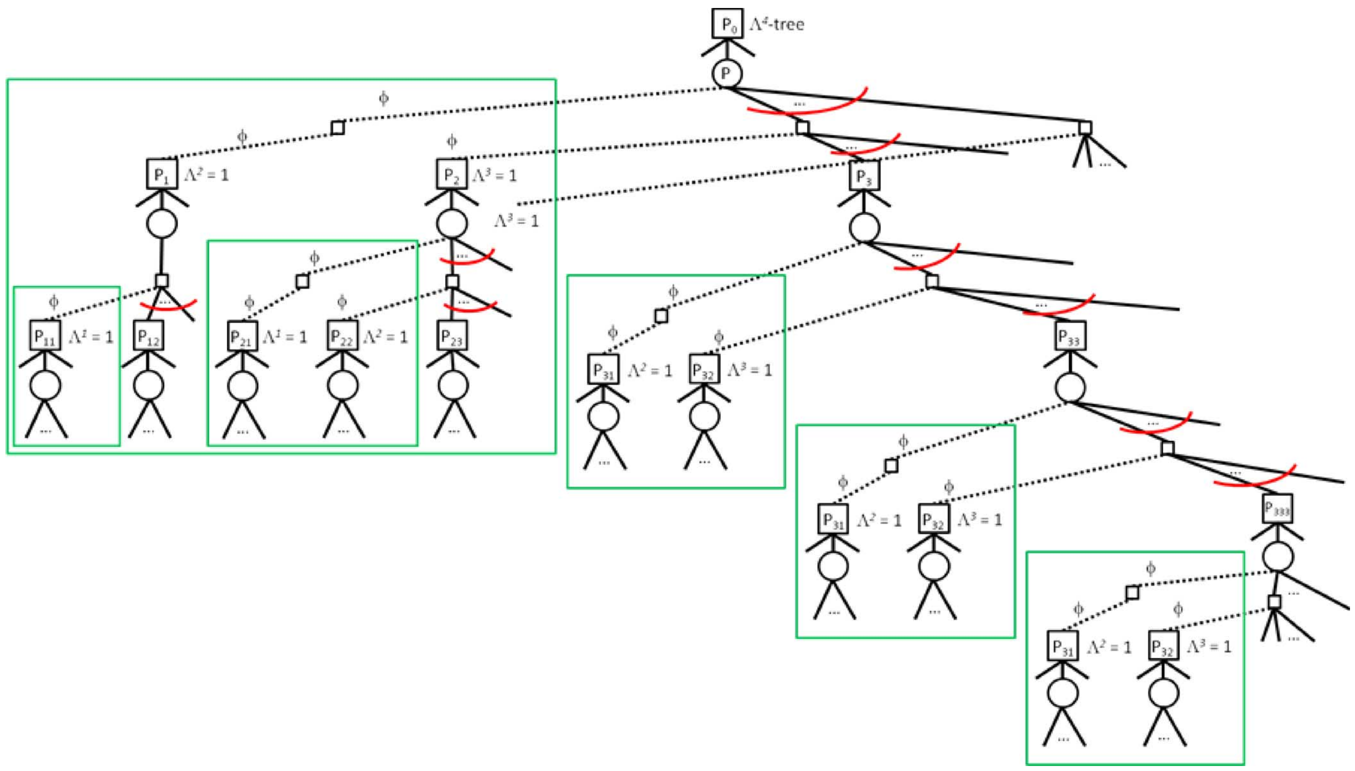


Fig. 20. Proof search tree for the position in Fig. 4(a).

TABLE I
STATISTICS OF SOLVING POSITIONS

Position	Number of nodes	Time (in seconds)
Fig. 2 (a)	65054	18.39
Fig. 3 (a)	498380	104.41
Fig. 4 (a)	210229668	44448.07

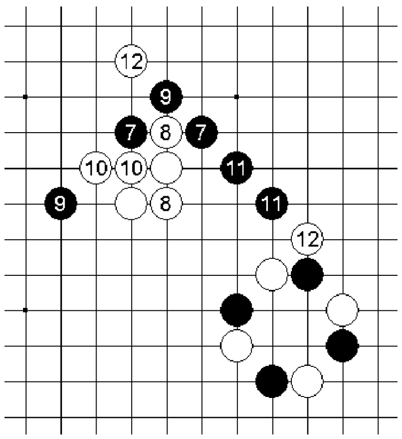


Fig. 21. Sequence of Λ^3-moves starting from 7.

helped find 26 winning nonthreat moves, including move 6 discovered by Huang [11].

Table I shows the number of nodes as well as the computation times used by our system to solve the positions in Figs. 2(a), 3(a), and 4(a) on an Intel Pentium Dual 2.00-GHz machine. The positions in Figs. 2(a) and 3(a) are solved without experts' assistance, while the position in Fig. 4(a) is solved with the help of experts, as above. All the above experiments were performed on 19×19 boards that most current *Connect6* tournaments use. We also used a simple tool to verify that the above example is still winning even in an infinite board. The details are omitted.

C. More Results

In addition to the two positions illustrated in Section V-B, we investigated more positions. Initially, we had experts use the integrated system to help us solve about ten more positions. Wu *et al.* [28] had recently automated with success the proof process by developing a new search algorithm, called job-level proof-number (JL-PN) search. Using the JL-PN search together with our RZOP search, we solved many more positions, up to 65 positions in total, with Λ^3-strategy, within a couple of months. The details of the 65 positions were listed in [27].

Among the 65 positions, 34 were not solved by the scheme, called the VCDT-for-null scheme. The scheme uses VCDTs (not VCSTs) after seminull moves in proof search trees such as the one in Fig. 2(c). If no VCDTs were found for the seminull moves as the one in Fig. 3(c), then the scheme failed to solve positions. In brief, the proof search trees in our RZOP search are as in Fig. 18(a), while those in the scheme are as in Fig. 8.

Many positions were not solved by the VCDT-for-null scheme illustrated below. For the three openings in Fig. 22(c), (d), and (f), the winning moves are live threes at 3. For the seminull moves that use the only stones to block Black's live threes, Black has no more double-threat moves to make. That is, Black cannot win by VCDTs. However, Black actually wins by VCSTs for these seminull moves. Hence, it is important that the proposed RZOP can solve them correctly.

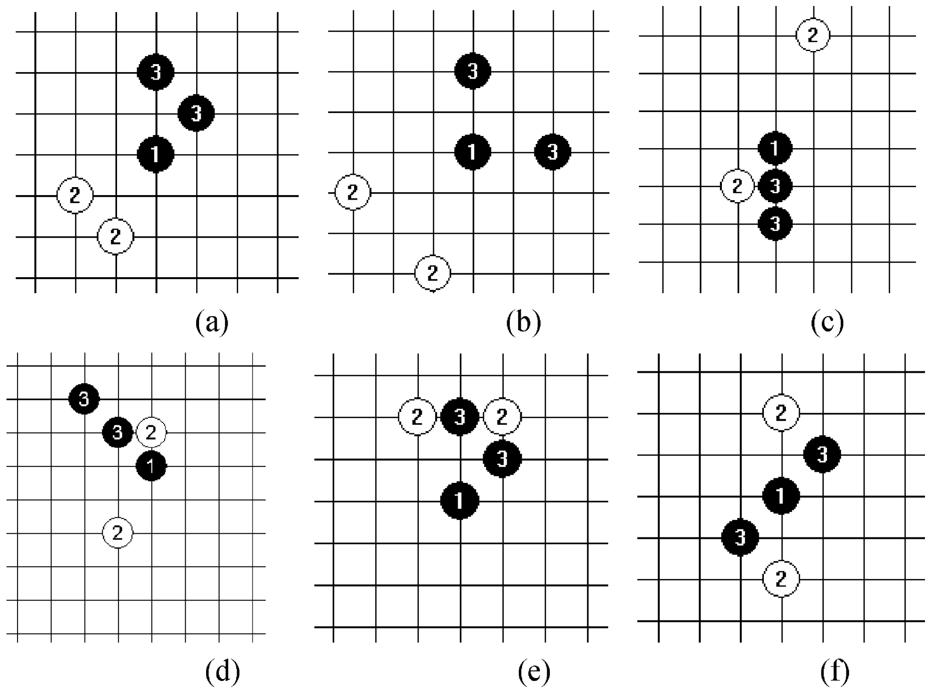


Fig. 22. Six three-move openings in which Black wins at 3.

The 65 positions include 12 three-move openings, among which ten cannot be solved by the VCDT-for-null scheme. Six of the ten openings are shown in Fig. 22. In particular, the fifth one, Mickey Mouse opening, used to be one of the popular openings before we solved it. Mickey Mouse opening was so named in [20], since White 2 and Black 1 together look like the face of Mickey Mouse. The sixth one, also called straight opening, is another difficult one.

Now, the question is whether there exist more cases requiring Λ^4-strategies like the one in Fig. 4. Since the one in Fig. 4 is the only one that we found so far, it is still an open problem to find some more.

VI. Conclusion

This paper investigates a new threat-based proof search for *Connect* games. The contribution of this paper is mainly the new search method, named RZOP search that uses relevance zones to help solve many positions in *Connect6* as well as *Connect* games. In theory, this method can be applied to *Connect* games with infinite boards. Practically, this paper demonstrates the method by solving two typical winning positions in Figs. 3(a) and 4(a) on 19 × 19 boards, as well as many *Connect6* positions and openings in Section V. In addition, the method can also be easily incorporated into *Connect6* program, such as NCTU6.

This paper also leaves some open problems.

1) Investigate more winning positions in *Connect6* that require Λ^4-strategies, such as the one in Fig. 4(a).
2) Investigate whether there exists a Λ^5-strategy in *Connect6*.
3) Optimize the proof search tree by pruning more branches efficiently [29].
4) Apply the new method (in the Appendix) to solving some real positions in general *Connect* games.
5) Investigate whether dual lambda search [19] is useful for *Connect6* or *Connect* games.

Using the JL-PN search together with our RZOP search, we successfully solved up to 65 positions with Λ^3-strategy. The 65 positions include 12 three-move openings; in particular, Mickey Mouse opening, which used to be one of the popular openings before we solved it. One might ask whether or when *Connect6* on 19 × 19 boards will be solved. So far, we still could not solve tens of the common openings, many of which experts believed to be well balanced for both players. Hence, the answer to this question is still unknown.

Appendix
Proof Search for Connect Games

In this Appendix, the verifier $V_{C6}(P, S)$ is generalized to general *Connect* games, *Connect*(m, n, k, p, q), while maintaining Property RZV. The generalized verifier is denoted by $V_{CK}(P, S)$. In the case that P is an endgame position or is in the attacker's turn (described in Sections IV-B and IV-C, respectively), the verifier $V_{CK}(P, S)$ is the same as $V_{C6}(P, S)$. So, the rest of this Appendix describes the verifier only in the case that P is in the defender's turn. Furthermore, the position P (in the defender's turn) can be classified into the following two: 1) the number of attacker threats t in P is at least $p + 1$; and 2) the number t is at most p. In the first case, the attacker wins already. Therefore, the verifier returns 1 and constructs relevance zones in the following operation.

Tp1-1) Construct relevance zones by following both operations T3-1 and T3-2, except that the terms "$i+2$" are replaced by "$i + p$."

Similar to Lemma 7, Lemma 11 shows that the verifier also satisfies Property RZV in this case.

Lemma 11: Assume that the defender is to move and the number of the attacker threats is at least $p+1$ in P. The verifier described above satisfies Property RZV.

Proof: The proof is similar to that of Lemma 7 and therefore omitted. ■

In the second case that the number of attacker threats t is at most p, the verifier performs the following operations.

Tp-1) For each of critical defenses M_D (both normal and relaxed), perform the following.
  a) Return 0 if the subverifier $V_{\text{sub}}(M_D, P, S)$ returns 0. Note that the subverifier is described below.
  b) Let $\Psi(P) = \Psi(P) \cup \Psi'(P_D)$.

Tp-2) Continue to construct relevance zones in operation Tp1-1, and return 1.

In operation Tp-1a, a subverifier $V_{\text{sub}}(M_D, P, S)$ is used to verify whether the attacker wins for all defender moves M'_D dominated by M_D in P, where M'_D has p squares (but M_D may have less than p squares). By dominate, we mean that all squares in M_D must also be in M'_D, but not *vice versa*. For the subverifier $V_{\text{sub}}(M_D, P, S)$, the constructed zone is denoted by $\Psi'(P_D) = \langle Z'_1(P_D), Z'_2(P_D), \ldots, Z'_r(P_D)\rangle$, where $P_D = P \oplus M_D$. In addition, the subverifier satisfies the following property (proved in Lemma 12).

Property RZS: If $V_{\text{sub}}(M_D, P, S)$ returns 1, the following condition holds. For all defender moves M'_D dominated by M_D, there exists some Ψ'_D such that $\Psi'_D \subseteq \Psi'(P_D)$ and Ψ'_D is in $RZ(P \oplus M'_D)$.

The subverifier $V_{\text{sub}}(M_D, P, S)$ performs the following operations.

Par-1) Assume that M_D has exactly $p - u$ defender stones, where u is the number of null stones in M_D and $0 \leq u \leq p$. In the case that $u > 0$, move M_D is a null or a seminull move.

Par-2) Return 0 if $V_{CK}(P_D, S)$ returns 0, where $P_D = P \oplus M_D$.

Par-3) Let $\Psi'(P_D) = \Psi(P_D) \ll u$.

Par-4) Return 1 if $u = 0$, i.e., the move is not a null or a seminull move.

Par-5) For each of unoccupied square $s \in \neg_{P_D}(Z_u(P_D))$, perform the following.
  a) Let the defender move $M_{D,s}$ be $M_D + \sigma_D(s)$.
  b) Return 0 if $V_{\text{sub}}(M_{D,s}, P, S)$ returns 0.
  c) Let $\Psi'(P_D) = \Psi'(P_D) \cup \Psi'(P_{D,s})$, where $P_{D,s} = P \oplus M_{D,s}$.

Par-6) Return 1.

Lemma 12 shows that the subverifier satisfies Property RZS, if all the recursive V_{sub} in Par-5b satisfy Property RZS and the verifier V_{CK} in Par-2 satisfies Property RZV.

Lemma 12: For a subverifier $V_{\text{sub}}(M_D, P, S)$ as described above, it satisfies Property RZS by assuming that all the recursive V_{sub} in Par-5b satisfy Property RZS and that the verifier V_{CK} in Par-2 satisfies Property RZV.

Proof: Assume that $V_{\text{sub}}(M_D, P, S)$ returns 1. Consider all defender moves M'_D (including p stones) that are dominated by M_D. Namely, let $M'_D = M_D + \sigma_D(\varphi)$, where φ has u additional unoccupied squares. For this lemma, it suffices to prove that there exists some Ψ'_D such that $\Psi'_D \subseteq \Psi'(P_D)$ and Ψ'_D is in $RZ(P \oplus M'_D)$. All of these defender moves M'_D are classified into the following cases.

1) All defender moves M'_D in which all additional squares s in φ are in $\neg_{P_D}(Z_u(P_D))$. The proof for this case is similar to that for case 1 in Lemma 10 as follows. Since this subverifier returns 1, the verifier $V_{CK}(P_D, S)$ in Par-2 returns 1. Since the verifier V_{CK} returns 1 and also satisfies Property RZV (from this lemma), $\Psi(P_D)$ is in $RZ(P_D)$. Since all additional $s \in \neg_{P_D}(Z_u(P_D))$, we obtain from Lemma 3 that $\Psi(P_D) \ll u$ is in $(P_D + \sigma_D(\varphi))$. Since
$$\begin{aligned} P_D + \sigma_D(\varphi) &= (P \oplus M_D) + \sigma_D(\varphi) \\ &= P \oplus (M_D + \sigma_D(\varphi)) \\ &= P \oplus M'_D, \Psi(P_D) \ll u \end{aligned}$$
is also in $RZ(P \oplus M'_D)$. In addition, since $\Psi(P_D) \ll u \subseteq \Psi'(P_D)$ from Par-3 in V_{sub}, $\Psi(P_D) \ll u$ is the Ψ'_D.

2) All defender moves M'_D where some additional square s in φ is in $Z_u(P_D)$. Since this subverifier returns 1, the recursive $V_{\text{sub}}(M_{D,s}, P, S)$ at Par-5b returns 1 as well, and therefore, satisfies Property RZS. From Property RZS, there exists some Ψ such that $\Psi \subseteq \Psi'(P_{D,s})$ and Ψ is in $(P \oplus M'_D)$. Since $\Psi'(P_{D,s}) \subseteq \Psi'(P_D)$ from operation Par-5c, we obtain $\Psi \subseteq \Psi'(P_D)$. Thus, Ψ is the Ψ'_D. ■

From Lemma 12, we derive Lemma 13 as follows.

Lemma 13: Assume that the defender is to move and the number of attacker threats is at most p in P. The verifier described above satisfies Property RZV by assuming that all the recursive subverifiers in operation Tp-1a satisfy Property RZS.

Proof: Assume that this verifier returns 1. For this lemma, it suffices to prove that the constructed $\Psi(P)$ is in $RZ(P)$. Since the verifier returns 1, all the recursive subverifiers in operation Tp-1a returns 1 as well. Assume that these subverifiers satisfy Property RZS. For proving $\Psi(P) \in RZ(P)$, it suffices to prove from Lemma 6 the following: for all defender moves M_D, there exists some Ψ_D such that Ψ_D is in $RZ(P \oplus M_D)$ and $\Psi_D \subseteq \Psi(P)$. All defender moves M_D are classified into the following two cases.

1) All defender moves M_D that block all the threats. There must exist some critical defense M'_D (either normal or relaxed) dominating M_D. Since $V_{\text{sub}}(M'_D, P, S)$ returns 1 and satisfies Property RZS from the above, there exists some Ψ_D from the property such that $\Psi_D \subseteq \Psi'(P \oplus M'_D)$ and Ψ_D is in $RZ(P \oplus M'_D)$.

2) All defender moves M_D that leave some threat unblocked. The attacker wins by connecting up to p on some unblocked threat segment, like S_{3T}. From the proof in Lemma 11, we obtain that there exists some Ψ_D such that $\Psi_D \subseteq \Psi'(P)$ and Ψ_D is in $RZ(P_D)$. ■

Theorem 3 concludes that the verifier $V_{CK}(P, S)$ in all cases satisfies Property RZV. Therefore, if $V_{CK}(P, S)$ returns 1, the constructed $\Psi(P)$ is in $RZ(P)$, and the attacker wins in P from Corollary 2. It can also be observed that the operations in Section IV-D are special cases of the operations described in this Appendix.

Theorem 3: The verifier $V_{CK}(P, S)$ satisfies Property RZV in all cases.

Proof: By induction, the verifier $V_{CK}(P, S)$ satisfies Property RZV in all cases from the above lemmas. ■

ACKNOWLEDGMENT

The authors would like to thank anonymous reviewers for their valuable comments.

References

[1] L. V. Allis, "Searching for solutions in games and artificial intelligence," Ph.D. dissertation, Dept. Comput. Sci., Univ. Limburg, Maastricht, The Netherlands, 1994.

[2] L. V. Allis, H. J. van den Herik, and M. P. H. Huntjens, "Go-Moku solved by new search techniques," *Comput. Intell.*, vol. 12, pp. 7–23, 1996.

[3] L. V. Allis, M. van der Meulen, and H. J. van den Herik, "Proof-number search," *Artif. Intell.*, vol. 66, no. 1, pp. 91–124, 1994.

[4] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for Your Mathematical Plays*, 2nd ed. Natick, MA: A K Peters. Ltd., 2003, vol. 3.

[5] A. de Bruin, W. Pijls, and A. Plaat, "Solution trees as a basis for game-tree search," *Int. Comput. Chess Assoc. J.*, vol. 17, no. 4, pp. 207–219, Dec. 1994.

[6] T. Cazenave, "Abstract proof search," in *Computers and Games*, ser. Lecture Notes in Computer Science, T. A. Marsland and I. Frank, Eds. Berlin, Germany: Springer-Verlag, 2001, vol. 2063, pp. 39–54.

[7] T. Cazenave, "A generalized threats search algorithm," in *Computers and Games*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 2003, vol. 2883, pp. 75–87.

[8] C.-P. Chen, I.-C. Wu, and Y.-C. Chan, "ConnectLib – A Connect6 Editor," 2009 [Online]. Available: http://www.connect6.org/Connect6Lib_Manual.htm

[9] Chinese Association for Artificial Intelligence, "Chinese Computer Games Contest," (in Chinese) [Online]. Available: http://www.caai.cn/

[10] H. J. van den Herik, J. W. H. M. Uiterwijk, and J. V. Rijswijck, "Games solved: Now and in the future," *Artif. Intell.*, vol. 134, no. 1–2, pp. 277–311, 2002.

[11] Y.-C. Huang, *Private Communication.* 2008.

[12] P.-H. Lin and I.-C. Wu, "NCTU6 wins man-machine Connect6 championship 2009," *Int. Comput. Games Assoc. J.*, vol. 32, no. 4, pp. 230–232, 2009.

[13] T. W. Lee, "One of early Tsumegos for Connect6," 2005 [Online]. Available: http://www.connect6.org/web/index.php?option=com_tsumego&task=loadTsumegoHistoryList&class_id=32

[14] Littlegolem, "Online Connect6 Games," 2006 [Online]. Available: http://www.littlegolem.net/

[15] Renju International Federation, "The International Rules of Renju," 1998 [Online]. Available: http://www.renju.net/study/rifrules.php

[16] Renlib, Renju—A Ranju Editor [Online]. Available: http://www.renju.se/renlib/

[17] W. Pijls and A. de Bruin, "Game tree algorithms and solution trees," in *Computers and Games*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 1999, vol. 1558, pp. 195–204.

[18] G. Sakata and W. Ikawa, *Five-in-a-Row*. Tokyo, Japan: The Ishi Press, 1981.

[19] S. Soeda, T. Kaneko, and T. Tanaka, "Dual lambda search and its application to Shogi endgames," in *Advances in Computer Games*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 2006, vol. 4250, pp. 126–139.

[20] Taiwan Connect6 Association, Connect6 Homepage, [Online]. Available: http://www.connect6.org/

[21] ThinkNewIdea Inc., CYC Game, (in Chinese) 2005 [Online]. Available: http://cycgame.com/

[22] T. Thomsen, "Lambda-search in game trees – with application to Go," *Int. Comput. Games Assoc. J.*, vol. 23, no. 4, pp. 203–217, 2000.

[23] J. Wagner and I. Virag, "Solving Renju," *Int. Comput. Games Assoc. J.*, vol. 24, no. 1, pp. 30–34, 2001.

[24] I.-C. Wu, "Proposal for a New Computer Olympiad Game—Connect6," 2005 [Online]. Available: http://ticc.uvt.nl/icga/news/Olympiad/Olympiad2006/connect6.pdf, or http://www.connect6.org/articles/RZOP/connect6.pdf

[25] I.-C. Wu, C.-P. Chen, P.-H. Lin, K.-C. Huang, L.-P. Chen, D.-J. Sun, Y.-C. Chan, and H.-Y. Tsou, "A Volunteer-computing-based grid environment for Connect6 applications," in *IEEE Int. Conf. Comput. Sci. Eng.*, Vancouver, BC, Canada, Aug. 29–31, 2009, pp. 110–117.

[26] I.-C. Wu and P.-H. Lin, "NCTU6-lite wins Connect6 tournament," *Int. Comput. Games Assoc. J.*, vol. 31, no. 4, pp. 240–243, 2008.

[27] I.-C. Wu and P.-H. Lin, "Benchmark for RZOP search," [Online]. Available: http://www.connect6.org/articles/RZOP/

[28] I.-C. Wu, H.-H. Lin, P.-H. Lin, D.-J. Sun, Y.-C. Chan, and B.-T. Chen, "Job-level proof-number search for Connect6," presented at the Int. Conf. Comput. Games Kanazawa, Japan, 2010.

[29] I.-C. Wu, H.-H. Lin, and P.-H. Lin, "A more efficient proof search for Connect6," 2010, in preparation.

[30] I.-C. Wu, D.-Y. Huang, and H.-C. Chang, "Connect6," *Int. Comput. Games Assoc. J.*, vol. 28, no. 4, pp. 234–242, 2006.

[31] I.-C. Wu and D.-Y. Huang, "A new family of k-in-a-row games," in *Advances in Computer Games*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 2006, vol. 4250, pp. 180–194.

[32] I.-C. Wu, D.-J. Sun, H.-H. Lin, P.-H. Lin, C.-P. Chen, L.-P. Chen, and H.-Y. Tsou, "A volunteer computing system for Connect6 Applications", National Chiao Tung Univ., Hsinchu, Taiwan, Tech. Rep., 2010.

[33] I.-C. Wu and S.-J. Yen, "NCTU6 wins Connect6 tournament," *Int. Comput. Games Assoc. J.*, vol. 29, no. 3, pp. 157–158, Sep. 2006.

I-Chen Wu (M'10) received the B.S. degree in electronic engineering and the M.S. degree in computer science from the National Taiwan University (NTU), Taipei, Taiwan, in 1982 and 1984, respectively, and the Ph.D. degree in computer science from Carnegie Mellon University, Pittsburgh, PA, in 1993.

Currently, he is with the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan. His research interests include artificial intelligence, Internet gaming, volunteer computing, and cloud computing.

Dr. Wu introduced the new game *Connect6*, a kind of six-in-a-row game, and presented this game at the 2005 11th Advances in Computer Games Conference (ACG'11). Since then, *Connect6* has become a tournament item at the Computer Olympiad. He led a team developing a *Connect6* program, named NCTU6. The program won the gold twice at the Computer Olympiad in both 2006 and 2008.

Ping-Hung Lin is currently working towards the Ph.D. degree at the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan.

He is the Current Chief Designer of the *Connect6* program NCTU6 that won the gold twice at the Computer Olympiad in both 2006 and 2008. His research interests include artificial intelligence and grid and cloud computing.

Theoretical Computer Science 412 (2011) 4558–4569

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Drawn k-in-a-row games

Sheng-Hao Chiang[a], I-Chen Wu[b,*], Ping-Hung Lin[b]

[a] National Experimental High School at Hsinchu Science Park, Hsinchu, Taiwan

[b] Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan

ARTICLE INFO

Article history:
Received 12 July 2009
Received in revised form 20 January 2011
Accepted 21 April 2011
Communicated by G. Ausiello

Keywords:
k-in-a-row games
*Connect*6
Hypergraphs

ABSTRACT

Wu and Huang (2005) [12] and Wu et al. (2006) [13] presented a generalized family of k-in-a-row games, called *Connect*(m, n, k, p, q). Two players, *Black* and *White*, alternately place p stones on an $m \times n$ board in each turn. Black plays first, and places q stones initially. The player who first gets k consecutive stones of his/her own horizontally, vertically, or diagonally wins. Both tie the game when the board is filled up with neither player winning. A *Connect*(m, n, k, p, q) game is drawn if neither has any winning strategy. Given p, this paper derives the value $k_{draw}(p)$, such that *Connect*(m, n, k, p, q) games are drawn for all $k \geq k_{draw}(p)$, $m \geq 1$, $n \geq 1$, $0 \leq q \leq p$, as follows. (1) $k_{draw}(p) = 11$. (2) For all $p \geq 3$, $k_{draw}(p) = 3p + 3d - 1$, where d is a logarithmic function of p. So, the ratio $k_{draw}(p)/p$ is approximately 3 for sufficiently large p. The first result was derived with the help of a program. To our knowledge, our $k_{draw}(p)$ values are currently the smallest for all $2 \leq p < 1000$.

1. Introduction

A generalized family of k-in-a-row games, called *Connect*(m, n, k, p, q), [12,13], was introduced and presented by Wu et al. Two players, *Black* and *White*, alternately place p stones on empty *squares*[1] of an $m \times n$ board in each turn. Black plays first, and places q stones initially. The player who first gets k consecutive stones of his/her own horizontally, vertically, or diagonally wins. Both players tie the game when the board is filled up with neither player winning. For example, *Tic-tac-toe* is *Connect*$(3, 3, 3, 1, 1)$, *Go-Moku* in the free style (a traditional five-in-a-row game) is *Connect*$(15, 15, 5, 1, 1)$, and *Connect*6 [13], played on the traditional Go board, is *Connect*$(19, 19, 6, 2, 1)$.

In the past, many researchers have been engaged in solving *Connect*(m, n, k, p, q) games. One player, either Black or White, is said to *win* a game, if he/she has a winning strategy such that he/she wins for all the subsequent moves. Allis et al. [1,2] solved *Go-Moku* with Black winning. Herik et al. [9] and Wu et al. [12,13] also mentioned several k-in-a-row games with Black winning.

A game is said to be *drawn* if neither player has any winning strategy. For simplicity of discussion in this paper, *Connect*(k, p) refers to the collection of *Connect*(m, n, k, p, q) games for all $m \geq 1$, $n \geq 1$, $0 \leq q \leq p$. *Connect*(k, p) is said to be *drawn* if all *Connect*(m, n, k, p, q) games in *Connect*(k, p) are drawn. Given p, this paper derives the value $k_{draw}(p)$, such that *Connect*$(k_{draw}(p), p)$ games are drawn. Since drawn *Connect*(k, p) games also imply drawn *Connect*$(k+1, p)$, the value $k_{draw}(p)$ should be as small as possible.

In the past, Zetters [15] derived that *Connect*$(8, 1)$ is drawn. Pluhar [11] derived tight bounds $k_{draw}(p) = p + \Omega(\log_2 p)$ for all $p \geq 1000$ (see Theorem 1 in [11]). However, the requirement that $p \geq 1000$ is unrealistic in real games. Thus, it is

* Corresponding author. Tel.: +886 3 5731855; fax: +886 3 5733777.
E-mail addresses: jiang555@ms37.hinet.net (S.-H. Chiang), icwu@csie.nctu.edu.tw, icwu@cs.nctu.edu.tw (I.-C. Wu), bhlin@csie.nctu.edu.tw (P.-H. Lin).

[1] Practically, stones are placed on empty intersections of Renju or Go boards. In this paper, when we say squares, we mean intersections.

0304-3975/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2011.04.033

important to obtain tight bounds when $p < 1000$. Hsieh and Tsai [10] have recently derived that $k_{draw}(p) = 4p + 7$ for all positive p. The ratio $R = k_{draw}(p)/p$ is approximately 4 for sufficiently large p.

In this paper, Theorem 1 (below) shows that $k_{draw}(2) = 11$, while the result in [10] is 15. Theorem 2 derives a general bound $k_{draw}(p) = 3p + 3d - 1$ for all $p \geq 1$, where d is a logarithmic function of p, namely $P(d-1) < p \leq P(d)$ and $P(d) = 2^d - d - 2$. When compared with [10], our $k_{draw}(p)$ values are smaller for all positive p, but they are the same for $k_{draw}(4)$. The ratio $R = k_{draw}(p)/p = 3 + (3d-1)/p$ is approximately 3 for sufficiently large p. Section 2 modifies the games slightly into those in a different version, named *Maker–Breaker*. Both Sections 3 and 4 will use this version to prove Theorems 1 and 2, respectively. When compared with a preliminary version [6], this paper derives a tighter bound for $k_{draw}(3)$ and a more general result, specifically as follows. For all the drawn games, *Connect*$(\infty, \infty, k, p, p)$, derived in [6], this paper also shows that all games in *Connect*(k, p) are also drawn, based on the Maker–Breaker argument.

Theorem 1. *Connect*$(11, 2)$ *is drawn.* □

Theorem 2. *Consider all* $p \geq 1$. *Let* d *be an integer and* $P(d-1) < p \leq P(d)$, *where* $P(d) = 2^d - d - 2$. *Then, Connect*$(3p+3d-1, p)$ *games are drawn.* □

2. Maker–Breaker version

According to the strategy-stealing argument raised by Nash (see [5]), White has no winning strategy in *Connect*(m, n, k, p, p), that is, when $q = p$. Therefore, for *Connect*(m, n, k, p, p), either Black wins or White ties. For simplicity of combinatorial analysis, many researchers [3,7,11] followed an asymmetric version of rules, called *Maker–Breaker*, where White does not win in all cases (e.g., even if White connects up to k consecutive stones). So, all White can do is to break, that is, to prevent Black from winning (connecting up to k consecutive stones). In contrast to Maker–Breaker, the version with the original rules is called *Maker–Maker*. Obviously, if White has a strategy to tie a *Connect* game in the Maker–Breaker version, White can tie the game in the original version (Maker–Maker) by simply following the same strategy. For simplicity of discussion in this paper, let *MBConnect*(k, p) denote the game *Connect*$(\infty, \infty, k, p, p)$ in the Maker–Breaker version. Corollary 1 shows an important property for *MBConnect*(k, p).

Corollary 1. *Assume that MBConnect*(k, p) *is drawn. Then, Connect*(k, p) *is drawn. That is, for all* $m \geq 1$, $n \geq 1$, $0 \leq q \leq p$, *Connect*(m, n, k, p, q) *games are drawn.* □

The reasons why Corollary 1 is satisfied are as follows.

1. According to the strategy-stealing argument (also mentioned in [13]), if Black has a winning strategy in *Connect*(m, n, k, p, q), then Black simply follows the strategy to win in *Connect*$(m, n, k, p, q+1)$. On the other hand, if Black has no winning strategy in *Connect*$(m, n, k, p, q+1)$, then Black has no winning strategy in *Connect*(m, n, k, p, q) either. Similarly, if White has no winning strategy in *Connect*(m, n, k, p, q), White has no winning strategy in *Connect*$(m, n, k, p, q+1)$.
   Assume that *Connect*(m, n, k, p, p) is drawn. Then, Black has no winning strategy in *Connect*(m, n, k, p, p). From the previous paragraph, we derive that, for all $0 \leq q \leq p$, Black has no winning strategy in *Connect*(m, n, k, p, q). On the other hand, since White in *Connect*$(m, n, k, p, 0)$ is equivalent to Black in *Connect*(m, n, k, p, p), White does not win in *Connect*$(m, n, k, p, 0)$ either. From the previous paragraph, we derive that, for all $0 \leq q \leq p$, White has no winning strategy in *Connect*(m, n, k, p, q). Thus, since neither has any winning strategy, *Connect*(m, n, k, p, q) games are drawn for all $0 \leq q \leq p$.
2. If Black has a winning strategy in *Connect*(m, n, k, p, q) in the Maker–Breaker version, then Black simply follows the strategy to win in *Connect*$(m+1, n, k, p, q)$, *Connect*$(m, n+1, k, p, q)$, or even *Connect*$(\infty, \infty, k, p, q)$ in the Maker–Breaker version. On the other hand, if Black has no winning strategy in *Connect*$(\infty, \infty, k, p, q)$ in the Maker–Breaker version, then Black does not win in *Connect*(m, n, k, p, q) in the Maker–Breaker version for all $m \geq 1$, $n \geq 1$, either.

Assume that *MBConnect*(k, p) is drawn. For the second reason, for all $m \geq 1$, $n \geq 1$, *Connect*(m, n, k, p, p) games are drawn in the Maker–Breaker version, as well as in the original version. For the first reason, *Connect*(m, n, k, p, q) games are drawn for all $m \geq 1$, $n \geq 1$, $0 \leq q \leq p$. Thus, *Connect*(k, p) is drawn and Corollary 1 is satisfied.

On the basis of Corollary 1, Sections 3 and 4 both simply derive drawn *MBConnect*(k, p) from Theorems 1 and 2, respectively, instead of deriving drawn *Connect*(k, p) directly. Moreover, to prove both theorems, we also need to define new Maker–Breaker games for smaller boards B, named *MBBoard*(B, p), in Definition 1.

Definition 1. *MBBoard*(B, p) is a Maker–Breaker game defined as follows.

1. The *game board* B is composed of a set of squares and a set of *lines*, each of which covers a subset of squares. For simplicity of discussion, all lines are (vertically, horizontally, or diagonally) straight and solid in all figures in the rest of this paper, as illustrated in Fig. 1.
2. In Move $2i-1$, where $i \geq 1$, Black is allowed to place p' stones on the game board B, where $p' \leq p$. In Move $2i$, White places p' or fewer stones.
3. Black wins when *occupying some line*. Note that Black is said to occupy a line if all the squares covered by the line are occupied by black stones. □

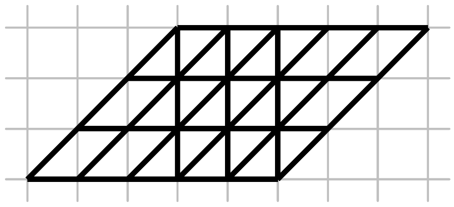


Fig. 1. The game board B_2.

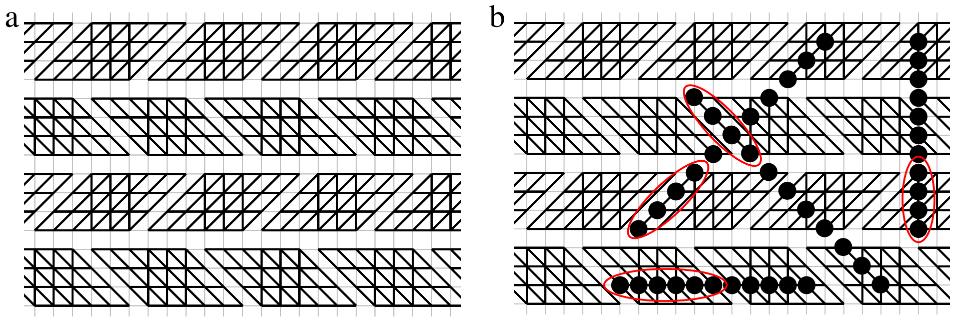


Fig. 2. (a) Partitioning the infinite board into disjoint B_2. (b) Covering one complete solid line for each segment of 11 consecutive squares.

The game *MBBoard*(B, p) is said to be a *drawn* game if Black has no winning strategy, that is, White has some strategy to prevent Black from winning in all cases.

In the above game, the game board B can be viewed as a kind of *hypergraph* G [4,8]. All squares in B are vertices in G, while all (solid) lines in B are *edges*, or so-called *hyperedges* in G, covering a set of vertices. For example, the board in Fig. 1 includes 6×4 squares with 4 horizontal, 3 vertical, and 6 diagonal lines (from the lower left to the upper right). The corresponding hypergraph includes 24 vertices and 13 (i.e., $4 + 3 + 6$) edges, accordingly. In the rest of this paper, we still use the terms game boards, lines, and squares, instead of graphs, edges, and vertices.

3. Proof of Theorem 1

The infinite board is partitioned into an infinite number of disjoint B_2 (without overlap and vacancy) as shown in Fig. 2(a), where B_2 is the game board shown in Fig. 1. From Lemma 1 (below), since *MBBoard*$(B_2, 2)$ is drawn, White has some strategy S such that none of the solid lines are occupied by Black. Let White follow S to play inside each B_2. Observed from Fig. 2(b), all segments of 11 consecutive squares vertically, horizontally, and diagonally must cover entirely one solid line among these B_2. Since none of these solid lines are occupied by Black from Lemma 1, none of the segments contain all 11 black stones. Thus, *MBConnect*$(11, 2)$ is drawn. From Corollary 1, *Connect*$(11, 2)$ is drawn. □

Lemma 1. *MBBoard*$(B_2, 2)$ *is drawn.*

Proof. A program was written to verify that none of the solid lines in B_2 are occupied by Black. The program is briefly described in Section 3.2. An intuition is given in Section 3.1. □

3.1. Intuition for Lemma 1

This subsection gives an intuition for the correctness of Lemma 1. Move 1 (by Black) is classified into the following cases.

1. Black only places one stone in the board, as illustrated in Fig. 3(a).
2. Black places two stones.
   2.1 Both are placed on the two squares marked "1" in Fig. 3(b), called *middle squares* for this game board.
   2.2 One of the two stones is placed on either of the two middle squares.
   2.3 Neither of the two stones is placed on the two middle squares.

In Case 2.1, White replies by placing two stones, as shown in Fig. 3(b); and in all the other cases, White replies by placing one stone on one of the two middle squares. Here, only Case 1 in Fig. 3(a) and Case 2.1 in Fig. 3(b) are illustrated. Intuitively, it is hard for Black to occupy a horizontal line, since the horizontal lines contain two more squares than the vertical and diagonal lines. Therefore, let us ignore and remove the horizontal lines for simplicity of analysis.

After Move 2 (by White), Fig. 4 shows the boards with active vertical and diagonal lines only. Let an *active line* be a line that does not yet contain a white stone. Since Black is never able to cover all the squares of some inactive line (not active),

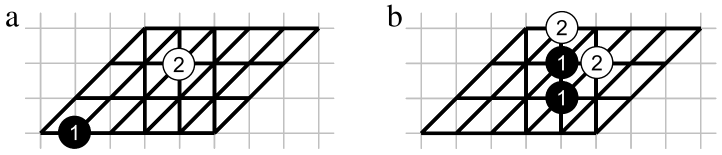


Fig. 3. The first two moves: (a) in Case 1, and (b) in Case 2.1.

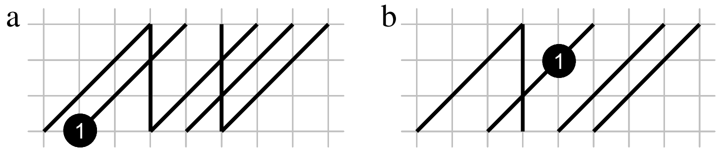


Fig. 4. The active vertical and diagonal lines after Move 2 (by White) in (a) Case 1, and (b) Case 2.1.

inactive lines are irrelevant to the results of games. Hence, the inactive lines can be removed from a board. In Fig. 4(a), the middle vertical line and the third diagonal line (from the left) become inactive and get removed after Move 2. In Fig. 4(b), the rightmost two vertical lines and the second and fourth diagonal lines (from the left) also become inactive and get removed, similarly.

A game board is called a *tree* if all the lines form no cycles in the board, as illustrated in both cases in Fig. 4. Lemma 2 (below) shows that a game is drawn if its game board is a tree which contains at most one black stone and in which each line covers at least four squares. Thus, from Lemma 2, the two games in Fig. 4 are drawn.

Lemma 2. *In a tree B_T, assume that there exists at most one black stone on B_T and that each line in B_T covers at least four squares. Then, MBBoard(B_T, 2) is drawn.*

Proof. Assume that there exists one black stone on some square s. Black cannot win in his/her next move for the following reason. Since Black can place at most two stones in a move, one line contains at most three stones (together with the one on s). Since each line covers at least four squares, Black cannot win in the next move.

Let Black place one stone on another square s' in the next move. Since the game board is a tree, we find at most one path (a sequence of lines) from s to s', and then let White place one stone on one of these lines in the path, if any. (Note that, if both $s$and s' are on the same line, White simply places a stone on that line.) Thus, B_T is broken into some trees, each of which contains at most one black stone. If Black places two stones in the next move, simply use two stones to break the game board as above. Thus, this lemma holds by induction. □

To prove Lemma 1 rigidly, we also need to consider the case that some horizontal line may be occupied by Black. Thus, the proof for this unfortunately becomes tedious. In practice, we wrote a program to prove it by searching all cases exhaustively, as briefly described in the next subsection.

3.2. Program description for Lemma 1

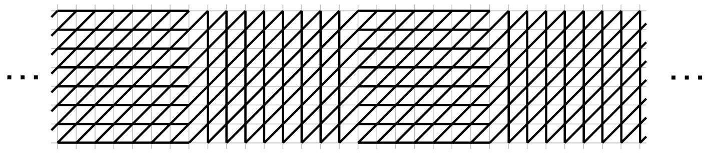
The program to prove Lemma 1 uses a recursive search routine to search the game space and to find a strategy for White to tie the game. When it is Black's turn, the search routine searches all possible Black moves exhaustively, and verifies that Black does not win in any of the moves and any of their subsequent moves recursively. For each of these Black moves, the search routine chooses a White move to play such that Black does not win subsequently. The search routine does not search deeper moves when Black occupies some line, or when it is provable that Black has no winning way subsequently, e.g., there are no more active lines.

After running the above program, it was proved that White is able to tie the game. The program searched 1291,140,480 game positions in 17,104 s on a PC with AMD Athlon$^{\text{TM}}$ 64 $\times$ 2 Dual Processor with 5200 + 2.70 GHz. However, for the purpose of publishing the search tree, a method described in [14] was employed to optimize the size of the search tree. Then, under the optimization, the program ran in 37 s and searched 844,618 game positions. The search tree was published in [14].

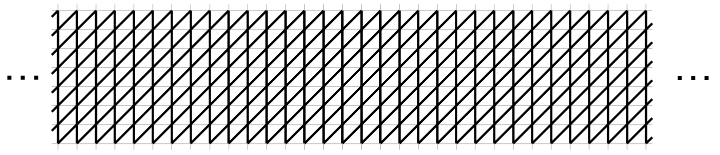
4. Proof of Theorem 2

In this proof, similar to that of Theorem 1, the infinite board is partitioned into an infinite number of disjoint game boards $B_Z(L)$ and $B_{-Z}(L)$ vertically interleaved without overlap and vacancy, as shown in Fig. 6. The game board[2] $B_Z(L)$ is shown

[2] The game board $B_N(L)$ is so named in this paper since the board shape consists of many Ns, while the game board $B_Z(L)$ is so named since the parts different from $B_N(L)$ look like Zs.



(a) $B_Z(L)$.



(b) $B_N(L)$.

Fig. 5. Two game boards: (a) $B_Z(L)$ and (b) $B_N(L)$.

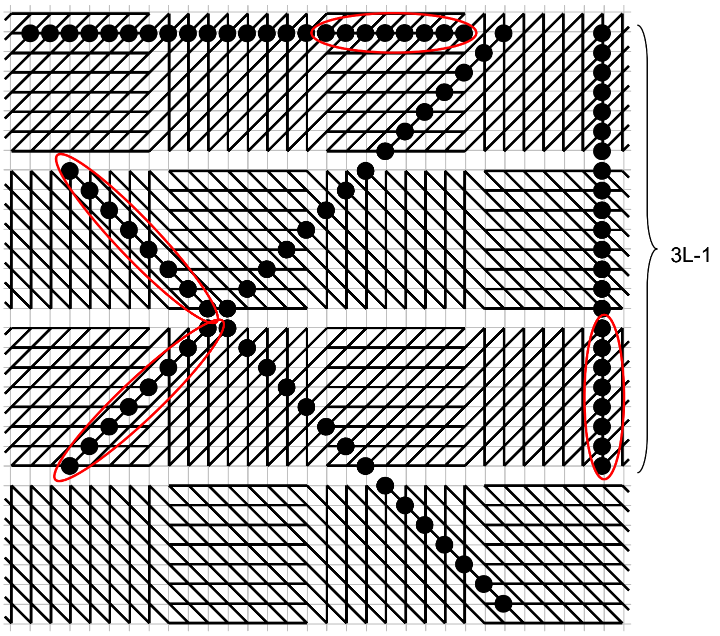


Fig. 6. Partitioning the infinite board into disjoint $B_Z(L)$.

in Fig. 5(a), where each (solid) line covers L squares and the game board extends infinitely to both sides. The game $B_{-Z}(L)$ is a horizontal mirror of $B_Z(L)$. Fig. 5(b) also shows another similar game board $B_N(L)$, which will be used in this section. Let *MBBoardZ*(L, p) denote the game *MBBoard*$(B_Z(L), p)$, and *MBBoardN*(L, p) denote *MBBoard*$(B_N(L), p)$, for simplicity of discussion. This proof will show that the following three properties are satisfied.

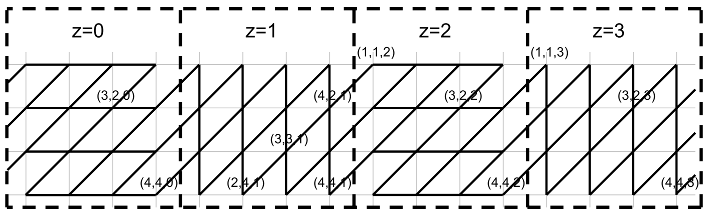
Property 1. *If MBBoardZ*(L, p) *is drawn, then MBConnect*$(3L - 1, p)$ *is drawn.*

Property 2. *If MBBoardN*(L, p) *is drawn, then MBConnect*$(3L - 1, p)$ *is drawn.*

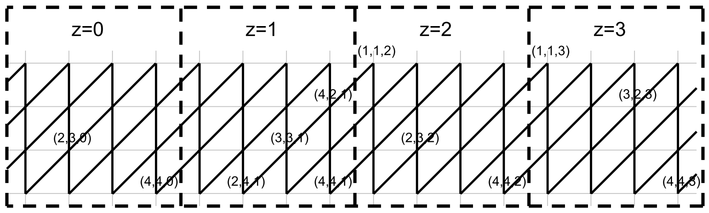
Property 3. *Consider all* $p \geq 1$. *Let* $P(d - 1) < p \leq P(d)$, *where* $P(d) = 2^d - d - 2$. *Then, MBBoardN*$(p + d, p)$ *games are drawn.*

First, Property 1 is satisfied for the following reason. As observed in Fig. 6, all segments of $3L - 1$ consecutive squares vertically, horizontally, and diagonally must contain one whole solid line among these $B_Z(L)$ and $B_{-Z}(L)$. Assume that the game *MBBoardZ*(L, p) is drawn. Then, White has some strategy S such that Black cannot occupy any solid lines inside each $B_Z(L)$ and $B_{-Z}(L)$. Thus, by following the strategy S inside each $B_Z(L)$ and $B_{-Z}(L)$, White prevents Black from occupying any segment of $3L - 1$ consecutive squares completely. Thus, *MBConnect*$(3L - 1, p)$ is drawn.

Then, both Properties 2 and 3 are shown in Sections 4.1 and 4.2, respectively. Section 4.1 shows that the game board $B_Z(L)$ is *isomorphic* to $B_N(L)$, in the sense of *hypergraphs* [4,8], and that Property 2 is satisfied from the isomorphism and Property 1. Section 4.2 proves that Property 3 is satisfied for all *MBBoardN* games listed in Property 3. Thus, Theorem 2 is satisfied from Corollary 1, Property 2 and Property 3. □



(a) $B_Z(4)$.



(b) $B_N(4)$.

Fig. 7. Coordinate mapping between $B_Z(4)$ and $B_N(4)$.

4.1. Isomorphism

Both game boards $B_Z(L)$ and $B_N(L)$ are hypergraph isomorphic [4,8] according to the following mapping. Let every L neighboring vertical or horizontal solid lines be grouped into one zone in both $B_Z(L)$ and $B_N(L)$, as shown respectively in Fig. 7(a) and (b). In both game boards, each square has a coordinate (x, y, z), where the square is in the xth column (from the left) and in the yth row (from the top) in zone z. Let each square at (x, y, z) on $B_Z(L)$ be mapped into the one at (x, y, z) on $B_N(L)$ when z is even, and at (y, x, z) on $B_N(L)$ when z is odd. All solid lines (or hyperedges) on $B_Z(L)$ are mapped into those on $B_N(L)$ accordingly, except that the ith horizontal line (from the top) of $B_Z(L)$ is mapped to the ith vertical line (from the left) of $B_N(L)$ in zone z, where z is odd.

Lemma 3. *Consider both MBBoardZ(L, p) and MBBoardN(L, p) games over all L and p. Then, MBBoardZ(L, p) is drawn if and only if MBBoardN(L, p) is drawn.*

Proof. According to the above mapping from $B_Z(L)$ to $B_N(L)$, placing one stone at (x, y, z) in $B_Z(L)$ is equivalent to placing one stone at (x, y, z) in $B_N(L)$ when z is even, and at (y, x, z) when z is odd, and *vice versa*. Since both $B_Z(L)$ and $B_N(L)$ are hypergraph isomorphic for the mapping, one solid line of $B_Z(L)$ is occupied by Black if and only if the mapped solid line of $B_N(L)$ is. Therefore, *MBBoardZ*(L, p) is drawn if and only if *MBBoardN*(L, p) is drawn. □

From Lemma 3 and Property 1, Property 2 is satisfied.

4.2. Drawn MBBoardN games

This section will prove that Property 3 is satisfied. First, we introduce the concept of exclusive squares in Section 4.2.1, which is used in the remaining subsections. In order to prove that all *MBBoardN* games are drawn in Property 3, we derive some initial drawn *MBBoardN* games in Section 4.2.2, and derive induction rules for *MBBoardN* games in Section 4.2.3. Finally, Section 4.2.4 concludes that Property 3 is satisfied.

4.2.1. Game boards with exclusive squares

In this subsection, we introduce the concept of *exclusive squares*, on which Black is not allowed to place stones. The game boards with exclusive squares are defined in Definition 2 (below).

Definition 2. *MBBoardX*(B, b) is a Maker–Breaker game defined as follows.

1. The game board B is the same as that in Definition 1, except for the following. For each line, one extra square is added as an *exclusive square*, as illustrated with solid bullets in Fig. 8(a)–(c).
2. In Move $2i - 1$, where $i \geq 1$, Black is allowed to place *any (positive) number* of black stones, say p' (≥ 1) black stones, on the game board B. However, Black is *not* allowed to place stones on these exclusive squares. In Move $2i$, White is allowed to place p' or fewer white stones on *any* squares (including exclusive squares).
3. Black wins if the following condition holds. An active line contains *more than* b black stones at time t_{2i} (when Black is to play), where $i \geq 0$. Time t_j indicates the moment after Move j and before Move $j + 1$, and t_0 indicates the initial moment. □

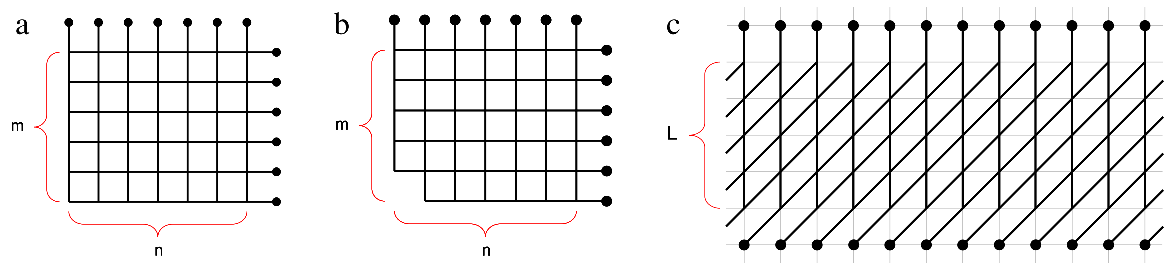


Fig. 8. Three game boards with exclusive squares (solid bullets). (a) $B_{recX}(m, n)$. (b) $B_{recX-}(m, n)$. (c) $B_{NX}(L)$.

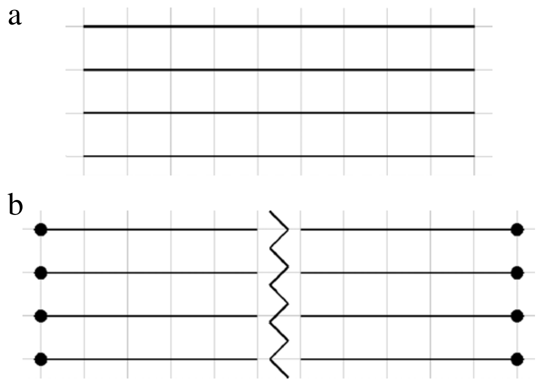


Fig. 9. An illustration. (a) The original game board. (b) Partitioned game boards with exclusive squares.

The game *MBBoardX*(B, b) is said to be a drawn game if White has some strategy to prevent Black winning in all cases.

The motivation of using exclusive squares is to partition a game board into two or more game boards with exclusive squares and then to use Lemma 4 (below) to derive some properties from the partitioned game boards. Let us illustrate it by a simple game *MBBoard*$(B, 9)$ as follows. Let the board B contain disjoint lines each with 10 squares (which are not covered by any other lines), as shown in Fig. 9(a). Then, partition the board B into two, one named B_{left} containing 5 squares of each line and the other B_{right} containing the other 5, and add exclusive squares to all lines as shown in Fig. 9(b). Clearly, both games *MBBoardX*$(B_{left}, 0)$ and *MBBoardX*$(B_{right}, 0)$ are drawn, for the following reason. Whenever Black places one or more stones on some line, White places one stone on the exclusive square of the line to defend. From Lemma 4, we obtain that *MBBoard*$(B, 10 - (0 + 0) - 1)$ is drawn; that is, *MBBoard*$(B, 9)$ is drawn. Obviously, it is true that *MBBoard*$(B, 9)$ is drawn, from the following observation. Whenever Black places one or more stones on some active line, White places one stone on that line in the next move to make it inactive. Note that Black must leave one square unoccupied in an active line, so White is allowed to place a stone on that line.

Lemma 4. *Consider a game board B, where each line covers at least L squares. Partition*[3] *the game board B into two disjoint game boards, B_1 and B_2. Assume that both games MBBoardX(B_1, b_1) and MBBoardX(B_2, b_2) are drawn and that $L - (b_1 + b_2) > 1$. Then, White has some strategy in MBBoard$(B, L - (b_1 + b_2) - 1)$ such that each active line in B contains at most $b_1 + b_2$ black stones at all times t_{2i} (when Black is to play), where $i \geq 0$. Implicitly, MBBoard$(B, L - (b_1 + b_2) - 1)$ is drawn.*

Proof. It suffices to prove by induction that White has some strategy such that each active line in B contains at most $b_1 + b_2$ black stones at all times t_{2i}, where $i \geq 0$. This implies that *MBBoard*$(B, L - (b_1 + b_2) - 1)$ is drawn, since Black cannot occupy any active line (at most $b_1 + b_2$ black stones) in the next move (at most $L - (b_1 + b_2) - 1$ black stones), and each line covers at least L $(\geq (b_1 + b_2) + L - (b_1 + b_2) - 1 = L - 1)$ squares.

It is trivial that the induction hypothesis is true initially.

Assume that the induction hypothesis is true at t_{2i}, when Black is to move. Consider Black's next move. Since Black can place at most $L - b_1 - b_2 - 1$ stones in a move, each active line must leave one square unoccupied. Now, investigate the black stones of this move in B_1. Since *MBBoardX*(B_1, b_1) is drawn according to the assumption, White must has some strategy for the game such that each active line contains at most b_1 black stones in B_1 at t_{2i+2}. Thus, White simply follows the strategy to place stones at the edge of B_1. In the case that White needs to place a stone on the exclusive square in one active line in B_1, White uses the following strategy. If the corresponding line in B is inactive (e.g., the line contains a white stone at the edge of B_2), simply ignore this line. Otherwise, if it is active, White simply places one stone on the unoccupied square of the line

[3] In the partitioning, we assume that each square belongs to either B_1 or B_2 and that each pair of squares in either B_1 or B_2 is covered by one line if they are also covered by the same line in B.

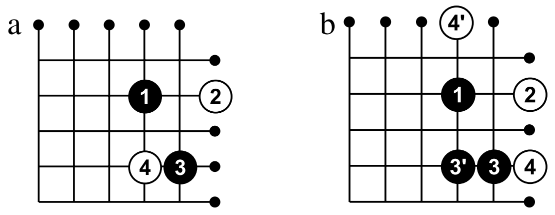


Fig. 10. Two cases for $B_{recX}(m, n)$.

as described above. Note that it does not matter even if the unoccupied square is at the edge of B_2. Thus, White ensures that each active line contains at most b_1 black stones at the edge of B_1 at t_{2i+2}. Similarly, White also ensures that each active line contains at most b_2 black stones in B_2 at t_{2i+2}. Thus, the induction hypothesis is true at t_{2i+2}. □

In this paper, we consider three game boards with exclusive squares, as shown in Fig. 8. The first game board, denoted by $B_{recX}(m, n)$ and shown in Fig. 8(a), consists of m horizontal lines and n vertical lines, each of which contains one extra exclusive square. The second, denoted by $B_{recX-}(m, n)$ and shown in Fig. 8(b), is the same as $B_{recX}(m, n)$ except that the square at the lower-left corner is removed. The third, shown in Fig. 8(c), is the original $B_N(L)$ extended with one exclusive square for each line. For simplicity, let *MBBoardNX*(L, b) denote the game *MBBoardX*($B_N(L)$, b) and $B_{NX}(L)$ denote the game board $B_N(L)$ with extra exclusive squares. Three properties related to the above three boards are shown respectively in Lemma 5, Lemma 6, and Lemma 7 (below).

Lemma 5. *MBBoardX*($B_{recX}(m, n)$, 1) *is drawn over all m and n.*

Proof. Let variables $\sigma_R(r)$ and $\sigma_C(c)$ respectively be the number of black stones in the rth horizontal line and that in the cth vertical line, if still active, and be 0, otherwise. Let variable $\sigma = \Sigma_R\, \sigma_R(r) + \Sigma_C\, \sigma_C(c)$. For this proof, it suffices to prove that White has a strategy such that $\sigma \leq 1$ at all times t_{2i} (when Black is to play), where $i \geq 0$.

Assume by induction that $\sigma \leq 1$ at some t_{2i}. Assume that, in Move $2i + 1$, Black places only one stone on square s at row r and column c. Obviously, Move $2i + 1$ increases σ by at most two (one for the vertical line and the other for the horizontal line). That is, $\sigma \leq 3$. White uses the following strategy to make Move $2i + 2$ such that $\sigma \leq 1$ at t_{2i+2}.

1. When $\sigma \leq 1$, simply place a stone randomly on one empty square, if any.
2. When $\sigma \leq 2$, simply choose one active line containing a black stone and block it by placing one white stone on the exclusive square in that line. Then, σ is at most 1.
3. When $\sigma = 3$ and an active line contains two black stones, simply block the active line by placing one white stone on the exclusive square in that line. Then, σ is at most 1.
4. In the remaining case that $\sigma = 3$ and none of the active lines contains two black stones, assume some $\sigma_R(r') = 1$, where $r' \neq r$, without loss of generality. Thus, the square s' at row r' and column c (both lines are active) must be empty (otherwise, we are in Case 3, since two black stones are in the same column). Therefore, simply place one white stone on s'. Since the stone blocks the two active lines in row r' and column c, σ is back to 1. This is illustrated by Moves 3 and 4 in Fig. 10(a).

However, if Black places several black stones, say p' black stones, in Move $2i+1$, we separate the move into p' submoves, each with one stone only. Then, White pretends that Black makes submoves one by one, and therefore follows the above strategy to place stones, except for the following case. If White is to place one stone on an empty square s' in some submove M as in Case 4, but one of the subsequent submoves M' places one black stone on s' too, the strategy needs to be changed as follows.

5. Place two white stones respectively on the exclusive squares of the two active lines in row r' and column c containing s'. Thus, σ is back to 1 too. Thus, for M', White replies by placing no more stones. In this case, the two white stones together are viewed as a reply to the two black stones at submoves M and M'. This case is illustrated by the example in Fig. 10(b). For Move 3, Black places two stones at 3 and 3'. Assume Black to make submoves in the sequence 3 and then 3'. For 3, White cannot reply by placing a stone on 3', since it will be occupied by Black. Therefore, White places stones on 4 and 4' to make σ back to 1, instead.

From the above strategy, $\sigma \leq 1$ is maintained at all times t_{2i}. Thus, this lemma holds. □

Lemma 6. *MBBoardX*($B_{recX-}(m, n)$, 1) *is drawn over all m and n.*

Proof. This proof is the same as that in Lemma 5, except for the first black stone and White's reply. The first black stone is placed on the board in the following three positions: (1) in the leftmost vertical line, (2) in the bottom horizontal line, and (3) in the rest of the rectangle. In Case 1, let White reply by placing one white stone on the leftmost vertical line as shown in Fig. 11(a), thus making this vertical line inactive. Now, the variable σ is only 1. Then, we simply follow the strategy described in Lemma 5 to maintain $\sigma \leq 1$. Similarly, in Case 2, let White reply by placing one on the bottom horizontal line. In Case 3, let White place one on the leftmost vertical line without loss of generality, while blocking the first black stone in

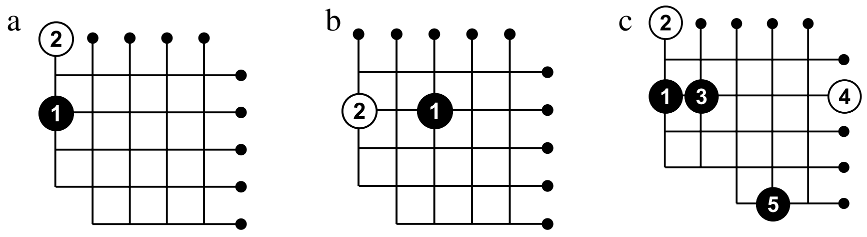


Fig. 11. (a) and (b): Two cases for $B_{recX-}(m, n)$ in and (c) another case for the board missing two corner squares.

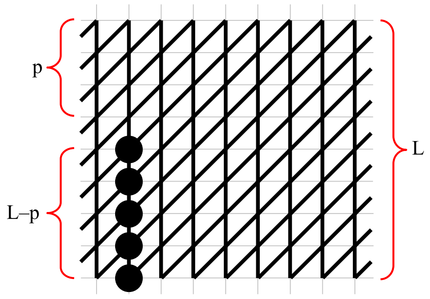


Fig. 12. The case that Black already occupies $L - p$ stones on an active line.

the same horizontal line as shown in Fig. 11(b). Similarly, since the variable σ is only 1, simply follow the strategy described in Lemma 5 to maintain $\sigma \leq 1$. Thus, White is able to maintain $\sigma \leq 1$ in all cases. That is, *MBBoardX*$(B_{recX-}(m, n), 1)$ is drawn. (Note that we may not maintain $\sigma \leq 1$ when two corner squares are missing, as illustrated in Fig. 11(c).) □

Lemma 7. *As described above, assume that the game MBBoardN*(L, p) *is drawn. Then, MBBoardNX*$(L, L - p - 1)$ *is drawn.*

Proof. Since *MBBoardN*(L, p) is drawn, White has a strategy S such that all active lines have at most $L - p - 1$ black stones at all times t_{2i} (when Black is to play). Otherwise, if an active line contains at least $L - p$ black stones, Black wins by simply placing p stones on this line, as illustrated in Fig. 12.

In the game *MBBoardNX*$(L, L - p - 1)$, assume that Black still places at most p black stones in Move $2i + 1$, where $i \geq 0$. Then, White simply follows strategy S (without placing stones on exclusive squares) such that all active lines in $B_{NX}(L)$ contain at most $L - p - 1$ black stones at all times t_{2i+2} (when Black is to play).

Assume that Black makes a move with more than p black stones. We separate the move into several submoves, each with at most p black stones. Then, White pretends that Black makes submoves one by one, and for each submove simply follows S to play, but with the following exceptional case. By following S, assume that White needs to make a submove on some empty squares, but some subsequent Black submoves will place stones on these empty squares. Without loss of generality, assume that White makes a submove M on an empty square s, but some subsequent Black submove M' will place a stone on s. Then, the strategy is changed as follows.

1. Place two white stones respectively on the exclusive squares of the two lines containing s, instead. The reason is similar to that in Case 5 in Lemma 5. Both lines containing s are no longer active. Let the black stone at s be added into M and removed from M'. Thus, the reply to M still prevents Black from having active lines with more than $L - p - 1$ black stones. Although the reply to M uses one more stone, M has one more stone on s too.

Thus, all active lines in the game *MBBoardNX*$(L, L - p - 1)$ have at most $L - p - 1$ black stones at all t_{2i} (when Black is to play). That is, *MBBoardNX*$(L, L - p - 1)$ is drawn. □

4.2.2. Initial drawn games

In this subsection, initial *MBBoardN*(4, 1), *MBBoardNX*(2, 1) and *MBBoardNX*(3, 2) games are shown to be drawn in Lemma 8, Lemma 9, and Lemma 10 respectively.

Lemma 8. *MBBoardN*(4, 1) *is drawn.*

Proof. Let us transform $B_N(4)$ into $B_{N-}(4)$ by shortening the solid lines, as shown in Fig. 13. Since $B_{N-}(4)$ is a tree and there are no black stones initially, $B_{N-}(4)$ is drawn, from Lemma 2. Obviously, this implies that $B_N(4)$ with extra longer lines is drawn too. □

Lemma 9. *MBBoardNX*(2, 1) *is drawn.*

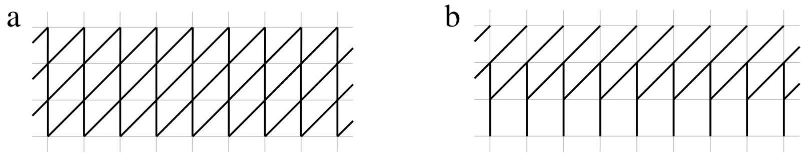


Fig. 13. (a) $B_N(4)$. (b) $B_{N-}(4)$, the same as $B_N(4)$ except that all the solid lines are shortened.

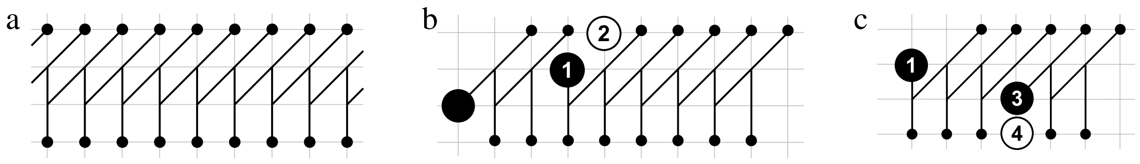


Fig. 14. (a) $B_{NX}(2)$. (b) The tree broken by White, Move 2. (c) The tree broken by White, Move 4.

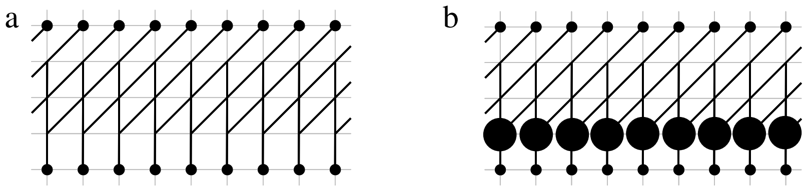


Fig. 15. (a) $B_{NX}(3)$. (b) $B_{NX}(3)$ occupied by some black stones initially.

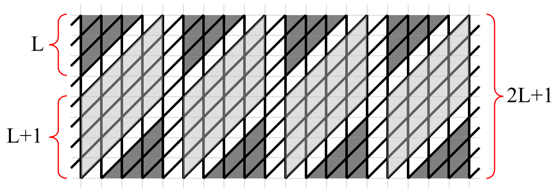


Fig. 16. Partitioning $B_N(2L + 1)$ into dark gray and light gray zones.

Proof. The game board $B_{NX}(2)$ is a tree, as shown in Fig. 14(a). First, we assume that Black places one stone for each move. It suffices to prove that White has a strategy such that at all times t_{2i} (when Black is to play) each of the trees (formed by all the active lines) satisfies that only the leftmost (active) line, if it exists, contains one black stone. For example, in Fig. 14(b), for Move 1 (by Black), Move 2 (by White) blocks the diagonal line on Move 1; and in Fig. 14(c), for Move 3, Move 4 blocks the vertical line containing the stone of Move 3. Thus, it is easy to see that no active lines contain two black stones at all times t_{2i}. If Black places several stones in one move, we simply pretend that Black places stones one at a time. White simply follows the above strategy without being disturbed by Black's multi-stone moves, since White replies by placing stones on exclusive squares where Black cannot place stones. Thus, *MBBoardNX*(2, 1) is drawn. □

Lemma 10. *MBBoardNX*(3, 2) *is drawn.*

Proof. For game board $B_{NX}(3)$ as shown in Fig. 15(a), assume that all squares above the bottom exclusive squares are initially occupied by black stones, as shown in Fig. 15(b). By ignoring these squares with black stones, the game board becomes $B_{NX}(2)$. From Lemma 9, at all times t_{2i} (when Black is to play), Black occupies at most one of the remaining two squares plus the one already shown in Fig. 15(b), that is, at most two. Thus, *MBBoardNX*(3, 2) is drawn. □

4.2.3. Induction rules

In this subsection, four induction rules are shown in Lemma 11, Lemma 12, Lemma 13, and Lemma 14 respectively.

Lemma 11. *Assume that MBBoardNX*(L, b) *is drawn, where* $0 < b < L$. *Then, MBBoardN*$(2L + 1, 2L - b - 1)$ *is drawn too.*

Proof. Partition the game board $B_N(2L + 1)$ into dark gray and light gray game boards, as shown in Fig. 16. Half of the dark gray board can be squeezed into $B_N(L)$, as shown in Fig. 17. The light gray game board is the union of disjoint $B_{recX}(L + 1, L + 1)$. Since *MBBoardNX*(L, b) is drawn from the assumption and *MBBoardX*$(B_{recX}(L + 1, L + 1), 1)$ is drawn from Lemma 5, *MBBoardN*$(2L + 1, (2L + 1) - (b + 1) - 1) =$ *MBBoardN*$(2L + 1, 2L - b - 1)$ is drawn from Lemma 4. □

Lemma 12. *Assume that MBBoardNX*(L, b) *is drawn, where* $0 < b < L$. *Then, MBBoardN*$(2L + 2, 2L - b)$ *is drawn too.*

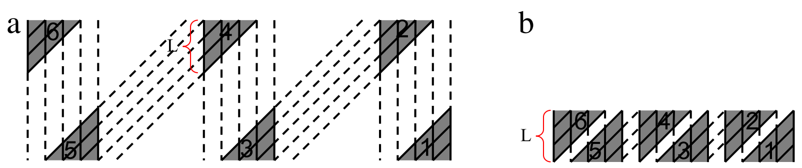


Fig. 17. (a) Half of the dark gray game board. (b) Squeezing the game board in (a) into a $B_N(L)$.

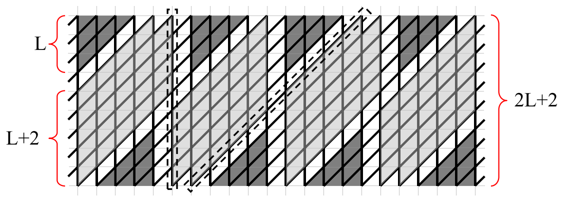


Fig. 18. Partitioning $B_N(2L+2)$ into light gray and dark gray zones.

Table 1
List of drawn *MBBoardN* games derived from Property 4, where $2 \le p \le 4$.

Drawn games		Drawn games derived from Lemma 11 or Lemma 12.
MBBoardNX(2, 1)	$\to$	*MBBoardN*(5, 2) and *MBBoardN*(6, 3)
MBBoardNX(3, 2)	$\to$	*MBBoardN*(7, 3) and *MBBoardN*(8, 4)

Table 2
List of drawn *MBBoardN* games derived from Property 5, where $5 \le p \le 13$.

Drawn games		Drawn games derived from Lemmas 13 and 14.
MBBoardN(4, 1)	$\to$	*MBBoardN*(9, 5) and *MBBoardN*(10, 6)
MBBoardN(5, 2)	$\to$	*MBBoardN*(11, 7) and *MBBoardN*(12, 8)
MBBoardN(6, 3)	$\to$	*MBBoardN*(13, 9) and *MBBoardN*(14, 10)
MBBoardN(7, 3)	$\to$	*MBBoardN*(15, 10) and *MBBoardN*(16, 11)
MBBoardN(8, 4)	$\to$	*MBBoardN*(17, 12) and *MBBoardN*(18, 13)

Proof. This proof is similar to that in Lemma 11, except that $B_{recX-}(L+2, L+2)$ is used (instead of B_{recX}) and some lines marked in dashed boxes in Fig. 18 are covered by two $B_{recX-}(L+2, L+2)$. For the lines covered by two $B_{recX-}(L+2, L+2)$, since each active line in $B_{recX-}(L+2, L+2)$ contains at most one black stone, each of these lines, if active, contains at most two black stones when Black is to play. For the other lines, we can still use Lemma 4 to derive that each line, if active, contains at most $b+1$ black stones when Black is to play. Since $b+1 \ge 2$, all lines contain at most $b+1$ black stones when Black is to play. Thus, the game $MBBoardN(2L+2, (2L+2)-(b+1)-1) = MBBoardN(2L+2, 2L-b)$ is drawn. □

Lemma 13. *Assume that MBBoardN(L, p) is drawn. Then, MBBoardN(2L + 1, L + p) is drawn too.*

Proof. Since *MBBoardN*(*L*, *p*) is drawn, *MBBoardNX*($L, L-p-1$) is drawn from Lemma 7. From Lemma 11, *MBBoardN*($2L+1$, $2L-(L-p-1)-1$) = *MBBoardN*($2L+1, L+p$) is drawn. Thus, this lemma holds. □

Lemma 14. *Assume that MBBoardN(L, p) is drawn. Then, MBBoardN(2L + 2, L + p + 1) is drawn too.*

Proof. Since *MBBoardN*(*L*, *p*) is drawn, *MBBoardNX*($L, L-p-1$) is drawn from Lemma 7. From Lemma 12, *MBBoardN*($2L+2$, $2L-(L-p-1)$) = *MBBoardN*($2L+2, L+p+1$)) is drawn. Thus, this lemma holds. □

4.2.4. The proof for Property 3

This subsection concludes in Lemma 15 that Property 3 is satisfied.

Lemma 15. *Property 3 is satisfied.*

Proof. Initially, the three games, *MBBoardN*(4, 1), *MBBoardNX*(2, 1) and *MBBoardNX*(3, 2), are shown to be drawn in Lemma 8, Lemma 9, and Lemma 10, respectively. From Lemma 11 or Lemma 12, we obtain the drawn *MBBoardN* games, for all $2 \le p \le 4$, as shown in Table 1. Then, from Lemmas 13 and 14, we obtain the drawn *MBBoardN* games, for all $5 \le p \le 13$, as shown in Table 2. By induction, all the remaining drawn *MBBoardN* games in Property 3 can be derived from Lemmas 13 and 14. □

5. Conclusion

The contributions of this paper are listed as follows.

- With the help of a program, this paper shows that *Connect*(11, 2) is drawn. Note that drawn *Connect*(k, p) implies drawn *Connect*(m, n, k', p, q) for all $k' \geq k$, $m \geq 1$, $n \geq 1$, $0 \leq q \leq p$. In contrast, the best known result [10] in the past was drawn *Connect*(15, 2).
- This paper shows that *Connect*($k_{draw}(p)$, p) games are drawn for all $p \geq 3$, where $k_{draw}(p) = 3p + 3d - 1$ and d is a logarithmic function of p. Specifically, d is an integer such that $P(d-1) < p \leq P(d)$ and $P(d) = 2^d - d - 2$. The values $k_{draw}(p)$ derived in this paper are currently the smallest for all $2 \leq p < 1000$ (the value is the same as that in [10] when $p = 4$).

Although this paper presents tighter bound for k, many interesting problems are still open. The following are two examples.

- Derive lower $k_{draw}(p)$ for $p < 1000$, especially for small p, e.g., $1 \leq p \leq 10$. These problems are more realistic in real games. For example, *Connect*(5, 1) favors Black [1,2], while *Connect*(8, 1) is drawn [17]. There is still a gap between 5 and 8.
  When $p = 2$, the gap is even wider. Currently, the conjecture by most *Connect*6 players are that *Connect*6, *Connect*(19, 19, 6, 2, 1), is drawn, and that Black wins in *Connect*(19, 19, 6, 2, 2). Both are still open problems. A search approach similar to those in [15,16] is perhaps helpful to solve the latter. However, from our experiences, it is very difficult to use the search approach to solve the former. It is also an important open problem to solve all *Connect*(n, 2), where $7 \leq n \leq 10$.
- Derive general tighter bounds than those in this paper and those in [11] simultaneously.

Acknowledgements

The authors would like to thank the National Science Council of the Republic of China (Taiwan) for financial support of this research under contract numbers NSC 95-2221-E-009-122-MY2 and NSC 97-2221-E-009-126-MY3. The authors would also like to thank Po-Ting Chen for his assistance with the program in Theorem 1 and the anonymous referees for their valuable comments.

References

[1] L.V. Allis, Searching for solutions in games and artificial intelligence. Ph.D. Thesis, University of Limburg, Maastricht, the Netherlands, 1994.
[2] L.V. Allis, H.J. van den Herik, M.P.H. Huntjens, Go-moku solved by new search techniques, Computational Intelligence 12 (1996) 7–23.
[3] J. Beck, On positional games, Journal of Combinatorial Theory Series A 30 (1981) 117–133.
[4] C. Berge, Graphs and Hypergraphs, North Holland, Amsterdam, 1973.
[5] E.R. Berlekamp, J.H. Conway, R.K. Guy, Winning Ways for your Mathematical Plays, vol. 3, 2nd ed., A K Peters. Ltd, Canada, 2003.
[6] S.-H. Chiang, I-C. Wu, P.-H. Lin, On drawn k-in-a-row games, in: The 12th Advances in Computer Games Conference, ACG12, Pamplona, Spain, May 2009.
[7] L. Csirmaz, On a combinatorial game with an application to Go-moku, Discrete Mathematics 29 (1980) 19–23.
[8] R. Diestel, Graph Theory, 2nd edition, Springer, New York, 2000.
[9] H.J. van den Herik, J.W.H.M. Uiterwijk, J.V. Rijswijck, Games solved: now and in the future, Artificial Intelligence 134 (2002) 277–311.
[10] M.-Y. Hsieh, S.-C. Tsai, On the fairness and complexity of generalized k-in-a-row games, Theoretical Computer Science 385 (2007) 88–100.
[11] A. Pluhar, The accelerated k-in-a-row game, Theoretical Computer Science 270 (1–2) (2002) 865–875.
[12] I-C. Wu, D.-Y. Huang, A new family of k-in-a-row games, in: The 11th Advances in Computer Games, ACG11, Conference, Taipei, Taiwan, 2005.
[13] I-C. Wu, D.-Y. Huang, H.-C. Chang, Connect6, ICGA Journal 28 (4) (2006) 234–242.
[14] I-C. Wu, P.-H. Lin, Search tree for drawn Connect(11, 2). Available at http://www.connect6.org/articles/drawn-connect-games/.
[15] I.-C. Wu, P.-H. Lin, Relevance-zone-oriented proof search for Connect6, IEEE Transactions on Computational Intelligence and AI in Games 2 (3) (2010) 191–207.
[16] I-C. Wu, H.-H. Lin, P.-H. Lin, D.-J. Sun, Y.-C. Chan, B.-T. Chen, Job-Level Proof-Number Search for Connect6. in: The International Conference on Computers and Games 2010, CG2010, Kanazawa, Japan, September 2010.
[17] T.G.L. Zetters, 8(or more) in a row, American Mathematical Monthly 87 (1980) 575–576.