國立交通大學

資訊科學與工程研究所

碩士論文

工作層級之 CNS 搜尋研究

Job-Level Conspiracy Number Search

研 究 生：陳柏廷

指導教授：吳毅成　教授

中華民國　九十九　年八月

工作層級之 CNS 搜尋研究
Job-Level Conspiracy Number Search

研 究 生：陳柏廷　　Student：Po-Ting Chen

指導教授：吳毅成　　Advisor：I-Chen Wu

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

August 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年八月

工作層級之 CNS 搜尋研究

研究生：陳柏廷　　指導教授：吳毅成

國立交通大學 資訊科學與工程研究所

摘要

六子棋是 2005 年由吳毅成教授所發明的一種棋類遊戲，由於六子棋是一種相當新的遊戲，開局譜的數量相對目前大眾熟知的遊戲象棋、圍棋，來的少很多。

2010 年由吳毅成教授的研究團隊，做出了一個六子棋驗證系統，使用一種叫做 Job-Level Proof Number Search 演算法，目的是可以驗證六子棋盤面勝敗，也成功的找出許多棋局的勝敗，該系統並配合網格計算，達到平行化加速的目的。

這篇研究目的是在改善六子棋驗證系統，除了達到加速驗證的效果，透過加入一種稱為 Conspiracy Number Search 的演算法，對於找不出勝敗的盤面，該驗證系統也能從搜尋的結果中，分析出適當的著手。

Job-Level Conspiracy Number Search

Student: Po-Ting, Chen Advisor: I-Chen, Wu

Institute of Computer Science and Engineering
National Chiao Tung University

Abstract

Connect6 is a board game which was invented by professor Wu. Because Connect6 is a novel game, the number of public openings is relatively small when compared with Chinese Chess and Go.

Professor Wu's research lab had designed the Connect6 verification system that used Job-Level proof Number Search to solve Connect6 positions. This system also used a desktop grid to achieve the parallel speedup.

The main purpose of this research is to improve the Connect6 verification system by using Conspiracy Number Search to find and analyze game positions.

誌謝

這篇論文的得以完成，首先感謝我的指導教授吳毅成老師，在研究期間，給予我相當多協助與意見，並不吝嗇的花許多時間在指導我研究的方法與態度。另外也感謝口試委員所給予的意見，讓這篇論文在架構及內容上更加完整及豐富。

在這篇論文研究期間，非常感謝鈊象電子股份有限公司所贊助的獎學金，使我在這段研究期間，能更加專心與投入研究。

再來感謝實驗室的學長，林秉宏、林宏軒學長，給予我相當多的意見，在研究的想法與方法上；接著也感謝 97 級實驗室的同學，林彥成、陳俊嶧、鄒忻芸、廖家茵與黃郁雯，在這兩年的碩士生涯中，使我的生活中充滿了許多的歡笑；另外非常感謝實驗室的學弟韓尚餘，在研究的實驗期間，提供並維護實驗用的機器，使本篇論文在最後的實驗中，可以如期完成；沒有你們，這篇論文是一定無法完成的，非常感謝各位在我的研究期間，不吝嗇給予的協助與幫忙。

最後感謝我的家人，在我最後這段研究的期間，給我相當多的鼓勵，使我都能以穩定的心情投入研究上，謝謝你們。

最後，謹以此篇論文，獻給所有在我這兩年研究路上，協助及陪伴我的老師、朋友及家人，謝謝你們。

民國九十九年八月　於　新竹交大工程三館 CYC-511 實驗室

目錄

圖表目錄

表格目錄

第一章、介紹

本篇研究會實做在六子棋人工 AI 上面，因此在一開始的章節中，會介紹到六子棋這個遊戲。

六子棋為國立交通大學吳毅成教授於 2005 年發明的新遊戲[31]，目前已登入國際線上維基百科全書，並成為奧林匹亞電腦賽局競賽證項目之一，在許多網站中，也有提供玩家六子棋對弈，如 Little Golem[3]及 CYC 遊戲大聯盟[7]。在本章節中會介紹這個遊戲的特性，並提出相關研究的動機，在章節 1.1 之中會介紹六子棋，章節 1.2 介紹何謂六子棋開局庫，章節 1.3 提出研究動機與目的。

1.1. 六子棋介紹

六子棋的特性為規則簡單、遊戲公平、以及變化複雜[1][5]，以下分別就這三個特性來做說明。

規則簡單。六子棋遊戲一般使用的棋盤大小為 19*19(圖 1)，先者持黑，後者持白，黑方的第一手只能下一顆子，接著黑白雙方每手各下兩顆子，在盤面上先連成六顆子的人贏得比賽(圖 2)。不同於五子棋，為了限制黑方(先手方)的優勢，黑方在遊戲中有許多限制，像是禁止雙活三、連六不算贏等等，六子棋則除了上述規則外，對於黑白兩方沒有額外的限制。

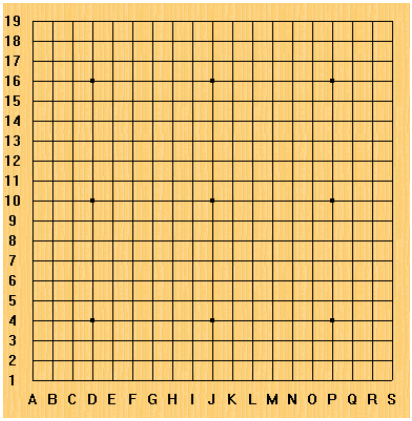


圖 1 六子棋棋盤

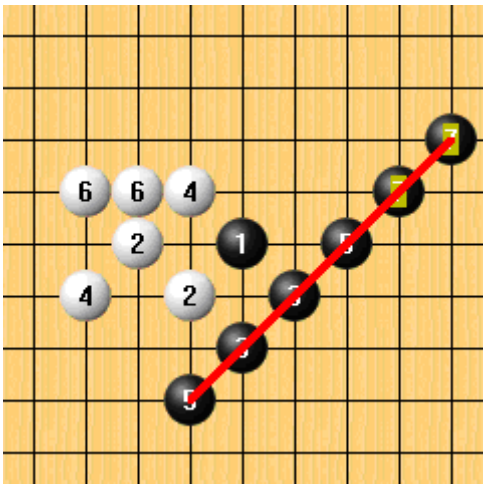


圖 2 六子棋棋局

遊戲公平。首先六子棋遊戲不具脫離戰場特性，所謂的脫離戰場代表一種下棋的策略，即下子時下在離目前盤面上其他棋子很遠的地方，當黑的第一個著手下再天元的位置，而白必須下子在天元附近，否則黑就會必勝。六子棋與五子棋遊戲類似，然而五子棋黑方(先下方)具有優勢，簡單來說，在五子棋遊戲中，當黑方下完一子時，會比白方在場上多一顆子，而白方在場上的棋子數，最多跟黑一樣多。而六子棋遊戲公平，其主要原因在於其遊戲平衡，根據規則，遊戲進行時，每一手下完，皆會比對方多一顆子，相較於五子棋，每一手下完，白子的數目頂多於黑子相同，另外從發明者吳教授對於公平性的定義上，六子棋是具備潛在公平性的特質。

變化複雜。在遊戲的複雜度方面，六子棋的複雜度是介於日本將棋與象棋之間，如表 1 所示。目前電腦棋類 AI 的發展方面，電腦西洋棋 AI 已有正式擊敗人類世界棋王的紀錄，電腦象棋 AI 則是已有人類象棋大師水準。

棋類遊戲	複雜度
19 路圍棋 Go(19*19)	10^{360}
日本將棋 Shogi	10^{226}
六子棋 Connect6	10^{140}-10^{188}
象棋 Chinese Chess	10^{150}
西洋棋 Chess	10^{123}
五子棋 Renju	10^{70}

表格 1 遊戲複雜度

1.2. 六子棋開局庫

在六子棋遊戲 AI 設計上，開局庫的設定也是很重要的一部份，本章介紹關於六子棋開局庫。

1.2.1 定石

所謂的定石，就是指當棋局遇到特定盤面時，會有固定的幾種下法，若是不照著這些下法下棋，雖然不一定會對我方造成必敗，但是會走到對我方不利的棋局。

定石的制定通常是有高段棋士所編纂，高段棋士會利用長期下棋的經驗，與經年累月下棋的技巧，制定並分析出定石。而開局譜就是只發生在開局時的定石。

1.2.2 開局庫製作

在一般的棋類遊戲設計中，可以分為開局、中局、殘局三個階段，開局設計方面通常是建立開局庫，中局跑搜尋演算法，殘局一種可能的設計，是建立殘局庫。而六子棋這個遊戲，沒有所謂的殘局，電腦 AI 設計主要會著手在開局以及中局兩部分，本論文的目的就是在增進開局庫的部分。

在許多的遊戲開局庫建置中，像是圍棋、象棋，都是藉由輸入大量開局譜，從中分析並建構開局庫，在遊戲之初，會將目前盤面拿去開局庫做搜尋，尋找該盤面接下來的下法。

1.2.3 六子棋開局庫

六子棋開局庫的設計，由於六子棋是個相當新的遊戲，於 2005 年發表，相較於象棋[4]、圍棋[6]，這個遊戲的開局譜相對少很多，因此對於利用輸入大量開局譜的方式建構開局，在六子棋上目前較不時用。

為了要建構開局庫，一種可能的方法，就是用程式去產生開局譜，在 2010 年時，六子棋發明人吳毅成教授的研究團隊，發展了一套六子棋驗證系統，裡面所採用的演算法為工作層級證明數演算法(Job-Level Proof Number Search，)[Wu et al. 2010]，簡稱 JL-PNS；該系統目的是可以去驗證六子棋盤面的勝敗，並也成功的找出許多六子棋盤面勝敗，如圖 3 所示，該系統驗證當時六子棋一個常見的開局盤面，米老鼠盤面，找出該盤面黑的必勝下法。

在利用驗證系統來建立開局譜方面，可以輸入所有找到勝敗的棋譜，如圖 3 的盤面，當我方持白時，由於已驗證該盤面是白必敗，因此在開局時，我們可以避免下到圖 3 必敗的開局，或者是當對方走到該盤面時，我們可以開局庫中必勝的下法。

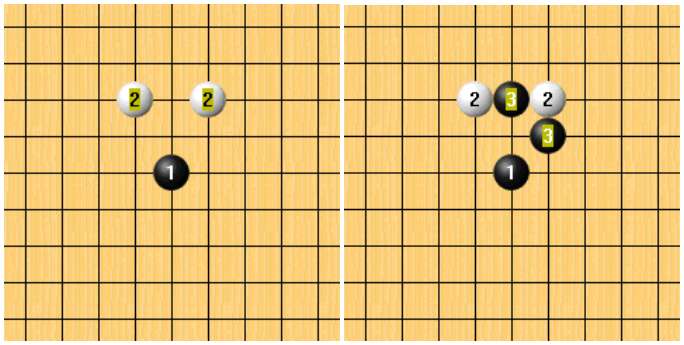
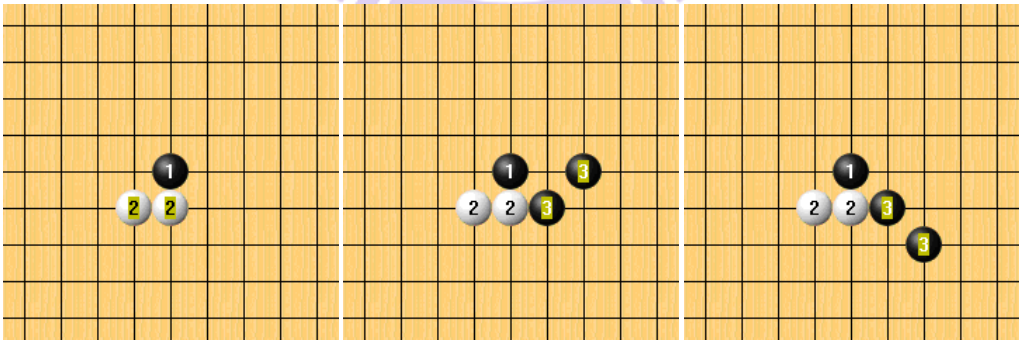


圖 3 必勝棋譜

1.3. 研究目的

上一章節中提到一種如何用程式建構開局的方法，可是對於驗證不出勝敗的盤面，該系統尚未提供尋找適當下的策略，如圖 4 中的盤面，該盤面目前還找不出勝敗；一方面可能是該盤面是個平手的棋局，也有可能其必勝路徑必需搜尋非常的久，在有限的時間內找不出來。

論文的目的是去尋找適當的下法；可以分為兩個部分來說明，一個是當驗證不出勝敗時，能利用程式分析出適當的著手，另一個是加速六子棋驗證系統，藉由更精確的選擇展開搜尋樹的節點，使驗證棋局上的效率提高，因為若能更精確的選點，除了加速驗證棋局外，在尋找適當下法上也是更精確的。



六子棋開局　　可能的下 1　　可能的下法 2

圖 4 六子棋開局

這篇論文的目的是想藉由程式來產生開局庫，也就是利用或是改善六子棋驗證系統，來尋找開局中適當的下法。

1.4 貢獻

這篇論文的貢獻，主要為以下幾項:

- JL-PNS K-Band 機制

  一種更新 JL-PNS 演算法中，證明數(Proof Number)和反證數(Disproof Number)的機制。

- 分析謀反數搜尋演算法(Conspiracy Number Search，簡稱 CNS)與證明數搜尋演算法(Proof Number Search，簡稱 PNS)之間的關係

  CNS 演算法及 PNS 演算法有許多相似之處，這篇論文會分析如何由 CNS 轉變成 PNS。

- 提出工作層級謀反數搜尋演算法(Job-Level Conspiracy Number Search，簡稱 JL-CNS)演算法

  提出一種平行化運算的 CNS 演算法。

- 提供開局選擇適當下法的機制

  利用 JL-PNS 或是 JL-CNS 演算法，選擇棋局適當的下法。

1.5 論文組織

本篇論文第二章是研究背景，在研究背景中，會介紹本篇論文使用的相關程式，及相關研究，包括 PNS、JL-PNS、CNS，第三章是研究方法，該章節中提出兩種改進方法，第一種是觀察 JL-PNS 的缺點，而提出一種改進的機制，第二種就是發展 JL-CNS 演算法，第四章是實驗，在該章節中會有各種改進機制及相關演算法的比較，第五章是結論以及未來展望。

第二章、研究背景

在這個章節中，會介紹吳毅成教授團隊研發出的六子棋驗證系統，在 2.1 章節中會介紹該驗證系統中用到的六子棋程式，2.2 章節中會描述工作層級(Job-Level)的系統環境，簡稱 JL 系統環境，接著介紹該系統所使用的工作層級證明數搜尋(Job-Level Proof Number Search，簡稱 JL-PNS)演算法，由於 JL-PNS 演算法是改良自證明數搜尋演算法(Proof Number Search，簡稱 PNS)演算法，因此在 2.3 章節先介紹 PNS 演算法，2.4 章節在介紹 JL-PNS 演算法。2.5 章節介紹謀反數搜尋(Conspiracy Number Search，簡稱 CNS)演算法。

2.1. 六子棋程式

這章節中，會介紹這篇論文用到相關的程式，本篇論文是在這些程式的基礎上所發展的。

2.1.1 交大六號

吳毅成教授研究團隊有研發一支六子棋程式，名稱為交大六號[14][17][27][28][30][32]，於第 11 屆及第 13 界國際奧林匹亞電腦賽局競賽獲得冠軍[2]。

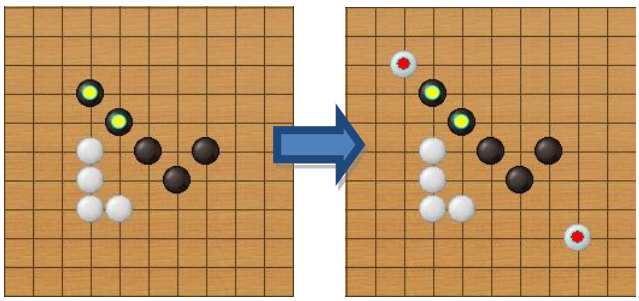


圖 5 交大六號選步

該程式提供下子功能，會依照搜尋結果回傳一步適當的下法，另外除了該下法的位置外，還會回傳該下法的分數及棋局狀態。

棋局狀態代表著目前棋局是對哪一方有利；以圖 5 為例，程式會回傳座標(10,8)、(11,9)兩個下法，棋局狀態為 W:a_w(對白非常有利)；目前總共有 13 種狀態。

狀態			
B:w	➔	黑方必勝	對黑方有利
B:a_w	➔	黑方非常有利	
B3	➔	黑方很有利	
B2	➔	黑方有利	
B1	➔	黑方稍微有利	
Stable	➔	雙方勢均	
W1	➔	白方稍微有利	
W2	➔	白方有利	對白方有利
W3	➔	白方很有利	
W:a_w	➔	白方非常有利	
W:w	➔	白方必勝	

狀態
Unstable 情勢不穩定
Unstable2 情勢非常不穩定

表格 2 交大六號棋局狀態

2.1.2 NCTU6-Verifier

在 NCTU6-Verifier 是由 NCTU6 選步的功能修改而來[Wu and Lin, 2009]，該驗證器功能，會計算出目前盤面可能的擋法，如圖 6，上面代表目前盤面，底下為 NCTU6-Verifier 計算出白方可能的擋法，若下的著手，不在 NCTU6-Verifier 的擋法之中，代表該下法為敗著，另外在 NCTU6-Verifier 計算出的可能擋法，代表的意義該擋法有機會擋住，不代表該著手一定不會敗。

NCTU6-Verifier 有一個不足的地方，就是其列出的可能擋法，並沒有去計算這些擋法優劣，因此並無法從其中得知這些下法的好壞，為了改善這個問題，一種利用 NCTU6 下子功能的驗證器，被提出，在下一章節中介紹。

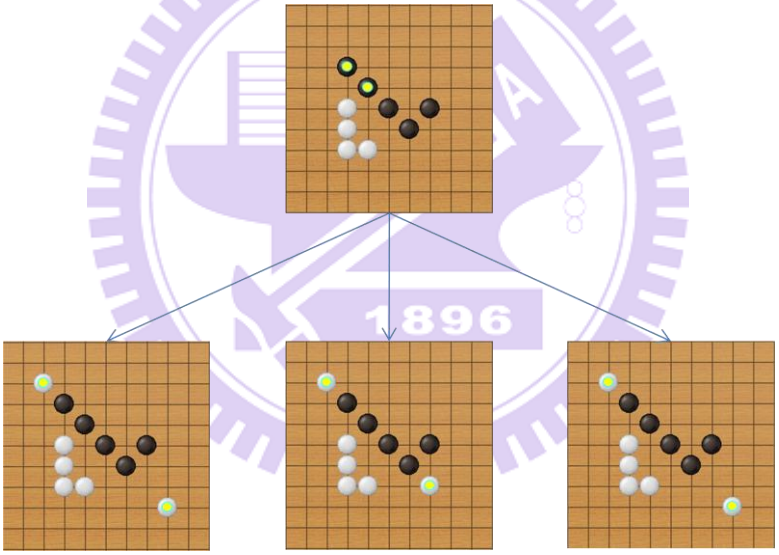


圖 6 NCTU6-Verifer 擋法

2.1.3 驗證(藉由 NCTU6)

NCTU6-Verifier 的功能可以由 NCTU6 下子功能做到，在 NCTU6 下子功能中，除了可以依照 AI 搜尋結果回傳一個最好的著手，也可以要求程式排除最佳著手，在由搜尋結果產生次佳的下法，利用這種排除著手方法，NCTU6 可以最好著手、第二好的著手、第三好的著手等等，而當所有可能的著手都被排除時，

就會回傳一個必敗訊息。

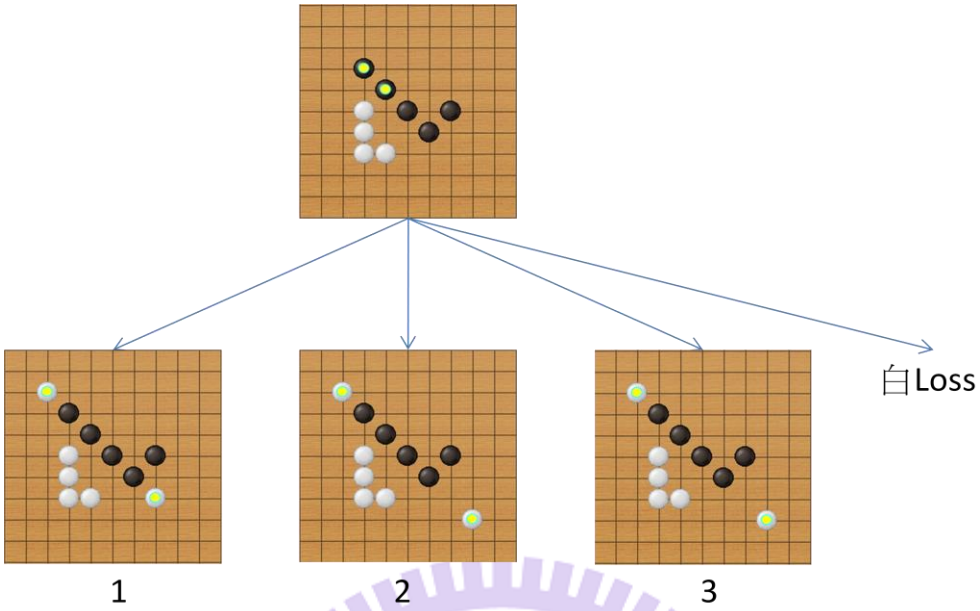


圖 7 Verification by NCTU6

2.1.4 六子棋編輯器

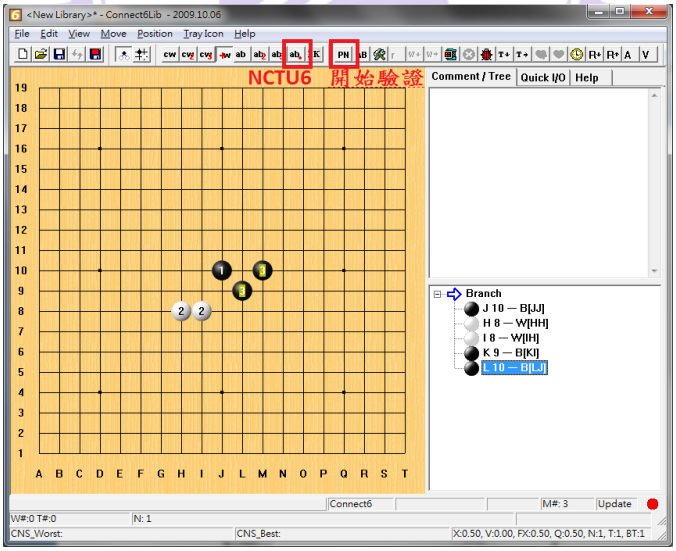


圖 8 六子棋編輯器

圖 8 為吳毅成教授研發團隊發展的六子棋編輯器，該編輯器除了提供六子棋棋譜編輯外，還提供上述提到的 NCTU6 功能與 JL-PNS 驗證棋局功能。

當需要用 NCTU6 功能時，會將目前盤面送出給 NCTU6 計算，接著 NCTU6 會將該盤面回傳給編輯器，如圖 9 所示，在實際的設置中，NCTU6 會放在其它的工作端電腦，當要用到該功能，會呼叫放有 NCTU6 的工作端電腦執行，其系統環境在下一章節中介紹。

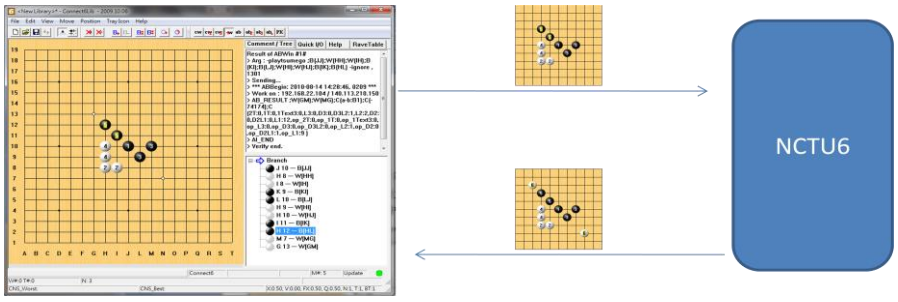


圖 9 編輯器 NCTU6 功能

2.2. JL 系統環境

JL 的概念在於將程式中常會用到的運算視為一個 Job，並將該運算抽離原程式，單獨成為另外一個運算程式，當原程式需要用到該運算時，不會在原本的程式中執行，而是呼叫該運算程式起來執行，並回收該程式運算的結果；由於將常用到的運算部份獨立出來，因此當程式需要同時執行該運算時，就可以平行化執行。

而六子棋驗證系統就是在這個概念之下，將展開節點(NCTU6)的工作視為一個 Job，因此可以同時展開多個節點。為了要完成這個目標，展開節點的程式會放在多台電腦上，我們稱這些負責運算展開節點的電腦稱為 Worker(工作端)，當需要用到展開節點的功能時，六子棋驗證系統會透過網路去呼叫工作端執行展開節點的工作。為了管理所有工作端，在六子棋驗證系統和工作端之間，會有一台管理系統，稱做 Broker，來決定與管理工作的發送與執行。

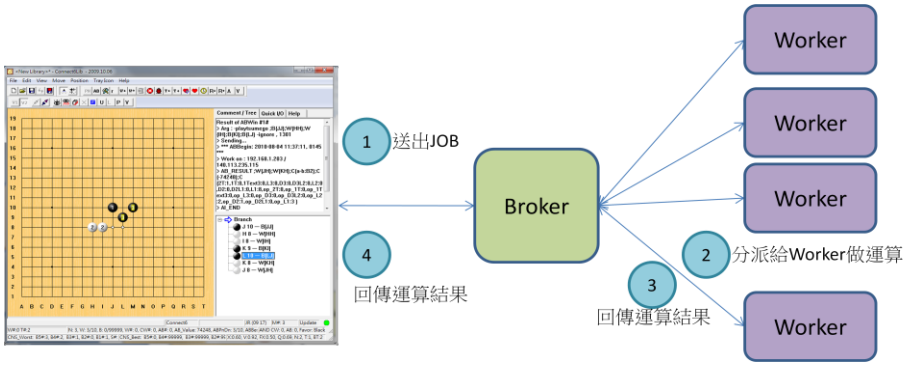


圖 10 JL 系統環境

圖 10 為 JL 系統環境架構圖[25][26]，最左邊的六子棋驗證系統代表著是應用程式。當六子棋驗證系統需要展開節點時，會送出 Job 給 Broker，Broker 會將該工作分派到一台 Worker 上面執行，接著運算完後，Worker 會將運算結果回傳給 Broker，Broker 再將結果回傳給應用程式，也就是六子棋驗證系統。

在 JL 的系統設計上，要避免將所有運算的 Job 投入到同一台 Worker 上面，也就是資源平均分配，另外當有多個使用者用此系統時，要如何將資源分配給多個使用者，也是這個系統考慮的問題之一，在此系統中，各 Job 會有優先權的概念，每位使用者依照身分，會給予不同的優先權，優先權高的 Job 在排程上會優先執行，若優先全相同，則平均分配運算資源。

2.3 證明數演算法(Proof Number Search)

證明數演算法(Proof Number Search)[8][9][10]，簡稱 PNS，是 Allis 在 1994 年提出，為一種 best first search，建構在 AND/OR 樹上，用在兩位玩家的遊戲上面，像是六子棋、五子棋、象棋、圍棋等等，其目的是在證明 true or false，像是棋局勝敗、圍棋是否吃子等問題，該演算法也實際的解決了許多棋局勝敗問題。

PNS 是個適用於證明勝敗的演算法，但是其存在著一些問題，像是記憶體的限制，因此之後有很多人提出 PNS 變形的演算法，來彌補這些不足。

為了解釋方便，以下假設 PNS 目的是證明勝敗。

2.3.1 AND/OR 樹

AND/OR 樹上的節點分為兩種，AND 節點及 OR 節點；正方型的節點代表 OR 節點，圓形的節點代表 AND 節點(如圖 7 所示)。

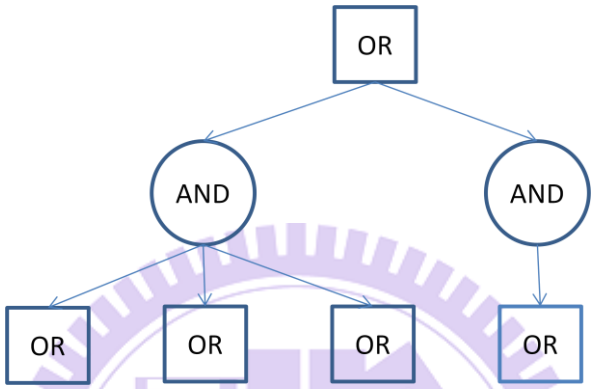


圖 11 AND/OR 樹

OR 節點要證明為必勝，必須其中一個子節點為勝；AND 節點要證明為必勝，必須所有子節點都為勝。反之，OR 節點要證明為必敗，必須所有子節點為敗；AND 節點要證明必敗，必須其中一個子節點為敗(如圖 11 所示)。

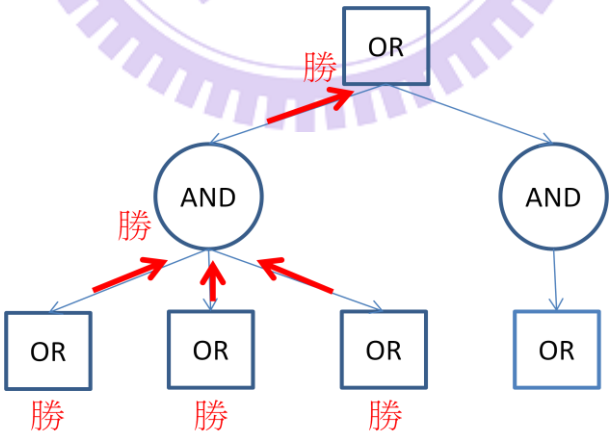


圖 12 AND/OR 樹勝敗關係

2.3.2 證明數(Proof Number)

證明數(之後簡稱 PN)代表要證明目前盤面必勝，至少還需展開多少節點(如圖 13 所示，節點左下方的數字代表 PN)。

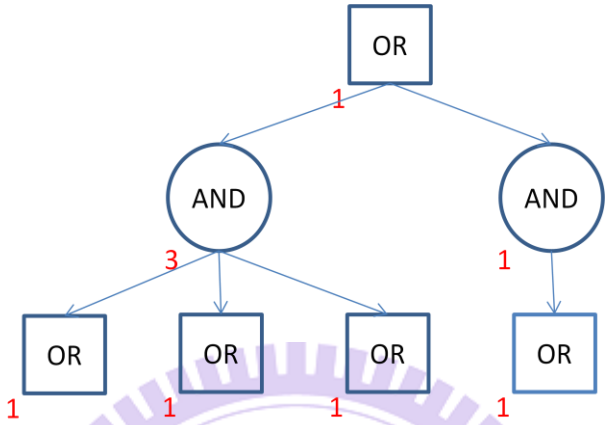


圖 13 證明數

其更新方式:

- OR 節點 : 取子節點中最小的 PN
- AND 節點 : 取子節點中 PN 的加總

2.3.3 反證數(Disproof Number)

反證數(之後簡稱 DN)代表要證明目前盤面必敗，至少還需展開多少節點(如圖 14 所示，節點又下方的數字代表 DN)。

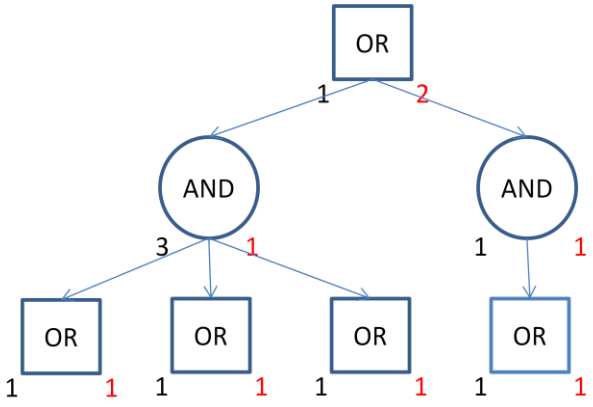


圖 14 反證數

其更新方式:

- OR 節點 : 取子節點中 DN 的加總
- AND 節點 : 取子節點中最小的 DN

2.3.4 PNS 演算法步驟

上面介紹了 PNS 建構在怎麼樣子的搜尋樹上面，並介紹節點上 PN/DN 所代表的意義，接來下要說明 PNS 搜尋的方式。

PNS 主要分為三個步驟:選點、展開、更新 PN/DN 數值。接下來依依介紹這三個步驟。

2.3.4.1 選點

決定哪一個 leaf 節點要展開。

當在 OR 節點時，要證明必勝只需一個子節點勝，就可以證明該節點必勝，然而要證明該節點敗，則需要證明所有子節點必敗；因此在 OR 節點時，會去選擇子節點中 PN 值最小的子節點，因為在證明敗的角度上，所有子節點皆須證明必敗，選哪個子節點並沒有差別。反之在 AND 節點時，由於證明必勝，則需證明所有子節點勝，證明必敗只需其中一個子節點敗即可，因此在 AND 節點時，會去選擇子節點中 DN 值最小的子節點(如圖 15)。

依上述選擇方式，會選擇到一個 leaf 節點，該 leaf 節點稱為最佳證明節點(Most Proving Node)，簡稱 MPN，接下來會對 MPN 做展開的動作。

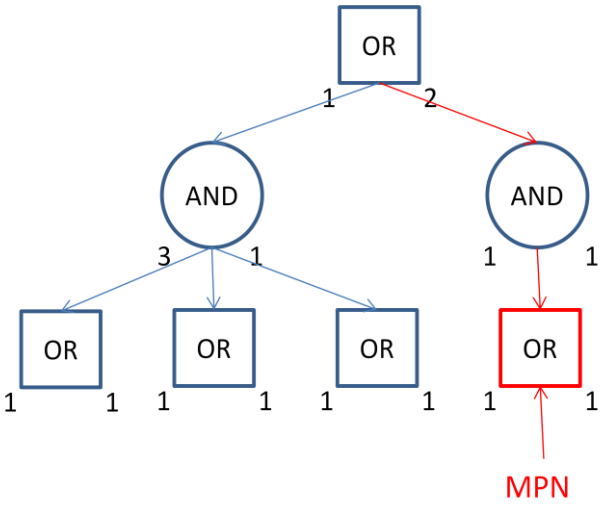


圖 15 PNS 選擇 MPN

2.3.4.2 展開

展開選定的 MPN。將該節點所有的下法產生出來。圖 15 展開如圖 16。

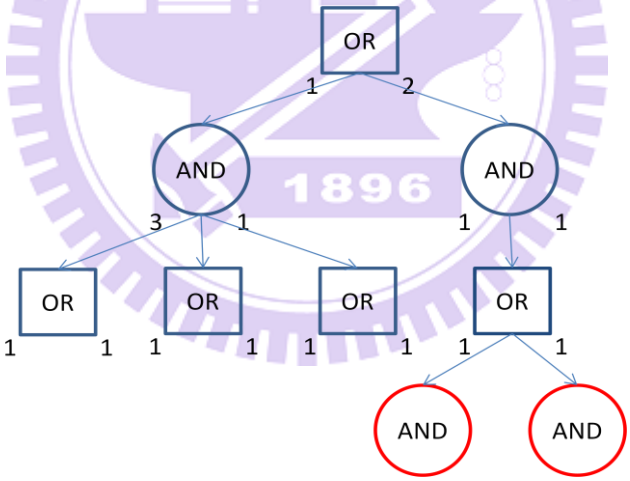


圖 16 PNS 展開節點

2.3.4.3 更新 PN/DN 數值

首先要先決定新產生出來節點的 PN/DN 數值，主要分為三種狀況，必勝、必敗、無勝敗；在必勝的情況下，則令 PN/DN 為 0/∞ ；在必敗的情況下，則令 PN/DN 為∞/0；無勝敗的情況下，則令 PN/DN 為 1/1。接著在依照 2.3.2 中的規

則，更新目前 PNS 搜尋樹上的 PN/DN 值。

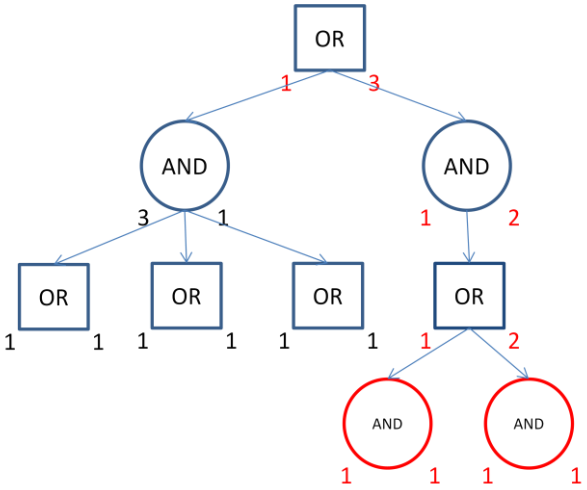


圖 17 PNS 更新 PN/DN 數值

2.3.4 PNS 的缺點

本章介紹 PNS 演算法兩個常見的缺陷，記憶體限制和重複盤面問題。

2.3.4.1 PNS 記憶體限制

PNS 為一種 Best First Search，會去往最有機會證明結束的分支去搜尋，其會將整顆搜尋樹儲存在記憶體裡面，以尋找最佳的節點，相較於 Depth First Search，只會儲存目前搜尋節點走的路徑上的節點，記憶體的用量相對大非常多。

為了彌補 PNS 記憶體用量大的不足，有許多改良自 PNS 的演算法，像是 PN^2，PDS、Df-PN 接是其變型[11][12][13][19][20][23][24]。

PN^2 為一種兩階層式的 PNS，演算法如同 PNS 一樣，差別在於會對新展開的節點做另一次的 PNS，來估計該節點初始的 PN/DN 值。

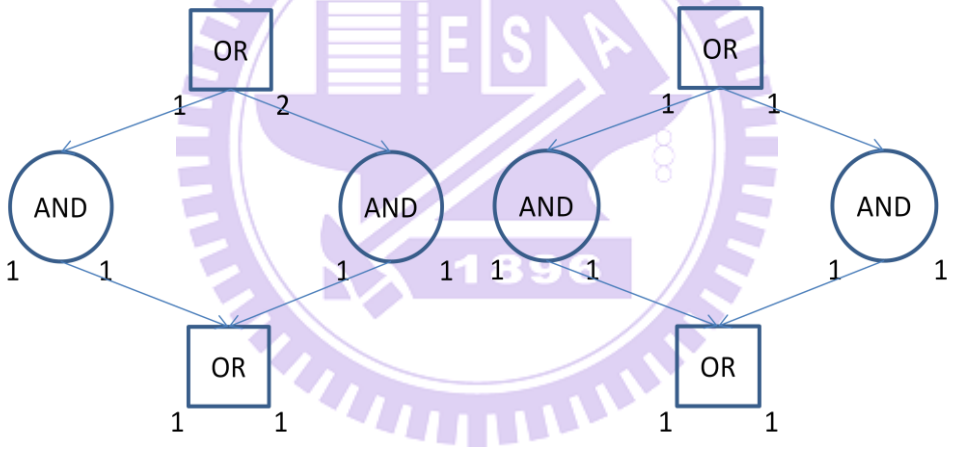
PDS、Df-PN 則是一種 Depth First 的 PNS，搜尋樹會往某一個分支挖深搜尋下去，該演算法中會有一組門檻(Threshold)，證明數門檻(Proof Number Threshold)

及反證數門檻(Disproof Number Threshold)，來決定何時停駛繼續挖深。

JL-PNS 由於每個節點都是由 NCTU6 產生，這些節點都是經由程式些做第一次篩選而產生，搜尋樹通常只會有數萬個節點，因此在 JL-PNS 沒有記憶體的問題。

2.4.3.2 PNS 重複盤面

在許多的搜尋樹中，常常會有相同盤面的問題，如圖 18 所示，而如果以上述的邏輯，更新 PN/DN 值時，就會發生數值更新錯誤，以圖 18 的例子，在根節點實際的 PN/DN 應該是 1/1，然而卻會錯誤的更新程 1/2。由於 PN/DN 值錯誤的更新，會使得在展開節點時，可能選擇到較差的節點。



根節點錯誤的 PN/DN 值　　實際 PN/DN 值

圖 18 PNS 相同盤面問題

2.4 工作層級證明數演算法

工作層級證明數演算法(Job-Level Proof Number Search)[30]，簡稱 JL-PNS，由交通大學吳毅成教授研究團隊於 2010 年提出。是一種平行化的 PNS 演算法[21]。該演算法中將展開節點視為一個 Job，不同於 PNS 演算法，展開節點工作(Job)

是交由其它程式來運算，因此稱為 JL。具有以下優點:

- 選擇 MPN 與展開節點可以獨立進行。
- 可同時展開多個節點。
- 降低 PNS 記憶體限制的問題

2.4.1 選擇多組 MPN

如同 PNS 演算法，JL-PNS 會選定 MPN，接著展開該節點，不同在於 JL-PNS 會選擇多組 MPN(圖 19)，而由於展開節點是交由其它程式來運算，因此可以同時展開多個節點。

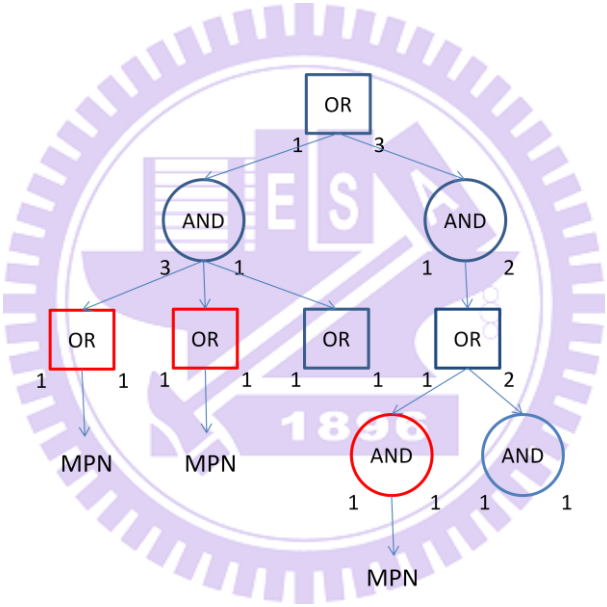


圖 19 選定多組 MPN

在選擇 MPN 上，依照原本 PNS 演算法選點方式，總是會選擇到同一個節點，因此在 JL-PNS 演算法中，會先假設選定到的 MPN 會 Loss，更新目前的搜尋樹，然後就可以找其它的 MPN。這種令 MPN 為 Loss 的方式，在該演算法中稱為虛擬敗(Virtual Loss)。

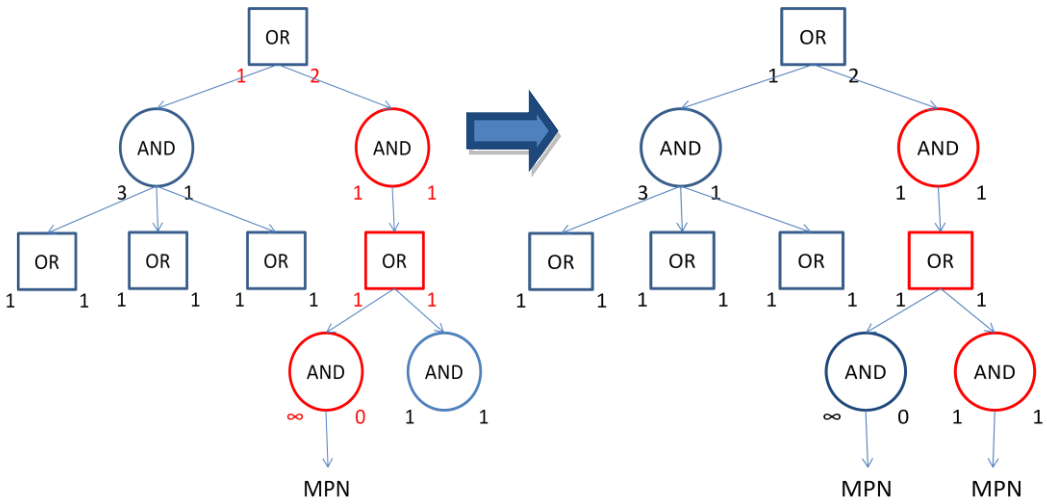


圖 20 虛擬敗(Virtual Loss)

在實際的做法中，選定的 MPN 不一定要設定成 Loss，可以有很多種設定方法，像是設定成 Win，在 JL-PNS 論文中有比較多種機制，但是各種做法的結果沒有明顯的差異，該篇論文最後選定 Virtual Loss 的機制來選擇 MPN。

2.4.2 節點初始化

在 PNS 演算法中，新展開的節點 PN/DN 值設定成 1/1，在 JL-PNS 中，所有展開的節點，皆是交由 NCTU6 程式來運算，程式會回傳該著手的棋局狀態，因此可以利用該著手的棋局狀態來初始化 leaf 節點的 PN/DN 值。

下列初始化的參數為 JL-PNS 測定的最佳參數。

狀態	PN/DN(根節點為黑)	PN/DN(根節點為白)
B:w	0/∞	∞/0
B:a_w	1/18	18/1
B3	2/12	12/2
B2	3/8	8/3
B1	4/6	6/4
Stable	6/6	6/6
Unstable	5/5	5/5
Unstable2	4/4	4/4
W1	6/4	4/6
W2	8/3	3/8
W3	12/2	2/12
W:a_w	18/1	1/18
W:w	∞/0	0/∞

表格 3 JL-PNS 初始化

2.5 謀反數搜尋演算法

謀反數搜尋演算法(Conspiracy Number Search)，簡稱 CNS。McAllester 在 1988 年發表了一篇論文[15][16][18][22]，其中提出了一種 Conspiracy Number Search(CNS)，該方法是在搜尋目前盤面的穩定狀態。在本篇論文研究中，希望能找出一種方法，能選出適當的下法，並又不失去原本驗證盤面的功能，而 CNS 演算法又與 PNS 有許多相似之處，因此在之後的改進本篇論文加入了 CNS 的概念。

2.5.1 Min-Max 樹

圖 21 為一個 Min-Max 樹，每個 Leaf 節點上會有該節點該盤面的狀態，裡面狀態的意義，如同 NCTU6 中的棋局狀態，為了方便解釋，我們假設目前 OR 節點為該黑方下子，AND 節點為該白方下子，因此內部節點的棋局狀態更新方式，OR 節點會去選擇子節點中棋局狀態對黑最好的做更新，AND 節點會去選擇子節點中棋局狀態對白最好的最更新。以節點 B 為例，由於接下來該白下，選擇 D 節點的下法，會使棋局狀態變成 W1，選擇節點 E 的下法，會使棋局狀態變成 W2，因此 E 節點的下法對白比較有優勢，因此 B 節點狀態就會取 E 來更新；節點 A 該黑下，因此在節點更新分數時，會取選擇所有下法中對黑最有利的，當選擇 B 節點下法時，會變成 W2 的棋局狀態，選擇 C 節點下法時，會變成 W1 的棋局狀態，因此 A 節點一定會去選擇 C 節點的下法，因此在更新棋局狀態時，會取 C 節點的 W1 更新。

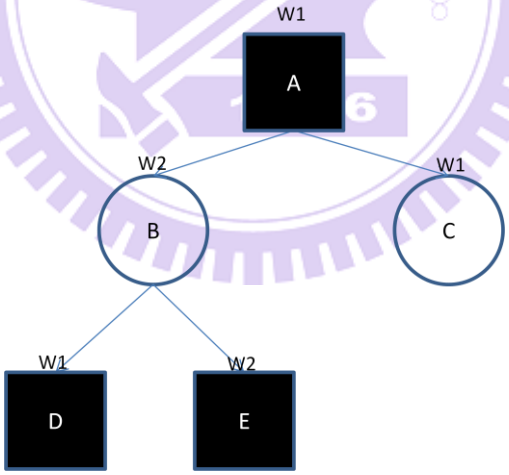


圖 21 Min-Max 樹

2.5.2 謀反數(Conspiracy Number)

謀反數(簡稱 CN)代表在目前的搜尋樹上，節點要變到其它節點狀態時，至少需要展開多少個 leaf 節點。表格 4 為圖 21 節點 A 的 CN，假設目前總共有六

種分數的狀態(B:w、B2、B1、W1、W2、W:w)；當要證明黑必勝(B:w)時，如同上面 PN 的概念，由於節點 A 是 OR 節點，因此只要一個子節點黑必勝，就算必勝，當節點 C 為黑必勝時，節點 A 就是黑必勝，所以 A 節點 B:w 狀態的 CN 值為 1；當要證明是黑必敗(W:w)時，必須所有下法皆是黑必敗，也就是要去證明 B、C 節點為黑必敗，而節點 C 要是黑必敗，則 D、E 兩個節點其中一個必需是黑必敗，因此 A 節點 W:w 的 CN 為 2，要證明黑必敗需 C、D(或是 C、E)兩個節點為黑必敗。

Grade	**CN**	**改變的節點**
B:w	1	C
B2	1	C
B1	1	
W1	0	C
W2	1	C
W:w	2	(C,D)或(C,E)

表格 4 圖 21 節點 A 之 Conspiracy Number

而搜尋樹的各節點狀態之 CN 值表示如圖 22 所示，其中 B:w 狀態之 CN 可等同於目前 PN 值，W:w 狀態的 CN 值等同於目前 DN 值。

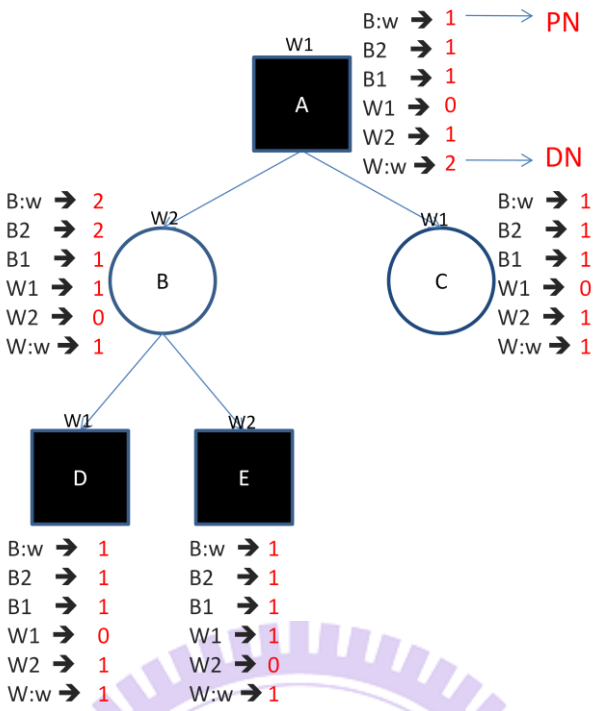


圖 22 各節點之 CN

2.5.3 Conspiracy Number Search(CNS)

如同 PNS 一樣，CNS 分為三個步驟:選點、展開、更新 CN 數值，以下分別解釋這三個部分，另外由於 CNS 選點方式比較不同，為了解釋方便，選點有一部分會拿到這三個部分最後再說明。

2.5.3.1 CNS-選點

從上面的例子中，可以知道 CN 與 PN/DN 的概念很相似，只是除了記錄節點的 PN、DN 值之外，PN、DN 之間很多種節點狀態(e.g. B2、B1…)，也會有一個值來記錄到達該狀態，至少還要展開多少節點。

PNS 在選點方時，是依照節點上面的 PN/DN 值來選擇，而 CNS 是依照節點中的 CN 值來選節點，然而每個節點上有很多組 CN 值，因此在選點一開始時，該演算法會先決定要參考哪個分數的 CN 來選點，而在選擇過程中，如同 PNS

一樣，會去選擇數值小的節點，CNS 會去選擇 CN 小的節點。

以圖 17 為例，假設目前參考 B2 狀態的 CN 來選點，節點 A 在選點時，由於節點 C 上 B2 狀態的 CN 比節點 B 上 B2 狀態的 CN 還小，因此會選擇節點 C，這是因為要變到狀態 B2，節點 C 只需改變其搜尋樹底下一個節點，目前棋局狀態就可以變成 B2，而節點 B 則需改變其搜尋樹底下兩個節點，目前棋局狀態才可以變成 B2，因此選擇節點 C 要達成的機會比較大。

然而若選擇參考 W:w 的 CN，節點 A 要改變到 W:w，必須節點 B 及節點 C 的棋局狀態都改變到 W:w，節點 A 選擇誰都沒有差，因此當參考的棋局狀態，比目前節點本身棋局狀態還差時，選點方面則是任意選擇；舉例來說，圖 22 中節點 A 選點時，若參考的棋局狀態是 W:w，則要變到這個狀態，必需所有子節點都證明是 W:w，對 A 節點來說，選哪個節點都沒有影響，因此節點 A 可能選到節點 B 或節點 C；當節點 B 選點時，參考的棋局狀態是 B2，則對節點 B 而言，節點狀態要變成 B2，必需所有子節點狀態都變成 B2，選哪個節點對 B 都沒有影響，因此在參考 B2 的節點狀態下，B 節點有可能會選擇 D 或 E 節點。

至於如何決定選點時參考哪個棋局狀態，會在之後的章節中說明。

2.5.3.2 CNS-展開

展開所有選定節點的下法，並計算新展開節點的分數。圖 23 為假設選到節點 C 作展開。

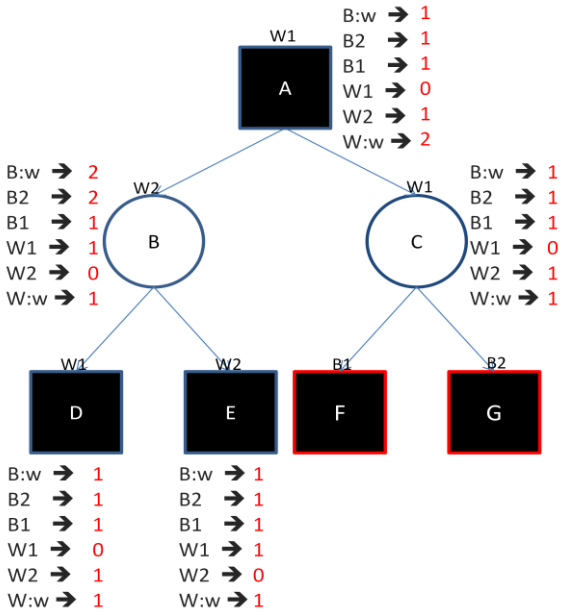


圖 23 CNS 展開節點 C

2.5.3.3 更新 CN

初始化新節點的 CN，接著依照上述規則，更新搜尋樹上結點的 CN。節點 CN 初始化方式為，在節點沒有勝敗的情況下，該節點目前棋局狀態對應的 CN 為 0，其它棋局狀態的 CN 為 1；以節點 F 為例(圖 24)，該節點目前分數為 S4，因此 S4 對應到的 CN 值為 0，其它分數對應到的 CN 為 1。若該節點棋局狀態為黑必勝或是黑必敗(W:w)時，則目前狀態對應的 CN 為 0，而其它棋局狀態對應 CN 值則設定成無窮大。

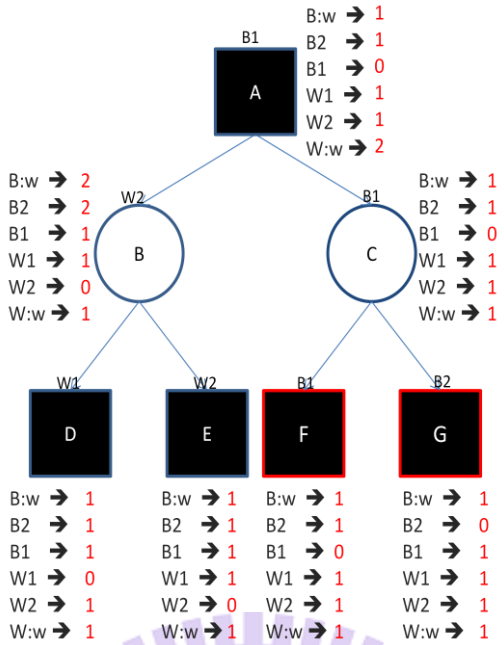


圖 24 更新 CN

2.5.3.4 決定參考的 CN

在 PNS 中，是參考 PN/DN 值選點，但是當目前盤面難以證明勝敗時，在選點方面就會有些問題，如圖 25，目前 A 節點的 PN 值是 100，可以看出要證明黑必勝上，已經有相當的難度，因此 CNS 概念在於，既然我們原本想要證明的狀態，已經難以證明，那就退而求其次，選擇次好的狀態，以圖 25 為例，既然要證明黑必勝已經不容易了，但是要證明 B2(目前的 CN 值為 15)，似乎還有機會，所以 CNS 在選點上就有可能會去參考 B2 來選點。

在 CNS 選點中，會有一個謀反數門檻(Conspiracy Number Threshold)，簡稱 CNT，代表當棋局狀態的 CN 超過該 CNT 時，我們就認為到達該狀態是不太可能，進而退而求其次，去選擇在 CNT 範圍內，次好狀態。

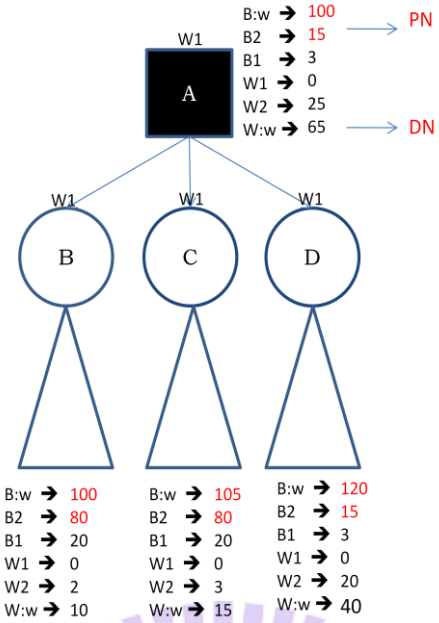


圖 25 PN、DN 過大時選點問題

而在選定參考的棋局狀態，是參考根節點的 CN 來決定，我們令 Vroot 為根節點的棋局狀態，Vmax 為沒有超過 CNT，對根節點最好的棋局狀態，Vmin 為沒有超過 CNT，對根節點最差的棋局狀態；以圖 26 為例，當 CNT 為 100 時，根節點 A 的棋局狀態為 W1，Vroot 為 W1，CN 未超過 CNT 的棋局狀態有 B2、B1、W1、W2、W:w，五個狀態之中，對 A 節點最有利的狀態是 B2(A 節點該黑下)，最沒有利的狀態是 W:w，所以 Vmax 為 B2，Vmin 為 W:w。

CNS 在選點時參考的棋局狀態，就是 Vmax 或是 Vmin 指向的棋局狀態，然而 CNS 在選點時，只參考一個棋局狀態來選點，因此 Vmax、Vmin 的選擇上面，會選擇離 Vroot 較遠的棋局狀態，來當作該次選點依據，圖 26 中 Vmax 以及 Vmin 距離 Vroot 都差兩個棋局狀態，因此在選擇上面就是任意選。

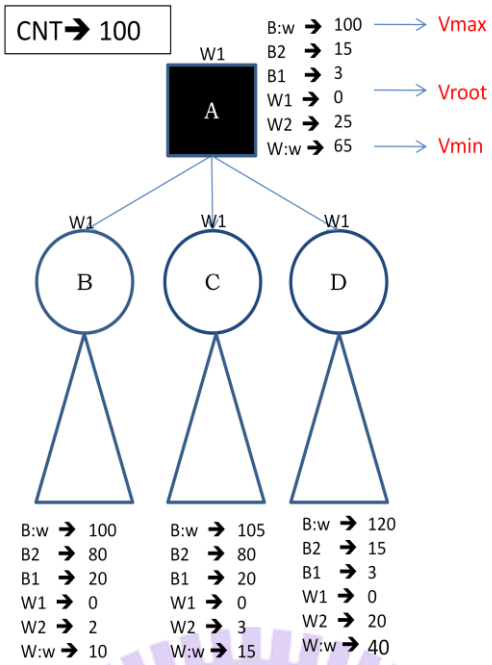


圖 26 Vroot,Vmax,Vmin

2.5.3.5 CNS 結束條件

不同於 PNS，結束條件是證明出勝敗，CNS 的結束條件是，根節點除了本身的棋局狀態，其它棋局狀態的 CN 皆超過 CNT 時，又搜尋結束，也意味著尋找出目前盤面的穩定棋局狀態，圖 27 為 CNT=100 時，結束搜尋的例子。

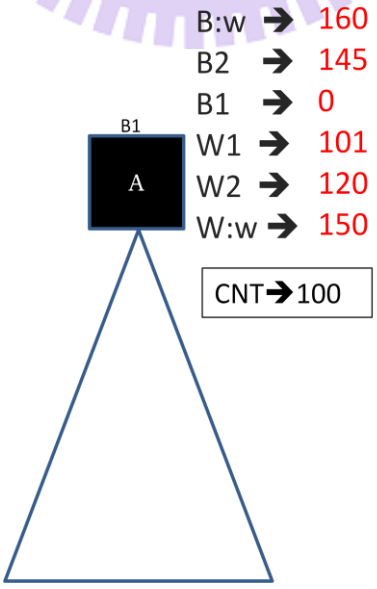


圖 27 CNS 結束搜尋

第三章、研究方法

在這章節中提出兩種改進的方式，第一種在 3.1 章節中介紹，該方法是觀察了工作層級證明數搜尋演算法(Job-Level Proof Number Search，簡稱 JL-PNS)在六子棋遊戲上，選點展開時的缺點，並提出改善機制，第二種提出工作層級謀反數搜尋演算法，在 3.3 章節中介紹。

3.1 K-Band

在這邊我們提出一種新的更新 PN/DN 值的方法，稱為 K-Band，我們將這個方法實際實做在 JL-PNS 上面。簡單來說，K-Band 的機制，就是做多取 K 個 PN/DN 值來做更新。在 3.1.1 章節中先介紹 K-Band 更新方式，3.1.2 會比較 K-Band 與動態加寬(Dynamic Widenings，簡稱 DW)差別，3.1.3 中會說明 K-Band 的機制同樣可以運用到謀反數搜尋演算法(Conspiracy Number Search，簡稱 CNS)上面。

3.1.1 節點更新問題

在六子棋遊戲中，有一種棋型稱為迫著(Threat)，當出現迫著時，防守方必需下子去抵擋迫著，否則攻擊方就會獲勝，如圖 28 所示，當黑選擇左邊盤面的下法，黑子會連成四顆(紅線上的四顆子)，此時防守方的白子，必須花兩顆子去抵擋黑方，否則黑方就會獲勝(連成六子)；這種花兩顆子才能抵擋的迫著稱為雙迫著，同理，花一顆子可以抵擋的迫著稱為單迫著；下出雙迫著時，儘管防守方能下的著手很少，但是不一定對攻擊方有利，如圖 28，左邊的盤面黑下出雙迫著，可是白的三種下法對雙方的都是均勢，而右邊黑的下法，既限制白的發展，又增加自己的發展性，儘管接下來白有很多可能的下法，但是其實都是對黑有利。

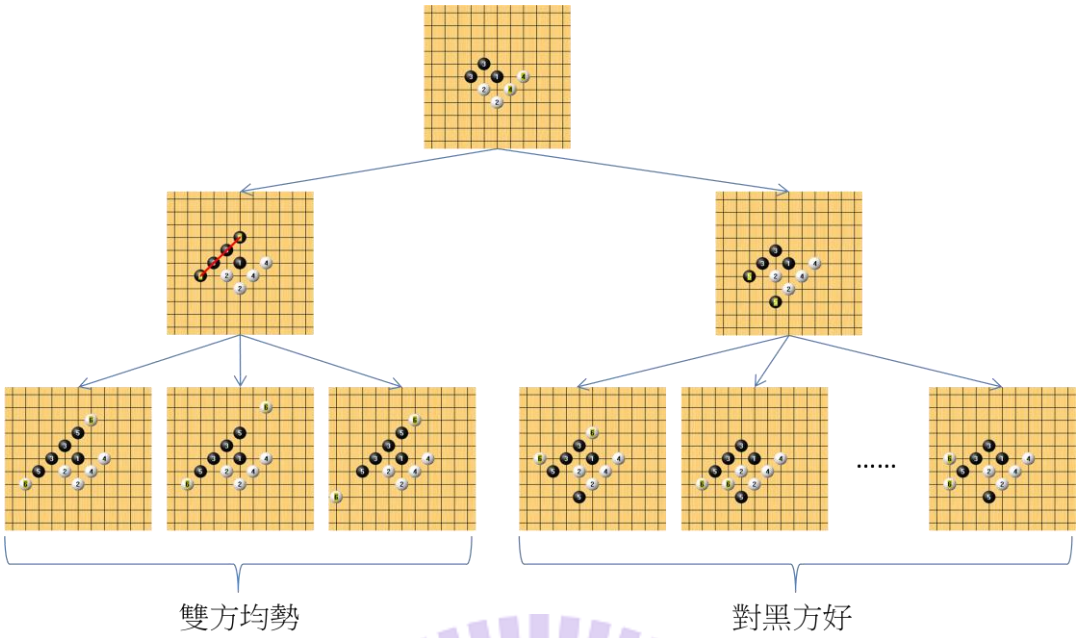


圖 28 節點更新問題-1

圖 29 將圖 28 表示成 AND/OR 樹，並加上 PN/DN 值，OR 節點黑色代表目前該黑下，AND 節點白色代表該白方下，由上述分析，我們知道往左方盤面下，是對雙方均勢，可是往右方盤面下，是對黑方好，在 JL-PNS 展開節點時，我們會希望往右方盤面做展開，可是由於雙迫著的關係，使得左方盤面白的下法較少，在 PN/DN 值更新時，反而會使 JL-PNS 演算法誤以為左邊比較容易證明黑必勝，在選點的時候會往左方盤面選擇。

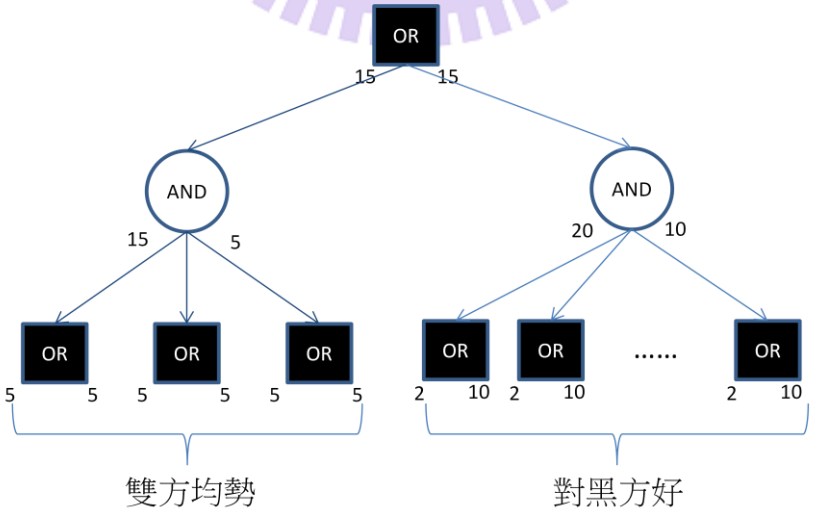


圖 29 節點更新問題-2

為了解決這個問題，我們決定在更新 PN/DN 上做改善，不同於以往證明數

搜尋演算法(Proof Number Search，簡稱 PNS)的更新方式，我們在更新 PN/DN 值時，只採取子節點之中，最具象徵性的幾個節點做更新，而不參考所有子節點做更新。

3.1.2 動態加寬

在提出我們的方法之前，我們研究了改變節點更新方式相關的論文。Yoshizoe 在 1988 年發表了一篇論文[33]，其中提出了一種在 PNS 中，更新 PN/DN 的新方式。

3.1.2.1 更新節點方式

不同於 PNS 的節點更新，會依據所有子節點的 PN/DN 值，在 DW 的方式中，不會參考所有子節點更新 PN/DN 值，而是參考其中特定幾個子節來更新 PN/DN 值。

在該策略中，會先排序所有子節點(圖 30)；OR 節點的排序子節點方式，依據子節點 PN 的值由小到大排序，AND 節點的排序子節點方式，則是依據子節點 DN 的值由小到大排序。

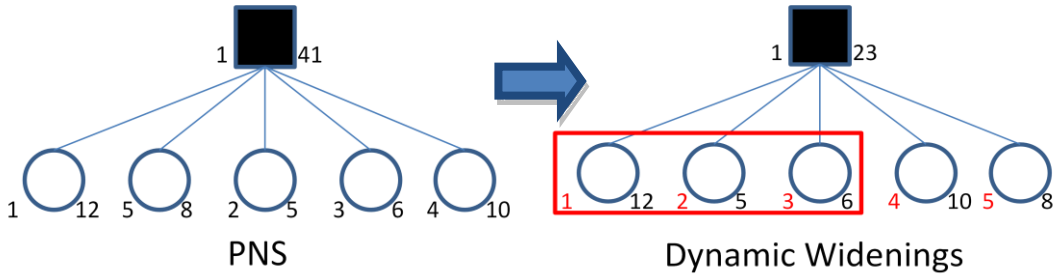


圖 30 OR 節點排序子節點

其更新節點的方式如同 PNS 一樣，只是取前面 K 個子節點做更新。OR 節點 PN 值取子節點中最小的 PN 值，DN 取排序子節點過後，前 K 個子節點加總；AND 節點 PN 值取子節點中，前 K 個子節點加總，DN 取子節點中最小的 DN。

3.1.2.2 DW 之誤判

在某狀態下，DW 的策略，會取到不是最有代表性的節點做更新，如圖 31，要證明黑必勝，左子樹實際要至少展開 53 個盤面，而右子樹只要 10 個，然而在 DW 取三個節點更新的機制下，圖中左子樹更新 PN 時，由於取到的 PN 為所有子節點中最小的 3 個，該 AND 節點的 PN 值相對就小很多，因此會造成更上去的節點誤判以為往左邊展開，可以較快證明黑必勝。

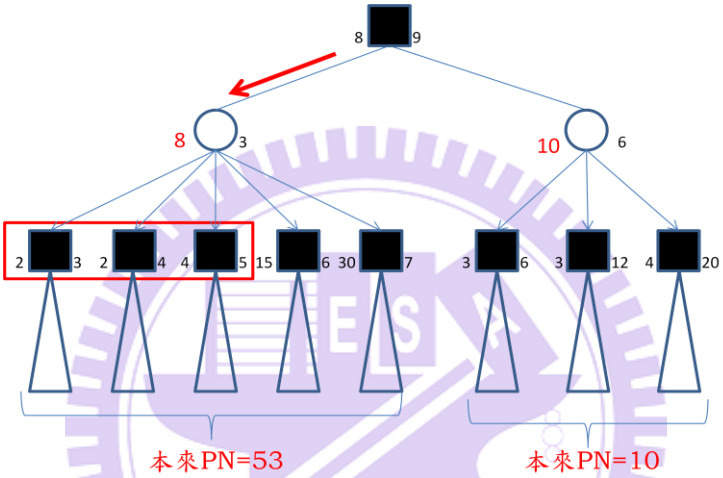


圖 31 DW 誤判

3.1.1 K-Band 更新

K-Band 為一種更新 PN/DN 的機制，方式是取前 K 個 PN/DN 值來更新。在 OR 節點時，PN 同樣是取子節點中最小的 PN 值更新，而 DN 則是取子節點中最大的前 K 個 DN 加總。如圖 32，在 2-Band 的情況下，OR 節點 PN 值取子節點中最小的值五 5 更新，而 DN 取子節點中 DN 最大的兩個加總(10+10)更新。

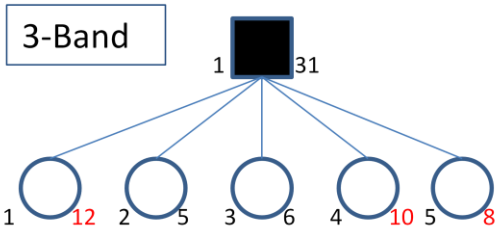


圖 32 3-Band OR 節點更新

反之，在 AND 節點時，PN 取子節點中最大的前 K 個 PN 加總，DN 取子節中最小的 DN，如圖 33。

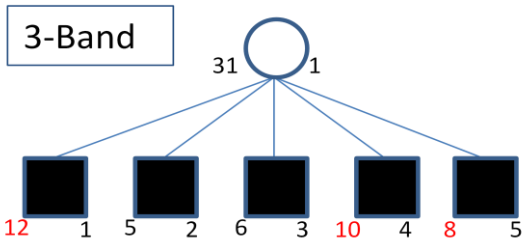


圖 33 3-Band AND 節點更新

3.1.2 比較 K-Band 與 DW

K-Band 與 DW 都是用特定幾個節點來更新 PN/DN 值，然而 K-Band 機制確是比較能反應目前節點的 PN/DN 值，原因在於 K-Band 在更新時 PN/DN 時都是取最好的 K 個 PN/DN 值。圖 34 為圖 31 搜尋樹，採用 3-Band 機制下 PN/DN 的數值，可以發現相較於 DW 策略，K-Band 的機制可以反映出左子樹要證明黑必勝，要來的比右子樹證明黑必勝來的難。

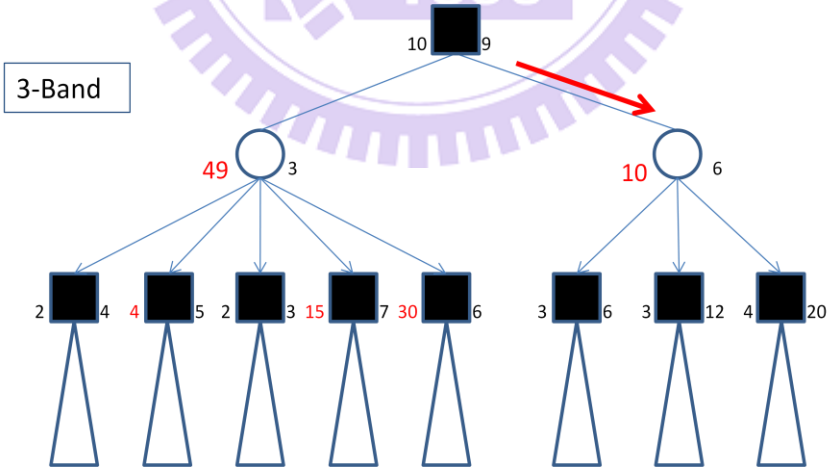


圖 34 3-Band 選點

3.1.3 CNS 採用 K-Band

CNS 中 CN 的更新同樣可以使用 K-Band 的機制，更新方式與 PNS 類似，同樣是取最好的 K 個 CN 加總，如圖 35 中 OR 節點 W:w 的 CN 原本為所有子節點 S1 的 CN 加總(5+4+6=15)，在 2-Band 的 case 則是取最大的兩個相加(5+6=11)；棋局狀態 W2 的 CN 值為子節點中 W2 最大的兩個 CN 加總(2+3=5)。

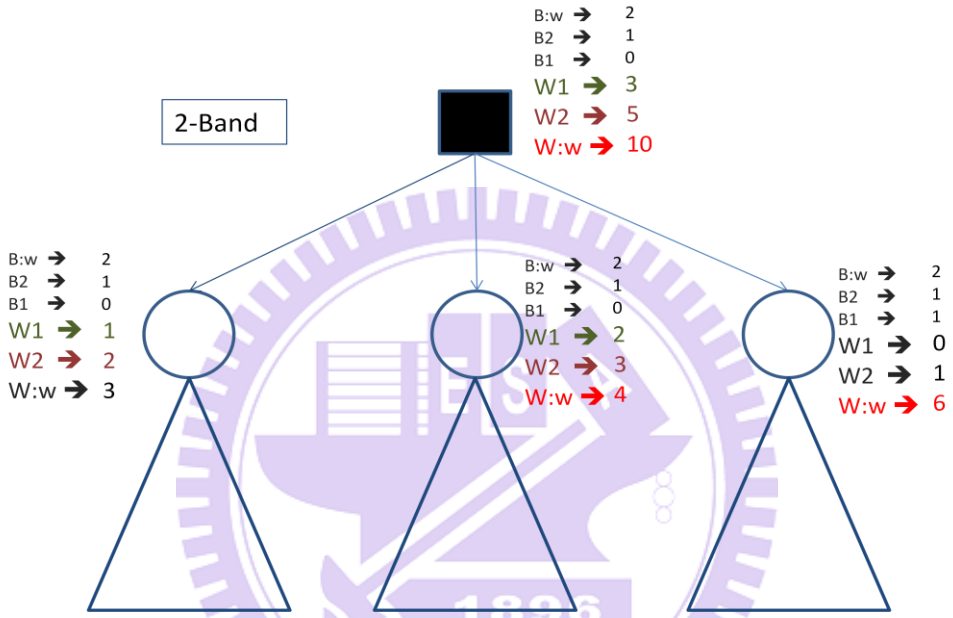


圖 35 2-Band CNS

3.2 分析 PNS 及 CNS

CNS 與 PNS 雖然最初的使用目的不同，但是其演算法卻有很高的相似度，像是同樣都需記錄所有子節點，同樣都是 Best First Search，演算法都分成選擇節點、展開節點、更新數值三部分；底下說明 CNS 在 CNT 為無窮大時，在做稍微改善，就可以變成 PNS。

3.2.1 分析 CNT 為無窮大時

在前面的章節介紹中，提到 CN 中也有包含 PN、DN，在 PNS 中，選擇哪個節點展開是參考 PN、DN，在 CNS 中，當 CNT 設定無窮大時，則 CNS 中選點參考棋局狀態的 CN，就是參考必勝或是必敗狀態來選點，也就是參考 PN、DN 來選點。

儘管 PNS 及 CNS(CNT 無窮大時)在選點時同樣參考 PN、DN 選點，選點的策略也都是選擇節點中 PN 或 DN 較小的節點，但是同一個搜尋樹上，用 PNS、CNS 兩種演算法，所選擇展開的節點(MPN)，卻是不一定相同，原因在於 CNS 上只參考一個棋局狀態來選擇。

3.2.2 改善 CNS 選點

CNS 與 PNS 在選點方面有一點很不相同，CNS 在選點的時候，只會參考一個數值(CN)來選點，而 PNS 是參考兩個數值(PN、DN)來選點；當 CNT 設定成無窮大時，若選點時是參考 PN，則就是在 OR 節點時，選擇節點中最小 PN 的節點，而在 AND 節點時，就隨機選；若選點時是參考 DN，則就是在 AND 節點時，選擇節點中最小 DN 的節點，而在 OR 節點時，就隨機選。

因此我們加了一些 CNS 的改善，令其在 OR 節點時，會參考 Vman 的 CN 來選點，在 AND 節點時，會參考 Vmin 的 CN 來選點。

有了這項改善，在令 CNT 為無窮大，則 CNS 就等同於 PNS；選點時在 OR 節點會挑選子節點中 PN 最小的，AND 節點會挑選子節點中 DN 最小者。

3.3 工作層級謀反數搜尋演算法 (Job-Level Conspiracy Number Search)

在這章節中提出工作層級謀反數搜尋演算法(Job-Level Conspiracy Number Search，簡稱 JL-CNS)演算法，如同 JL-PNS 一樣，展開節點交由其它程式計算、可以同時展開多個 MPN 的特性，差別在於 JL-CNS 在選點時參考 CN 來選點。

3.3.1 建構 CN

在 JL 的系統上，展開節點的工作是交由 NCTU6 運算，每個節點都會有個棋局狀態，在 JL-CNS 如同 CNS 一樣會為每個棋局狀態記錄一個 CN 值。

3.3.1.1 利用棋局狀態建構 CN

前面介紹過 NCTU6 總共有 13 種棋局狀態，這些棋局狀態可以分做 3 類；第一類對黑有利的狀態，分別是 B:w、B:a_w、B3、B2、B1，B:w 代表黑必勝，其次依照對黑方有利程度大到小排序，是 B:a_w、B3、B2、B1；第二類是對白有利的狀態，依照對白方有利程度大到小排序，分別是 W:w、W:a_w、W3、W2、W1，其中 W:w 代表是白必勝；第三類是局勢不明朗，分別是 stable、unstable、unstable2。

首先我們先將所有棋局狀態，依照對攻擊方(根節點下子方)有利的程度，由大到小做排序，假設 OR 節點該黑下，則棋局狀態對黑方有利程度，由大到小設定成 B:w、B:a_w、B3、B2、B1、Stable、W1、W2、W3、W:a_w、W:w，由於 Stable 代表狀態穩定，對雙方均勢，因此可以想做分數是在中間，然而有兩個棋局狀態 Unstable 及 Unstable2 就比較難區分其對雙方有利程度，因為這兩個狀態代表分數難以判定。最終我們的做法是，將這兩個狀態視為 Stable，不同的地方

在於，節點初始化的時候，Stable、Unstable、Unstable2 會給予不同的初始值。

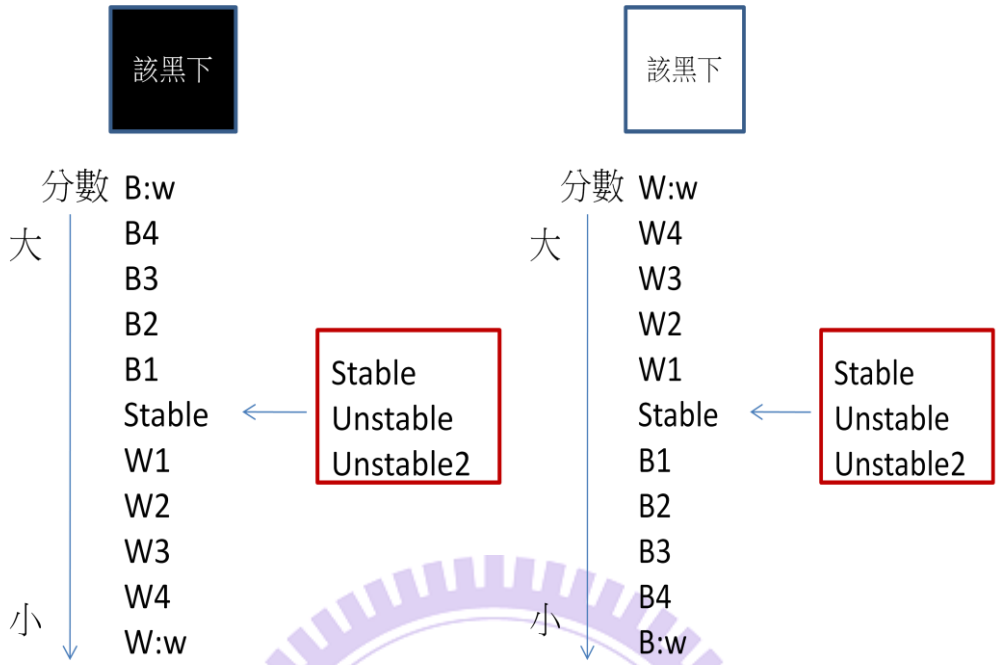


圖 36 OR 節點之 CN

3.3.2 選擇多個 MPN

如同 JL-PNS 一樣，會選擇多個 MPN 做展開，JL-CNS 的概念也是一樣，會去選擇多個 MPN 做展開，只是選擇 MPN 的方式是參考 CN。

JL-CNS 選點的方式，不同於以往 CNS 只參考一種 CN 來選擇，而是採用 3.2.2 章節中提出的改善機制來選點，也就是在 OR 節點參考 Vmax，AND 節點參考 Vmin。

圖 37 為根節點該黑下，CNT 是 4 的狀況下，JL-CNS 選點例子，此時的 Vmax 為 B:w，Vmin 為 W1，節點 A 在選點時，由於 C 在 B:w 的 CN 值較小，因此會選擇節點 C，而節點 C 在選點時，會參考棋局狀態 W1 的 CN，因此會選到 W1 CN 較小的 F 節點，F 節點就成為 MPN。

如同 JL-PNS 選點時虛擬敗(Virtual Loss)的概念，我們設定目前選的 MPN 為 Loss，圖 38 中選定的第一個 MPN 為 F 節點，由於根節點是該黑下，因此將 MPN

設定成 Loss，就是設定 F 節點成 W:w，並更新搜尋樹上 CN 的數值。

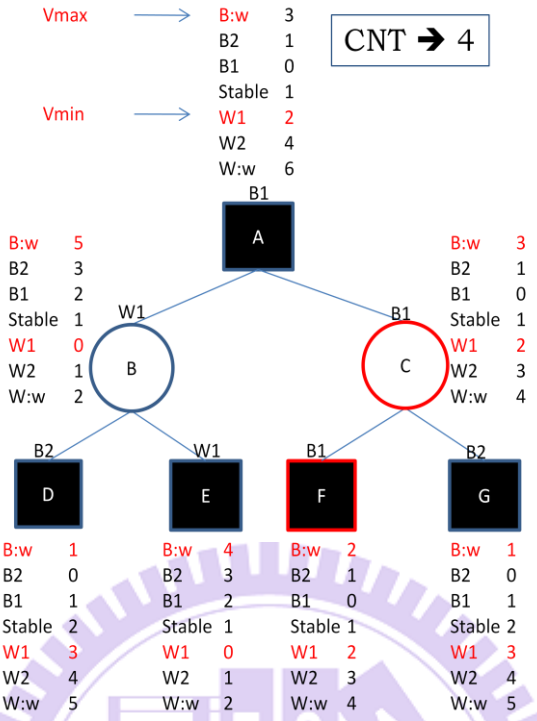


圖 37 JL-CNS 選點

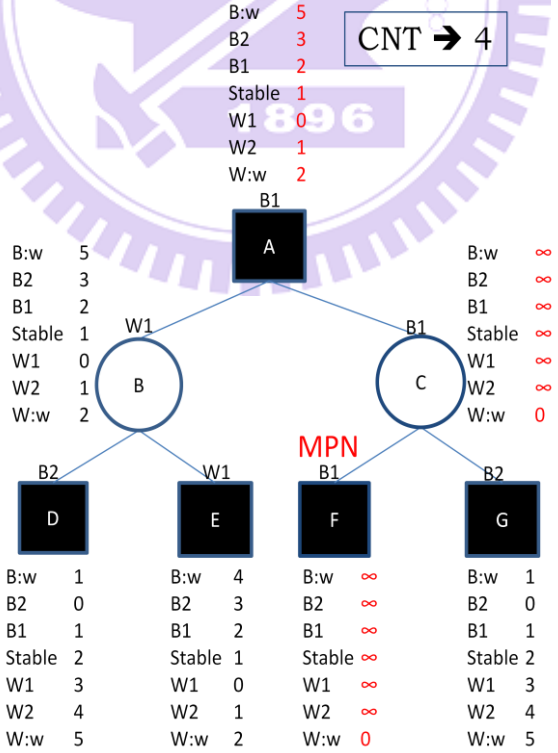


圖 38 JL-CNS 選點之 Virtual Loss

圖 39 顯示圖 38 中的搜尋樹如何選到另一個 MPN。

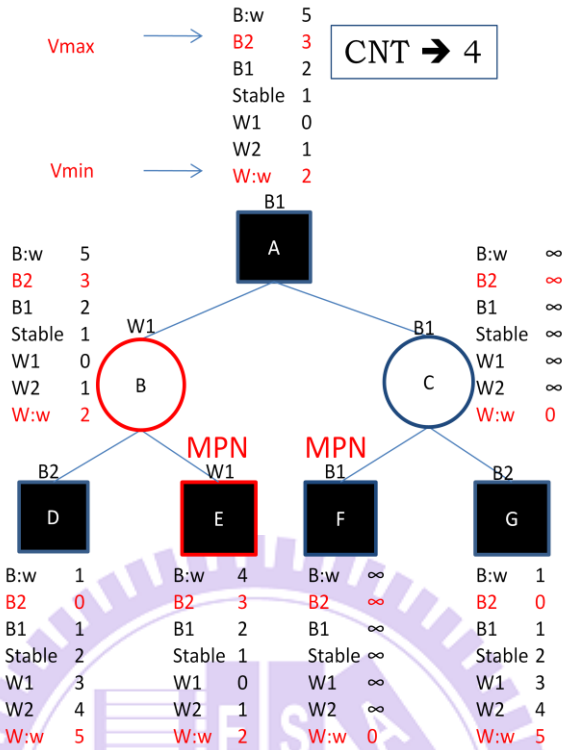


圖 39 選定多組 MPN

在實際的設置上，若目前搜尋之盤面，尚未證明出勝敗，但是 Vmax 或是 Vmin 選定的棋局狀態為根節點本身的棋局狀態時，我們會將 Vmax 指向目前棋局狀態有利一層的棋局狀態，Vmin 則會指向目前棋局狀態不利一層的棋局狀態。假設目前有七種棋局狀態如圖 39 所示，當目前根節點的狀態為 Stable，而 B1、B2、B:w 狀態的 CN 值皆超過 CNT，原本在 Vmax 選擇上面會選擇 Stable，然而當 Vmax 所代表棋局狀態為根節點目前棋局狀態時，會將 Vmax 設定成對黑更裡利的狀態，也就是 B1。

3.3.3 Conspiracy Number 初始化

原始 CNS 演算法中，會將本身分數之外的 CN 設成 1，本身分數的 CN 設定成 0，如同 JL-PNS 一樣，對於初始值的設定，我們認為離目前節點分數越遠分

數的 CN 值，相對來講應該給予較高的值，舉例來說假設目前一個新展開的節點分數是 W1，在 B:w 分數的 CN 初始值應該要比 B1 分數的 CN 要來的高，因為實際上該節點要到達 B1 狀態要來的比到 B:w 狀態還困難。

表格 5 中列出節點 CN 初始化設定(假設根節點該黑下)，每一列代表一個分數的 CN 初始化設定。該參數設定，是參考 JL-PNS 中實驗出的最佳參數做修改，當 CNT 設定無窮大時，JL-CNS 就變成如 JL-PNS，因此棋局狀態 B:w、W:w 的初始值設定，就如同 JL-PNS 中 PN、DN 初始值設定，而其它棋局狀態 CN 設定，就依照該棋局狀態離目前棋局狀態的距離來決定，以 B3 為例，其棋局狀態之 B:w 和 W:w 設定如同 JL-PNS 中 PN、DN 設定，在 B3 鄰近的棋局狀態為 B:a_w、B2，在 JL-PNS 初始設定出，B:a_w 狀態時的 PN/DN 值為 1/18，B:a_w 變到 B:w 至少展開一盤面，代表改變到鄰近棋局狀態，至少展開一個盤面，因此在 B3 棋局狀態 CN 的設定上面，變到 B:a_w 以及 B2 的 CN 值設定為 1，其餘狀態會依照相同邏輯，參考 JL-PNS 參數設定。

	B:w	B:a_w	B3	B2	B1	S	W1	W2	W3	W:a_w	W:w
B:w	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
B:a_w	1	0	1	2	3	4	6	8	10	12	18
B3	2	1	0	1	2	3	4	6	8	10	12
B2	3	2	1	0	1	2	3	4	6	8	10
B1	4	3	2	1	0	1	2	3	4	6	8

	B:w	B:a_w	B3	B2	B1	S	W1	W2	W3	W:a_w	W:w
W:w	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0
W:a_w	18	12	10	8	6	4	3	2	1	0	1
W3	12	10	8	6	4	3	2	1	0	1	2
W2	10	8	6	4	3	2	1	0	1	2	3
W1	8	6	4	3	2	1	0	1	2	3	4

	B:w	B:a_w	B3	B2	B1	S	W1	W2	W3	W:a_w	W:w
S	4	2	1	1	1	0	1	1	1	2	4
U	5	3	2	1	1	0	1	1	2	3	5
U2	6	4	3	2	1	0	1	2	3	4	6

表格 5 JL-CNS 節點 CN 初始化設定(根節點該黑下)

第四章、實驗

在 JL-PNS 論文中的實驗數據，顯示在工作層級(Job-Level，簡稱 JL)的演算法下，會有 Super-Linear 和 Sub-Linear 的現象，代表在這個演算法其實是相當不穩定的，我們很難去測定各種改善機制之間的優劣，在實驗之中，我們嘗試了很多的比較，藉由多組的數據，去比較上述演算法的優劣。

本章節會比較工作層級證明數演算法(Job-Level Proof Number Search，簡稱 JL-PNS)以及工作層級謀反數搜尋演算法(Job-Level Conspiracy Number Search，簡稱 JL-CNS)加上 K-Band 機制之間的優劣，而在比較上述演算法之前，得先選出每個演算法的最佳參數。4.1 章節介紹實驗環境，在 4.2 章節之中，會先決定 JL-CNS 的最佳謀反數門檻(Conspiracy Number Threshold，簡稱 CNT)， 4.3 章節會決定在 K-Band 機制下，K 值要取多少，4.4 章節比較 K-Band 與動態加寬(Dynamic Widenings，簡稱 DW)優劣，4.5 章節中會綜合比較所有演算法，並比較其在尋找最佳著手時的差異。

4.1 實驗環境

在 JL 系統的工作端(Worker)，使用 8 台雙核心的個人電腦，運算時脈 2.0GB，記憶體為 3.0GB，總共有 16 個運算資源可以使用。雖然在執行上述 JL 的演算法實，主要的運算時間是花在工作端，但是底下還是列出六子棋驗證系統上的電腦配備，運算時脈為 2.7GB，記憶體為 4.0GB。

在 4.2 到 4.5 章節中，決定演算法參數會彼此之間的優劣，我們會執行 15 個有勝敗的盤面(如附錄 I 所示)，這 15 個盤面為實驗一，題目 1 到題目 15，代表簡單的題目到難的題目，題目數越大，題目越難解；另外在證明題目上面皆使用 8 個工作端，並比較其解題目所花的時間。

另外在實驗二(如附錄 II 所示)中有 9 個無勝敗的盤面，對於這 9 個盤面，我

們有分析出該盤面好的下法。分析下法方面，我們是參考 Little Golem 棋類遊戲網站，統整所有的六子棋譜，去計算實驗二中 9 個盤面，接下來下法中勝率明顯高的著手。在 4.5 的比較中，會使用各演算法，去驗證該盤面，比較各演算法找到最佳著手的總結點數。

4.2 JL-CNS CNT

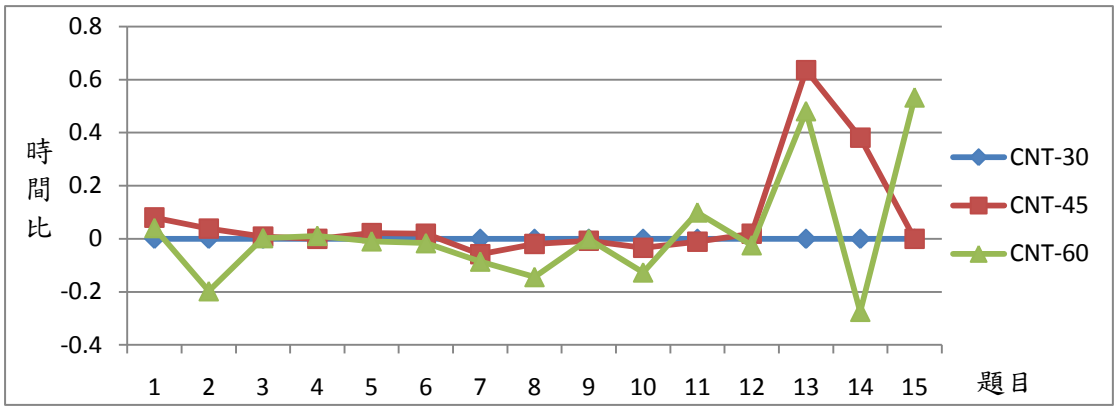
當棋局狀態的 CN 超過 CNT 時，我們就認為要到達該狀態是有難度的，進而試著去證明其它棋局狀態是否有機會到達；可以想做當 CN 超過 CNT 時，該狀態就不容易到達。

在決定 JL-CNS 之 CNT 之前，我們先使用 JL-PNS 驗證實驗一中的 15 個有勝敗盤面，並記錄必勝方的 PN 值最大到哪個值(和必敗方的 DN 最大到哪個值)，統計的結果發現最大的數值為 32，因此可以知當 PN(或 DN)是 32 時，都還有可能到達必勝(必敗)狀態。

因此在 CNT 的設置上，我們從 CNT=30 開始測試(32 附近)，接著每次增加門檻 15 往上測試，依序是 CNT=45、CNT=60 等等。在 JL-CNS 上面有兩種策略，JL-CNS1 及 JL-CNS2，在測試 CNT 時，我們以 JL-CNS1 來當作實驗依據。

4.2.1 JL-CNS CNT 實驗結果

圖 40 為 JL-CNS 的 CNT 實驗數據，橫坐標代標題目 1 到題目 15，縱座標代表是時間比。時間比公式如圖所示，$T_{CNT}(S)$代表題目 S 在 CNT 下所執行的時間，我們 CNT=30 做為基準，其它 CNT 參數都跟 CNT=30 做比較。當時間比 0 大代表該題目執行速度比 CNT=30 還要慢，比 0 小代表該題目執行速度比 CNT=30 還要快。



$$\text{時間比} = \log_2 \frac{T_{CNT}(S)}{T_{30}(S)}$$

圖 40 CNT 實驗數據

4.2.2 JL-CNS CNT 選定

從圖 40 數據可知在證明題目上面，沒有絕對的好與壞，實驗一中的 15 個題目在上述三種 CNT 下，證明速度各有好壞；另外從數據中可以發現，在難的題目驗證上，證明速度的動盪較大，因此在選擇最佳 JL-CNS 的 CNT 時，我們統計了各 CNT 在實驗一總驗證時間，如表格 6 各 CNT 驗證總時間所示，當 CNT 值越大時，其總驗證時間越長，因此在此我們選擇 CNT-30 為最好的參數。

CNT	**30**	**45**	**60**
總時間(sec)	57578	61831	62244

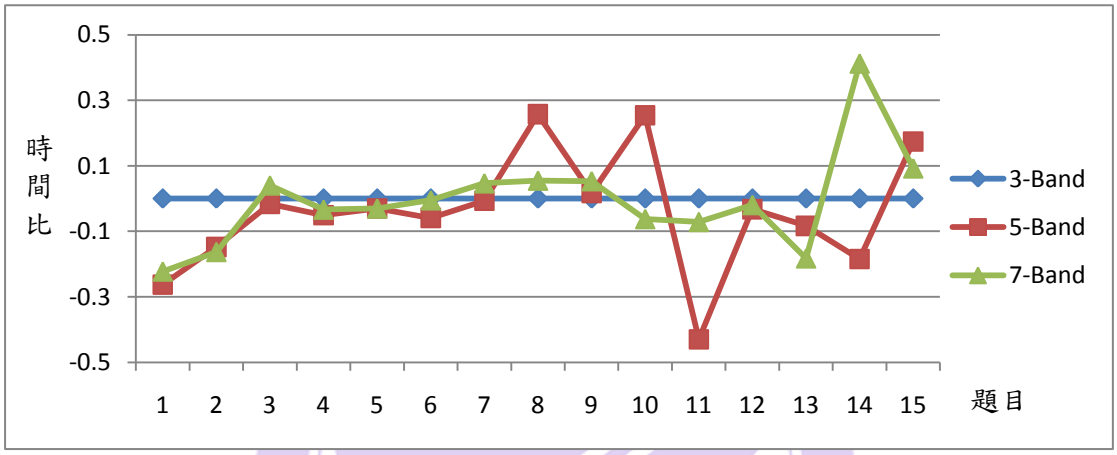
表格 6 各 CNT 驗證總時間

4.3 K-Band

這個章節我們將 JL-PNS 加上了 K-Band 機制，並測試當 K 值為 3、5、7 時，JL-PNS 驗證實驗一所需的時間。

4.3.1 K-Band 實驗結果

圖 41 為 K-Band 實驗結果，橫坐標代表題目，縱座標代表時間比；時間比的公式如圖 41 所示，$T_{K\text{-}Band}(S)$代表在題目 S 在 K-Band 機制下驗證所花的時間，在實驗結果中，我們 3-Band 為比較基準，當時間比高於 3-Band 代表驗證時比 3-Band 久，反之當時間比低於 3-Band 代表驗證時間比 3-Band 快。



$$\text{時間比} = \log_2 \frac{T_{K-Band}(S)}{T_{3-Band}(S)}$$

圖 41 K-Band 比較

4.3.2 K-Band 選定

從實驗數據中，很難比較出不同 K 值時，演算法的優劣，因此我們這邊同樣比較了總驗證時間；在總驗證時間上，5-Band 所花的時間最少，因此在之後的比較上面，我們選定 5-Band 為參考的參數。

演算法	3-Band	5-Band	7-Band
總時間(sec)	61269	58902	65199

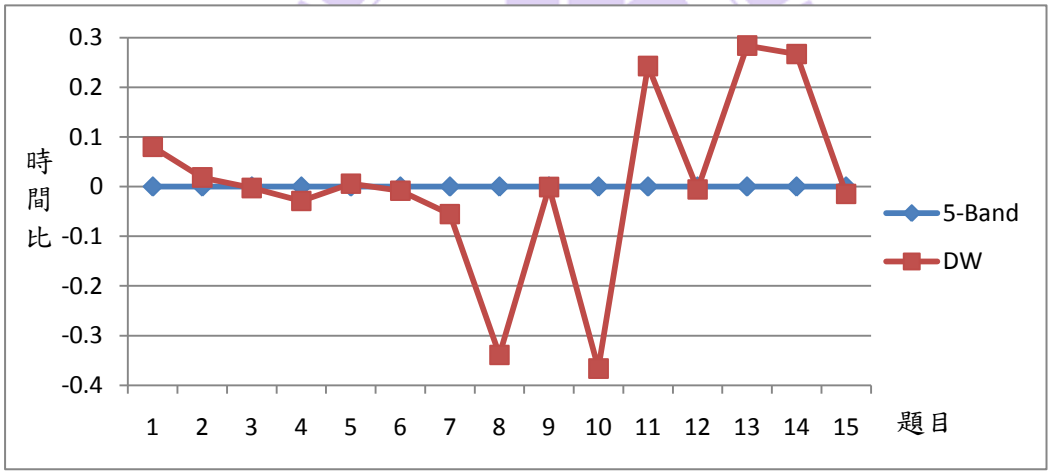
表格 7 不同 K-Band 比較

4.4 比較 K-Band 與 DW

從上述的比較中，JL-PNS 在 5-Band 機制下，稍微好於其他 K 值參數(3-Band、7-Band)，在這章節的比較中，我們比較 JL-PNS 加入 5-Band 機制和加入 DW 考慮前 5 個節點的機制下，兩個演算法的好壞。

4.4.1 實驗結果

圖 42 為實驗結果，橫軸代表題目，縱軸代表時間比，比較以 5-Band 為基準，低於 0 代表該題目證明時間比 5-Band 快，高於 0 代表該題目證明時間比 5-Band 慢。



$$\text{時間比} = \log_2 \frac{T_{\text{使用的演算法}}(S)}{T_{5-\text{Band}}(S)}$$

圖 42 K-Band 與 DW 之比較

4.4.2 比較

從所有題目來看，驗證時間上各有好壞，在大的題目方面 5-Band 驗證表現較好，從表格 8 顯示，在驗證總時間上面，5-Band 比較好，因此從數據顯示，5-Band 的機制稍微好於 DW 的機制。

演算法	5-Band	DW
總時間(sec)	58902	62400

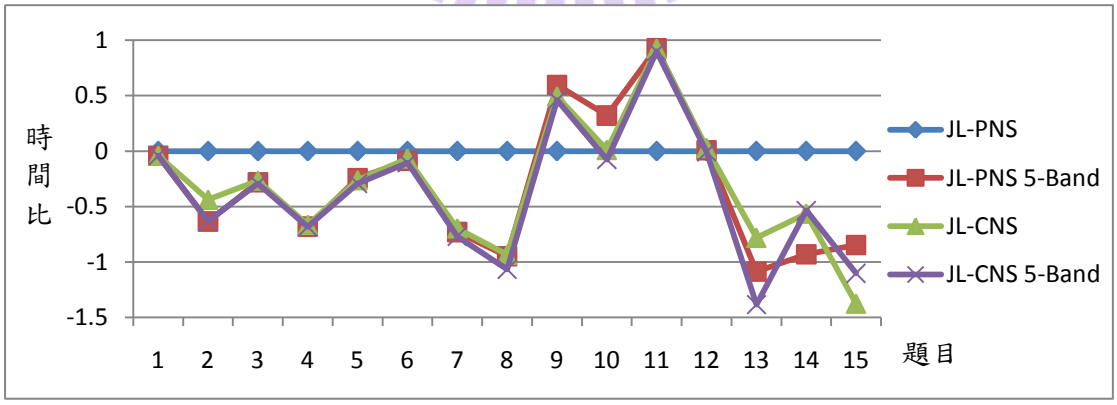
表格 8 K-Band 與 DW 之比較

4.5 綜合比較

在這一章節會比較 JL-PNS 和 JL-CNS 演算法，包含有無 K-Band 機制，才前幾章節中，決定 JL-CNS 最佳的 CNT 為 30，K-Band 最佳的 K 值為 5，因此在這章節一開始會先比較 JL-PNS、JL-CNS、5-Band JL-PNS、5-Band JL-CNS，在驗證實驗一時所花的時間。

4.5.1 實驗一綜合比較實驗結果

實驗一綜合比較實驗結果如圖 43 所示，橫坐標代表題目，縱座標代表時間比；比較基準是以 JL-PNS 做為基準，時間比比 0 大代表該題目驗證的時間比 JL-PNS 演算法還久，反之，時間比比 0 小代表該題目驗證的時間比 JL-PNS 演算法還快。



$$\text{時間比} = \log_2 \frac{T_{\text{使用的演算法}}(S)}{T_{\text{JL-PNS}}(S)}$$

圖 43 實驗一 綜合比較

4.5.2 實驗一結論

首先比較加上 K-Band 機制，JL-PNS 加上 K-Band 機制後，在驗證的時間上有明顯的變快，儘管在少部分題目變慢(題目 9、10、11)，但在大部分題目是變好的(2、3、4、5、7、8、13、14、15)，另外在較難的題目，驗證所花的時間也明顯較少。

JL-CNS 加上 K-Band 機制後，相較於 JL-PNS，並沒有明顯的改變，在難的題目上稍微有點差別，從總驗證時間上，JL-CNS 加上 K-Band 機制，時間有稍微變少一點，因此在選擇方面，還是認為 5-Band JL-CNS 稍微比 JL-CNS 好

另外比較 JL-CNS 與 JL-PNS，從數據顯示，JL-CNS 在盤面的驗證時間，會比 JL-PNS 來的佳，將兩個演算法加上 5-Band 機制後，兩演算法的驗證時間變得差不多。

演算法	**JL-PNS**	**5-Band JL-PNS**	**JL-CNS**	**5-Band JL-CNS**
總時間(sec)	83360	58902	57578	57253

表格 9 綜合比較時間表

4.5.3 實驗二

在選擇適當下法策略上，目前尚無找出一種完美的策略，然而在這篇論文中，我們試了三種可能的方法: 最大節點數量、PN/DN 比值、最佳棋局狀態，在接下來的實驗中會說明及比較這三種策略，找到最佳解的解題數。

我們採用三種演算法，JL-PNS、5-Band JL-PNS、5-Band JL-CNS，展開兩萬個節點，接著採用上述三種選點策略，分析其是否能找到正確解。

4.5.3.1 最大節點數量

在” 最大節點數量”的策略中，會將展開節點數量最多的分支，當做目前最佳著手。如圖 44，目前有三種下法，其分支數量為 4500、15000 及 500，在”最大分支數量”選點策略中，會將節點數 15000 的分支視為最好著手。

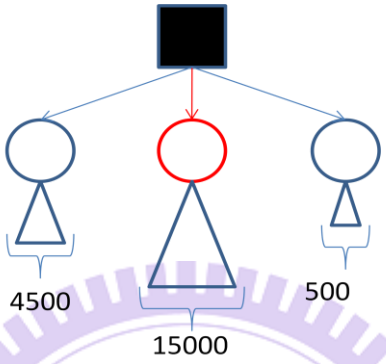


圖 44“最大分支數量”選點策略

表格 10 為各演算法在採用” 最大節點數量”策略選點時，選到最佳解的解題數量，在這種策略，5-Band JL-CNS 演算法由於全解出來，因此稍微好於另外兩種演算法。

演算法	**JL-PNS**	**5-Band JL-PNS**	**5-Band JL-CNS**
解題數	8	8	9

表格 10“最大節點數量”解題數

4.5.3.2 PN/DN 比值

在” PN/DN 比值”的策略中，若目前為 OR 節點，則在選擇上會選擇子節點中 PN 除上 DN 比值最小的節點，視為最佳著手；圖 45 中選點時，由於中間的節點 PN 除上 DN 的值 0.1，為三種下法中最小的，因此在這種策略下，會在則該節點會最佳著手；另外當目前節點為 AND 節點，則在選擇上會選擇子節點中

PN 除上 DN 比值最大的節點。

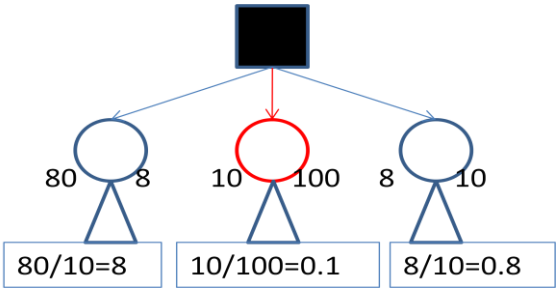


圖 45 “PN/DN 比值”選點策略

表格 11 為各演算法在採用” PN/DN 比值”策略選點時，選到最佳解的解題數量，在這種策略下，三種演算法解題數量相同，同為 8 題，因此三種演算法同樣好。

演算法	**JL-PNS**	**5-Band JL-PNS**	**5-Band JL-CNS**
解題數	8	8	8

表格 11 “PN/DN 比值”解題數

4.5.3.3 最佳棋局狀態

在” 最佳棋局狀態”的策略中，會將對目前下子最有利棋局狀態的節點，當做目前最佳著手。如圖 46，目前有三種下法，其棋局狀態分別為 W2、B2 及 W1，在” 最佳棋局狀態”選點策略中，由於目前該黑下，因此會將棋局狀態 B2 的分支視為最好著手。

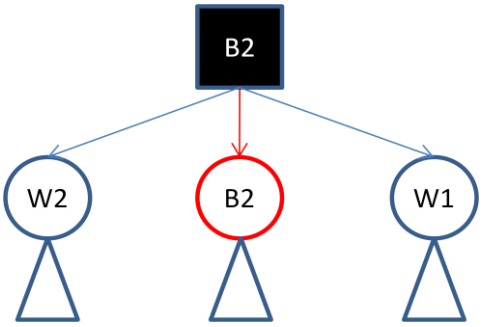


圖 46 “最佳棋局狀態”選點策略

表格 12 為各演算法在採用” 最佳棋局狀態”選點時，選到最佳解的解題數量，在這種策略，JL-PNS、5-Band JL-PNS 演算法解出了全部的 9 題，5-Band JL-CNS 解出了 7 題，因此 JL-CNS 稍微差於其它兩種演算法。

演算法	**JL-PNS**	**5-Band JL-PNS**	**5-Band JL-CNS**
解題數	9	9	7

表格 12 “最佳棋局狀態”解題數

4.5.3.4 最佳著手問題

在實驗二中比較了三種演算法，在找適當下法的表現，可以發現三種演算法能找出大部分的題目，然而儘管三種演算法幾乎都能找到適當的下法，可是卻存在一些問題，在對這幾個盤面的觀察中，發現展開節點的過程，往往會集中在某個分支，這個現象可以讓我們對該分支，某種程度上相信是一個好的下法，可是當要決定次好的下法時，因為其它分支搜尋數量都不足，往往很難分辨，也就是同一層搜尋分配不均造成的問題(圖 47)。

會有這個現象原因在於，JL-PNS 演算法是要找出勝敗演算法，當某個分支有機會達成該目標時，就會集中去證明該分支，儘管有多個必勝下法，也可能只有一個下法會很集中，JL-PNS 並不會去找次好的必勝方法，而 JL-CNS 與 JL-PNS 有許多相似之處，當某個分支離目前欲證明的棋局狀態很接近時，也會投入很多資源進去做搜尋，儘管其它下法也有機會達成該目標。

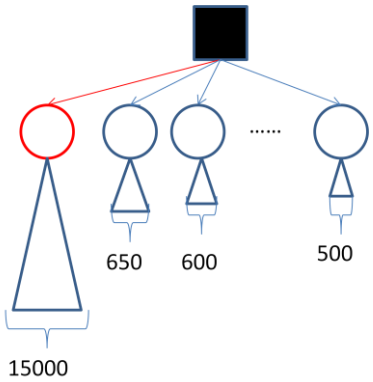


圖 47 選定最佳著手問題

第五章、結論與未來展望

在本篇論文之中提出了兩種演算法，工作層級謀反數演算法(Job-Level Conspiracy Number Search，簡稱 JL-CNS)與 K-Band 策略；在驗證棋局速度方面，JL-CNS 比原本工作層級證明數搜尋演算法(Job-Level Proof Number Search，簡稱 JL-PNS)表現更好，而加上 K-Band，兩種演算法(JL-PNS、JL-CNS)在驗證的速度上則是差不多，因此在加上 K-Band 機制下，對於驗證盤面時間上面，很難確實的比較兩個演算法優劣，唯一可確定的是加上 K-Band 機制有助於盤面驗證。

在本篇論文中，能更加快速的驗證盤面，原因在於改善原演算法 JL-PNS 選點時的策略；JL-CNS 中除了藉由證明數(PN)、反證數(DN)值來選點，當 PN、DN 值過高，較沒有參考價值時，會選擇次好的狀態來選點、搜尋；K-Band 機制則是改變 PN、DN、CN 值更新的方式，其想法很簡單，取最具代表性的 K 個節點更新，原本 PNS 演算法中，會確實的記錄證明必勝必敗所需展開的 leaf 節點個數，而選點依據 PN、DN 小的節點，代表著選擇節點時，會去選擇對手可能下法較少的著手，在許多遊戲中，目前盤面下法少，在某種程度上的確對下子方不利，但是在六子棋遊戲中，前面介紹過當攻擊方下出雙迫著時，防守方的下法相對會很少，卻不一定對攻擊方有利；由於在 JL-PNS、JL-CNS 節點會依照棋局狀態給予不同初始值，因此當採用 K-Band 更新數值時，某種程度在選點上會降低下法數的影響，轉而參考棋局狀態選點。另外能加速盤面驗證，代表著在選點上是較精確的，能選到較好的節點，因此對之後在選擇適當下法上有一定程度的幫助。

在實驗二中比較了三種演算法，也提供了三種選步策略，可以看出在選步策略上面，三種演算法各有好壞，目前難以確定最適當的方式，然而在正確解題上，三種演算法不管在哪種選步策略下，都能適當的找出絕大部分的最佳著手，但儘管如此，卻還是存在著無法確定次佳著手的問題。

最後，在未來展望方面，可以試著將工作層級(Job-Level)演算法用到其它棋類遊戲與發展 Job-Level-MonteCarlo；在 JL-PNS 與本篇 JL-CNS 研究中，是實做在六子棋遊戲上面，然而這些方法，可以同樣套用到象棋、圍棋等其它棋類遊戲 AI 上面，來解決開局、勝敗、死活等問題。JL-CNS 與 JL-PNS 選擇適當下法，往往對於次好下法的決定上較不準，因此可以試著嘗試 Monte-Carlo 的選點機制，製作 Job-Level Monte Carlo(JL-MC)演算法，Monte Carlo 中選點利用 UCT 做選點，會參考照搜尋節點個數與勝率。

參考文獻

[1] Connect6 Homepage, available at http://www.connect6.org/

[2] ICGA(International Computer Games Association) , available at http://ticc.uvt.nl/icga/

[3] Little Golem, available at http://www.littlegolem.net/jsp/

[4] WIKIPEDIA-Chinese Chess, available at http://en.wikipedia.org/wiki/Chinese_Chess

[5] WIKIPEDIA-Connect6, available at http://zh.wikipedia.org/wiki/Connect6

[6] WIKIPEDIA-GO, available at http://zh.wikipedia.org/wiki/Go

[7] CYC 遊戲大聯盟, available at http://cycgame.com/

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

[9] 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.

[10] 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.

[11] 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

[12] 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.

[13] 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.

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

[15] Lorenz, U., Controlled Conspiracy-2 Search, Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS), Springer LNCS, pp. 466-478, 2000.

[16] Lorenz ,U., Parallel controlled conspiracy number search, Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA) , ACM Press, pp. 320-321, 2001.

[17] 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.

[18] McAllester, D.A., Conspiracy numbers for min-max search, Artificial Intelligence, Vol. 35, pp. 287-310, 1988.

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

[20] Pawlewicz, J., and Lew, L., Improving depth-first pn-search: 1+ε 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.

[21] 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.

[22] Schaeffer , J, Conspiracy numbers, Artificial Intelligence, Vol. 43(1), pp. 67-84, 1990.

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

[24] 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

[25] 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.

[26] 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.

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

[28] 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.

[29] Wu, I.-C., Lin, H.-H., Lin, P.-H., Sun, D.-J., Chan, Y.-C., Chen, B.-T., Job-Level Proof Number Search For Connect6. Computers and Games, 2010

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

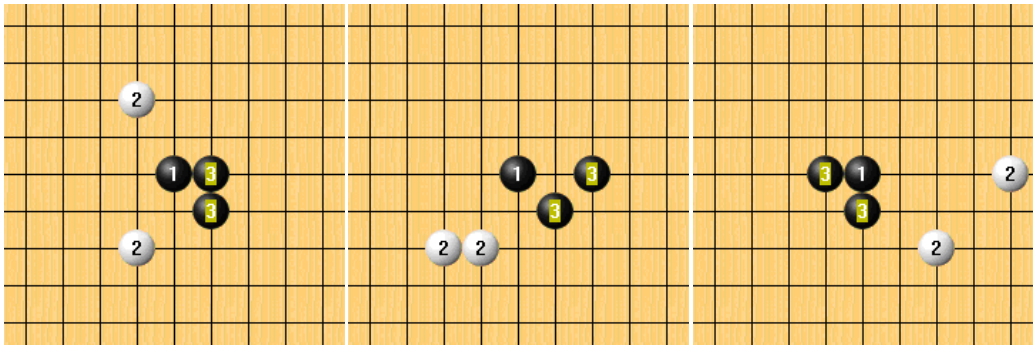
[31] Wu, I.-C., and Huang, D.-Y., A New Family of k-in-a-row Games. The 11th Advances in Computer Games Conference (ACG'11), pp. 180-194, Taipei, Taiwan, 2005.

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

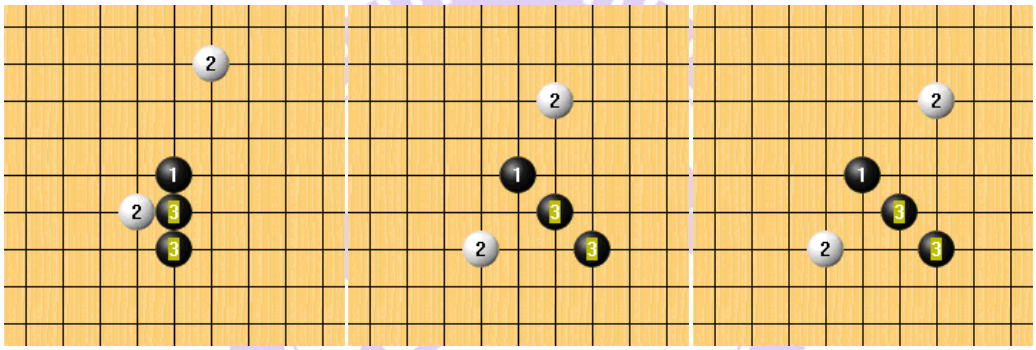
[33] Yoshizoe, K, "A New Proof-Number Calculation Technique for Proof-Number Search". Computers and Games, 2008.

附錄 I

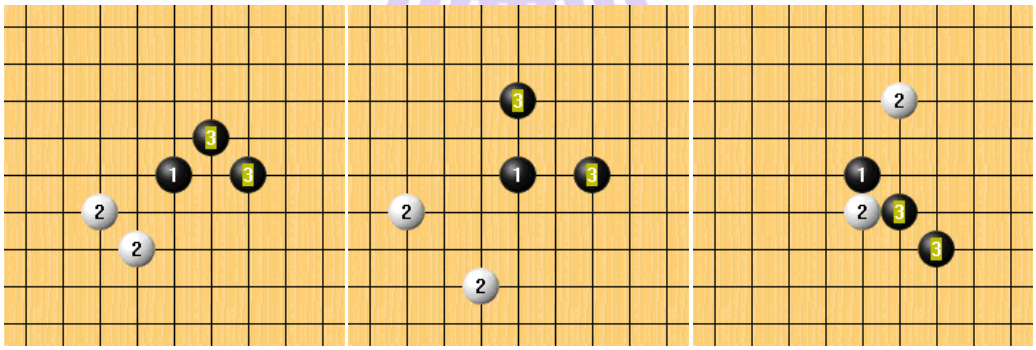
底下 16 個盤面為實驗一題目，其中 1 到 14 題為黑必勝，15 題為白必勝。



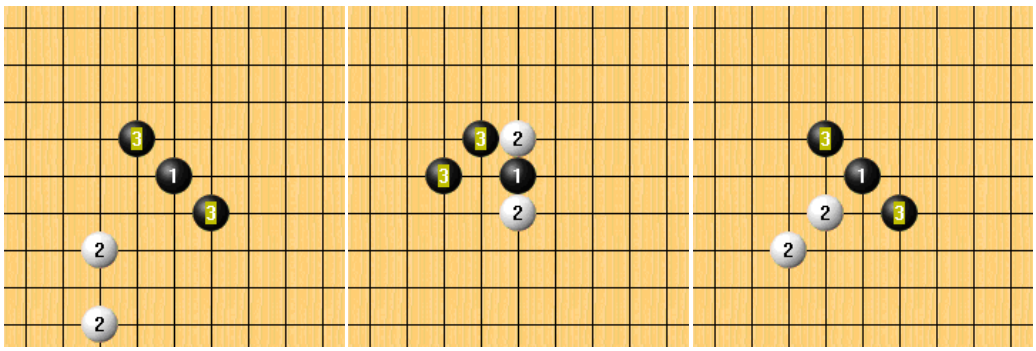
(1) (2) (3)



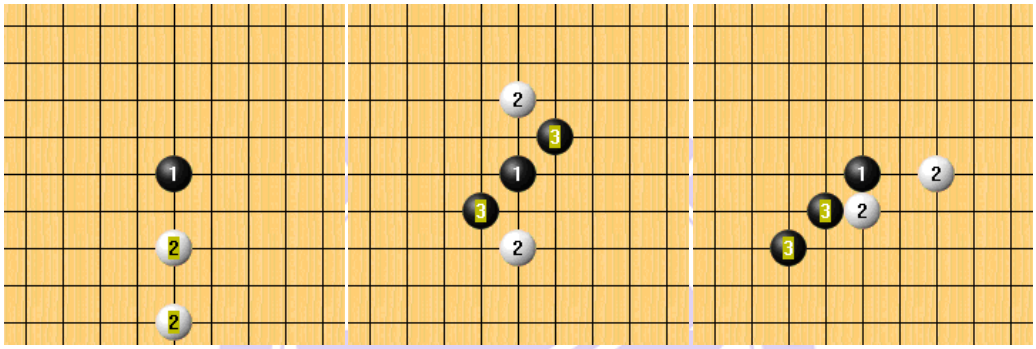
(4) (5) (6)



(7) (8) (9)



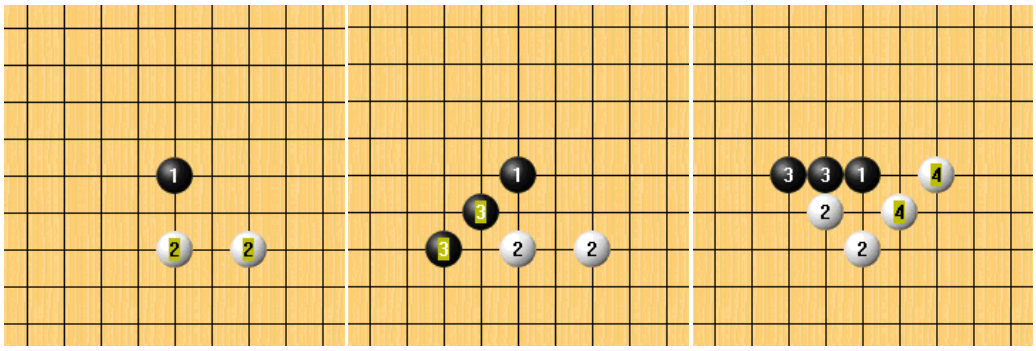
(10) (11) (12)



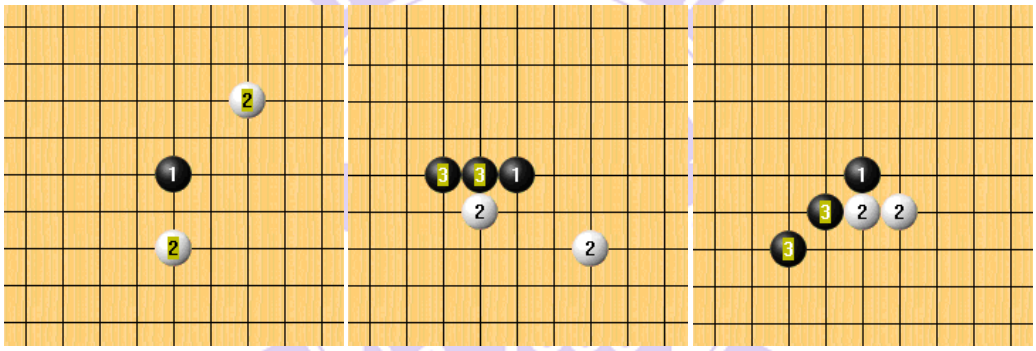
(13) (14) (15)

附錄 II

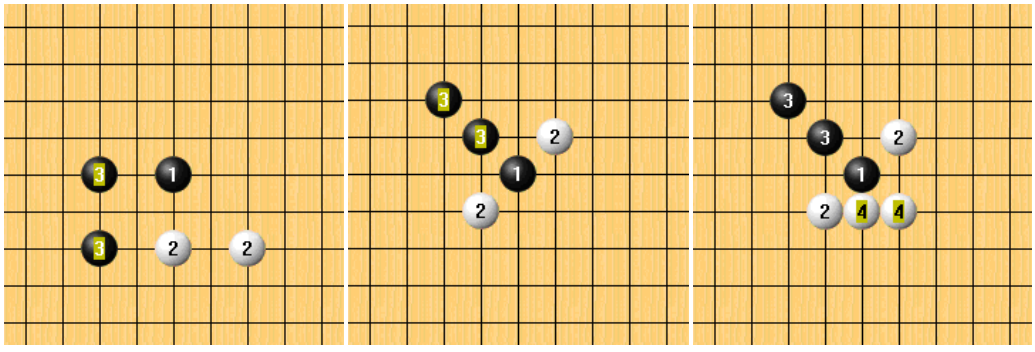
下圖為 9 個無勝敗，但是已知較好著手的盤面。



(1) (2) (3)



(4) (5) (6)



(7) (8) (9)