國立交通大學

資訊科學與工程研究所

碩士論文

以區域相關攻擊為主的六子棋搜尋演算法

Dependency-Based Search for Connect6

研究生：康皓華

指導教授：吳毅成　教授

中華民國　101　年 8 月

以區域相關攻擊為主的六子棋搜尋演算法

Dependency-Based Search for Connect6

研 究 生：康皓華　　Student：Hao-Hua Kang

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 2012

Hsinchu, Taiwan, Republic of China

中華民國 101 年 8 月

以區域相關攻擊為主的六子棋搜尋演算法

研究生：康皓華　　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

六子棋是在 2005 年由吳毅成教授所發明的一種棋類遊戲，近年來已經發展成為世界性的遊戲。本實驗室也針對六子棋研發了一個 AI 程式——NCTU6，它曾獲得兩屆六子棋奧林匹亞賽局競賽六子棋項目的金牌，以及在人機競賽中擊敗許多棋士。然而目前 NCTU6 仍有需改進的地方，當 AI 在進行迫著空間搜尋時，會花費相當多的時間做一些不必要的搜尋。

這篇論文的目的是為了利用區域相關攻擊的搜尋演算法(Dependency-Based Search)的概念來改善 NCTU6 的迫著空間搜尋演算法(Threat Space Search)。然而要將區域相關攻擊演算法的概念套用於六子棋極為困難，這篇論文提出了一些方法來解決相關的攻擊問題。

實驗結果證實了區域相關攻擊演算法應用於六子棋的效果優異，迫著空間搜尋的速度提升了約 3 倍，尤其是針對某些搜尋量特別大的盤面，能夠有效壓低搜尋時間，最高達到 50 倍以上的加速。

Dependency-Based Search for Connect6

Student: Hao-Hua Kang Advisor: I-Chen, Wu

Institute of Computer Science and Engineering
National Chiao Tung University

Abstract

The Connect6 game, first introduced by Professor I-Chen Wu in 2005, now becomes one of the popular games in the world. NCTU6 is a Connect6 AI program developed in our lab, which has won gold medals in ICGA tournaments twice and defeated many professional players in Man-Machine Connect6 championships.

The objective of this thesis is to apply the concept of Dependency-Based Search to Threat Space Search of NCTU6. Though the AI of NCTU6 works very well, it takes a lot of time on Threat Space Search for traversing many unnecessary positions.

The result of the experiment shows that Dependency-Based Threat Space Search speeds up by a factor of 3 in average, and about 50 times faster for some hard positions.

誌謝

這篇論文的完成，首先我要感謝我的指導教授吳毅成教授，教授日以繼夜地陪伴我做研究，也給了我諸多相關建議，甚至教導我許多研究方面的態度，使得本篇論文的研究得以有如此的結果。同時也必須感謝口試委員們，在我的研究方面給予諸多指導，我才能順利完成這篇論文。

感謝我的父母，謝謝你們從小辛苦教養我，在我求學的路上給我支持，教導我人生中待人處事的規矩，並給了我一個自由學習與發展的環境，讓我能夠在多元的學習環境中真正找尋我的學習興趣。

感謝實驗室的學長們。林秉宏學長講解了 AI 程式與演算法細節，使我得以快速理解實驗室開發多年的程式，進行研究與修改。林宏軒學長幫忙我檢查論文內容，也給了我很多程式上以及論文方面的建議。陳柏廷學長教導我相關演算法的內容。蔡心迪學長為我講解了很多程式上的細節。

感謝我的同學與學弟們。劉浩雲跟我合作開發編輯器，使得今日得以有功能更完善的編輯器能夠輔助論文的製作。郭青樺、吳東穎與張元耀維護實驗室的機器與網頁系統，使得我的實驗能夠順利完成。張傑閔耐心地與我一起更正論文的內容，以及接續後續的實驗。陳千越與廖挺富跟我合作課外的專案，一同練習各種程式設計的技術，提升演算法相關的能力。

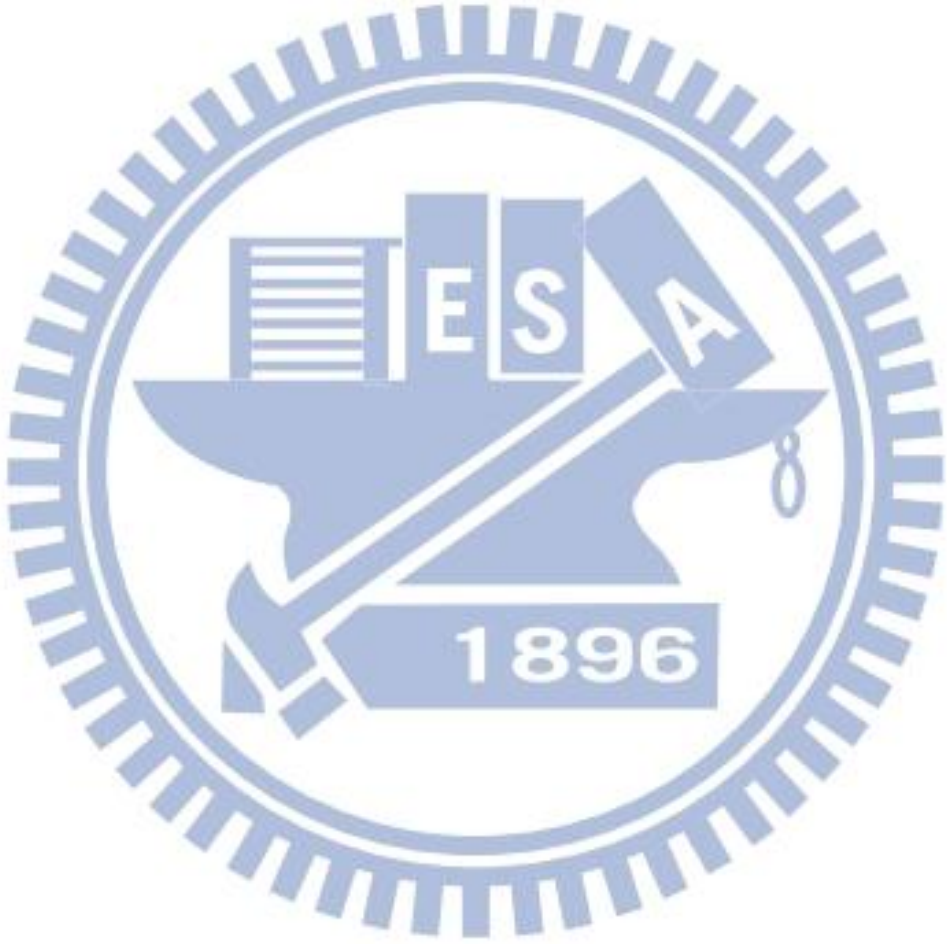
感謝我的女友張心怡，謝謝她在我焚膏繼晷地進行各項實驗及論文的寫作時，仍在我旁邊無微不至地照顧我，讓我能專心於研究。

最後，僅以此篇論文獻給所有陪伴我的家人、女友、老師、同學和朋友們，謝謝你們。

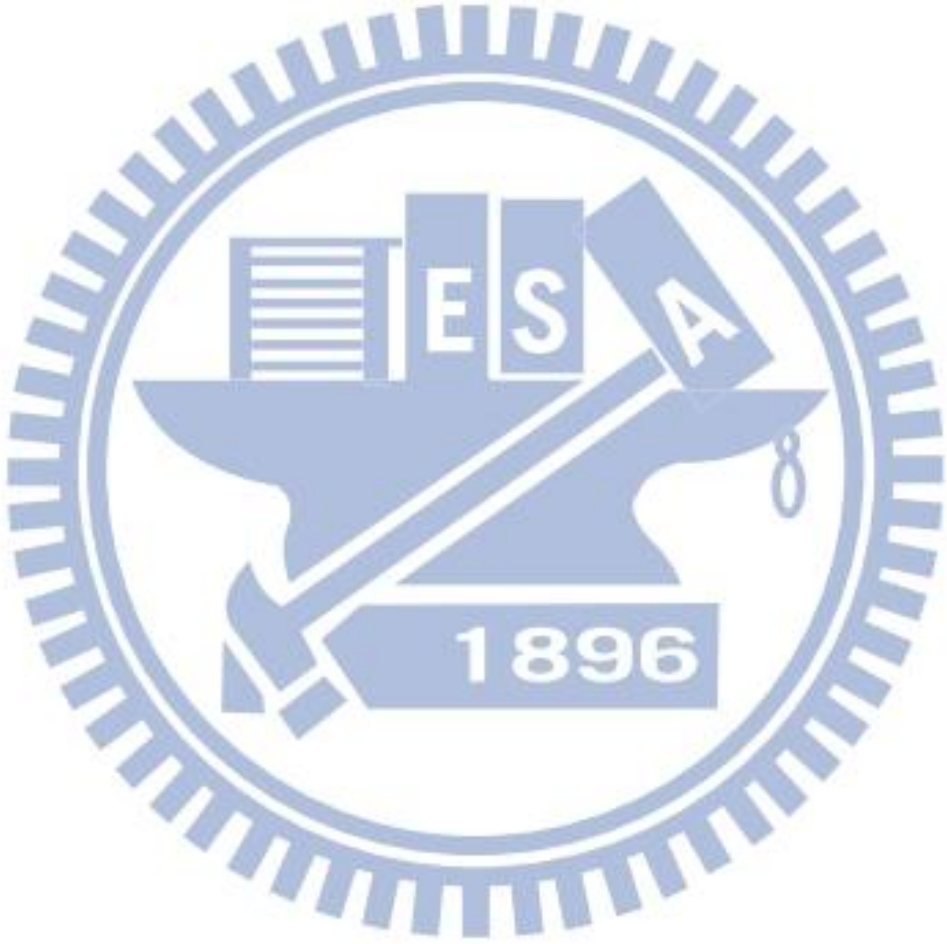
民國一百零一年八月　於　新竹交大工程三館 EC511 實驗室

目錄

圖表目錄



表格目錄



第一章、介紹

在一開始的章節中，會來介紹一些背景知識。第 1.1 節介紹六子棋的規則與特性，第 1.2 節介紹本實驗室所研發的 NCTU6 六子棋 AI 程式，第 1.3 節提出研究的動機和目的，第 1.4 節簡單整理此篇論文的貢獻，第 1.5 節介紹整篇論文的架構。

1.1 六子棋介紹

六子棋是交大吳毅成教授在 2005 年 9 月發表於第十一屆電腦賽局發展學術研討會(ACG 11)的一個遊戲[9][14][15]，由五子棋改良而成。以下將會介紹六子棋的有三個重要特性──規則簡單、遊戲公平、以及變化複雜。

第一個特性：規則簡單。六子棋遊戲使用十九路棋盤，先手持黑，下一顆子，接著雙方輪流各下兩子，先連成六子或更多者為勝。若下滿棋盤後仍未有玩家獲勝，則為和局。相較於五子棋，為了限制黑方的優勢，在規則的部分則針對黑方在遊戲中設定許多限制[8]。

第二個特性：遊戲公平。以五子棋而言，無論黑方或白方下子後，盤面中的黑子都不會少於白子(圖 1)。以六子棋而言，每一手下完後，皆會較對方多一顆子(圖 2)，因此具備潛在公平性的特質。

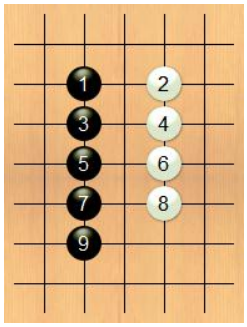


圖 1. 五子棋棋局

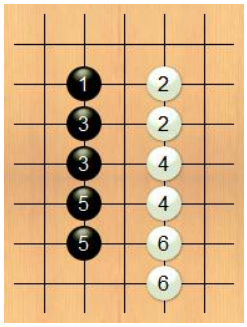


圖 2. 六子棋棋局

第三個特性：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(表 1)[15]。由於六子棋一次可以下兩顆子，使得變化相當豐富，每顆子平均約有 300 個位置可以下，因此一手的變化大約有 45,000 種。最後考慮一局平均的長度約 30 到 40 之間後，可推出六子棋的複雜度約為 10^{140}~10^{188}。

Games	Game Tree Complexity
Go （圍棋）	10^{360}
Shogi （將棋）	10^{226}
Connect6 （六子棋）	**$10^{140\sim188}$**
Chinese Chess （象棋）	10^{150}
Chess （西洋棋）	10^{123}
Go-Moku （五子棋）	10^{70}

表 1. 遊戲複雜度

1.2 NCTU6

NCTU6 為交大吳毅成教授實驗室團隊所開發的六子棋 AI 程式，又稱為交大六號，主要開發成員有以下幾位：吳毅成、林秉宏、林宏軒、蔡心迪、康皓華、張傑閔。NCTU6 開發至今，曾獲得過不少獎項，由表 2 可看出其棋力非常強[7][12][16]。

時間	比賽	NCTU6 結果
2006	第十一屆國際奧林匹亞電腦賽局競賽六子棋組	冠軍
2008	第十三屆國際奧林匹亞電腦賽局競賽六子棋組	冠軍
2008	世界棋王周俊勳與電腦六子棋對抗賽	3 勝 0 負
2008	第一屆人腦對電腦六子棋大賽	11 勝 1 負
2009	第二屆人腦對電腦六子棋大賽	8 勝 0 負
2011	第三屆人腦對電腦六子棋大賽	5 勝 3 負

表 2. NCTU6 戰績表

1.3 研究目的

由於六子棋在大多數盤面可以直接利用迫著空間搜尋(Threat Space Search，簡稱 TSS[1][4][14])來找到某一方的必勝解，因此迫著空間搜尋的搜尋效率變得格外的重要。NCTU6 在這部分的表現已經非常優越，但是在中盤甚至終盤時，只要盤面變得很大，搜尋時就容易在各個區域來回找尋可能的迫著攻擊，造成搜尋時間過長以及效率低落，使得很多必勝的盤面因時間限制而無法被搜尋到，也就是區域相關攻擊搜尋(Dependency-Based Search，簡稱 DB-Search[1][2])的問題。而此篇論文的目的就是為了解決此一問題而提出適用於六子棋的解決方法。

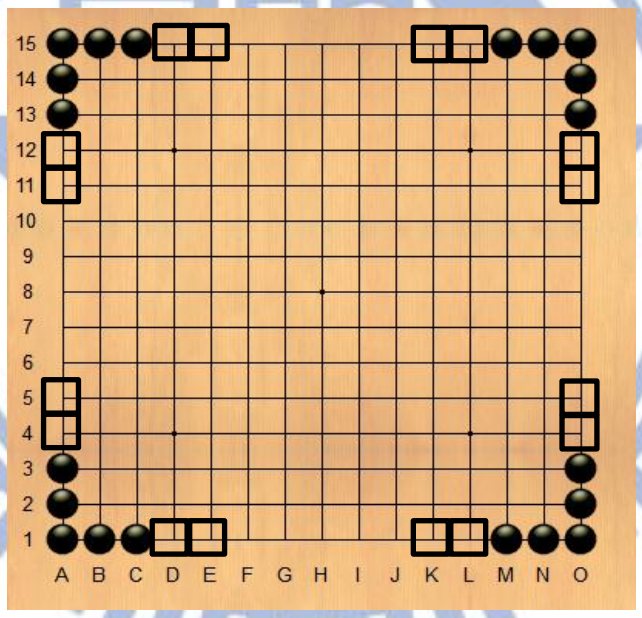


圖 3. 五子棋 DB-Search 代表盤面

Allis 對於 DB-Search 問題的敘述是以五子棋為例，考慮圖 3 的盤面。若將此盤面進行直接的搜尋，大約要搜尋 $8! \times 2^8 \simeq 10^7$ 個盤面，非常耗時。即使搭配了 Transposition table 的技術，避免搜尋重複的盤面，也仍然需要搜尋約 $3^8 \simeq 6000$ 個盤面。以人類玩家而言，可以立即看出無論從這 16 個攻擊位置的任一處進攻，都無法找到必勝解。因此在五子棋的迫著空間搜尋時，只需考慮一連串 Dependency 性質的攻擊迫著，在一個區域內

攻擊並找尋必勝解，而不用費時在不同的區域間來回搜尋，這是 DB-Search 最重要的特性。

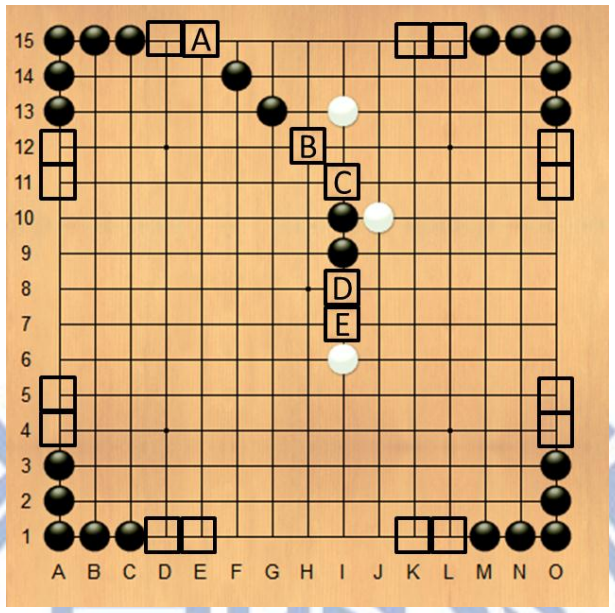


圖 4. 五子棋 DB-Search 存在必勝解的盤面

圖 4 為存在必勝解的盤面。依照 DB-Search 的方式，從 A 開始，接著攻擊 C，最後是 D，黑方就能因形成活四而勝利。這一連串的攻擊，前後手都是互相相關的，B 造成的迫著包含著 A，C 造成的迫著也包含著 B，直到勝利。因為相關性，將攻擊的棋串集中在一個區域，而不是四處分散。

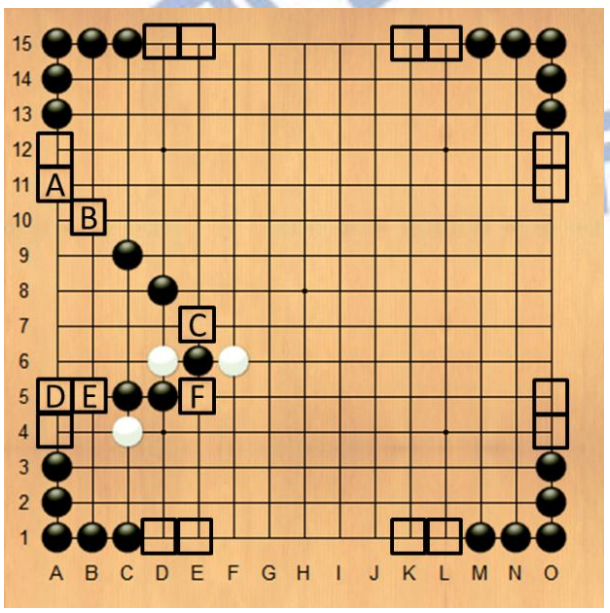


圖 5. 五子棋 DB-Search 需合併的盤面

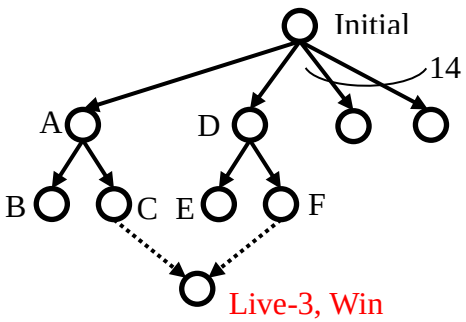


圖 6. 圖 5 的 DB-Search 樹

不過在完全考慮 DB-Search 的情形下，無法找到所有可能的必勝解，如圖 5。其中一條攻擊的路徑為 A➔C，之後便沒有其他與 C 相關的迫著攻擊。另一條攻擊路徑為 D➔F。兩條攻擊路徑看似都沒有後續攻擊，但是當這兩條攻擊路徑同時存在時，C 和 F 便可合併形成活三，使得攻擊得以延續(圖 6)。因此當兩條或更多攻擊路徑盤面沒有衝突，且能夠形成額外的攻擊迫著時，應該要予以合併。這是 DB-Search 另一個重要的特性：Combination。

這種 DB-Search 的問題同樣也發生在六子棋的盤面，且六子棋又較五子棋的問題更為複雜。主要的原因是六子棋一手有兩顆子，所以會有額外的一些問題，例如多重區域搜尋、反手問題以及演算法架構上的問題。

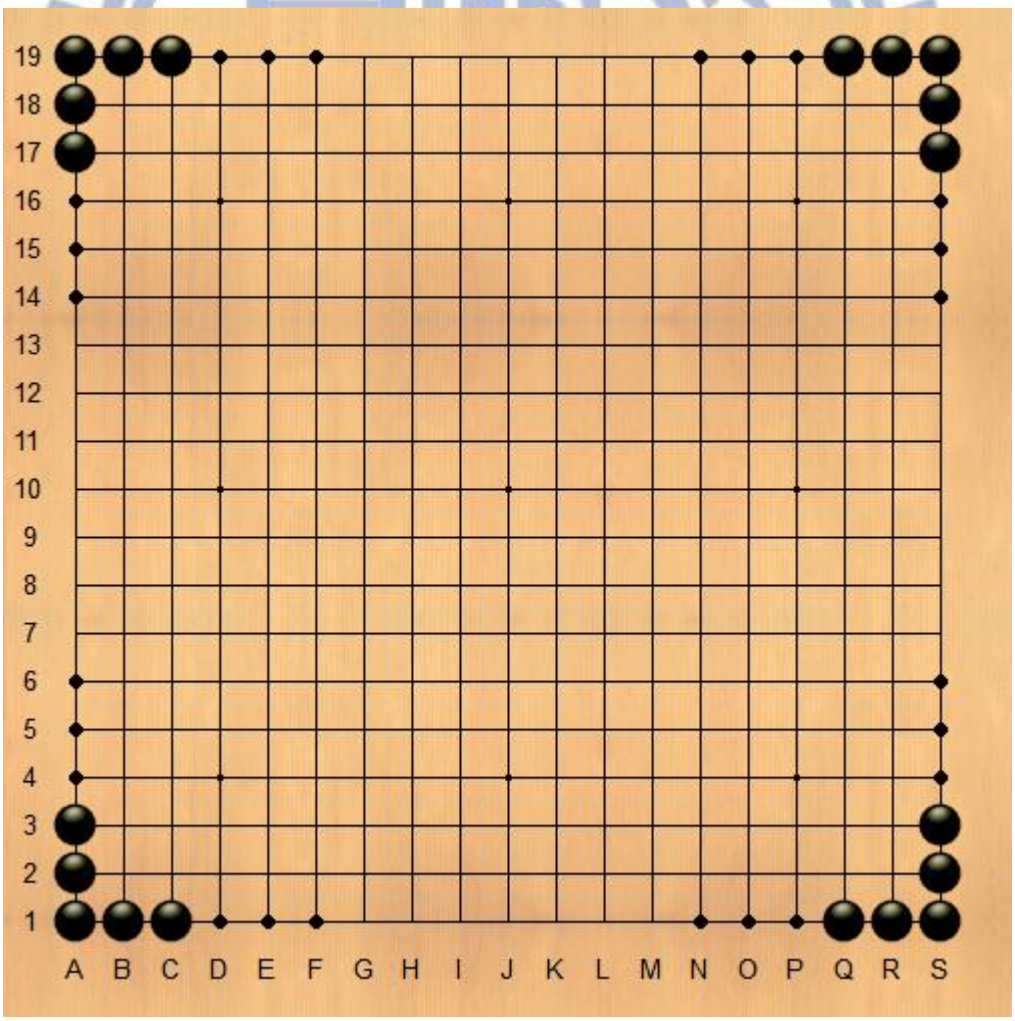


圖 7. 六子棋多重區域搜尋問題

六子棋的迫著攻擊時，攻擊方可以不斷以兩個迫著來進行攻擊。若考慮圖 7，攻擊方可以任意選擇兩個死三來搭配，形成兩個死四，迫使防守方兩顆子各需擋住一個死四。依照圖 4 複雜度的計算方式，可以推測出直接的迫著搜尋會搜尋$C_2^8 \times 9 \times C_2^6 \times 9 \times C_2^4 \times 9 \times 9 \simeq 1.6 \times 10^7$個盤面，即使搭配了 Transposition table 的技術也仍然需要搜尋約 $4^8 \simeq 65000$ 個盤面。但實際上人類玩家仍可以立即看出這個盤面不存在必勝路徑。

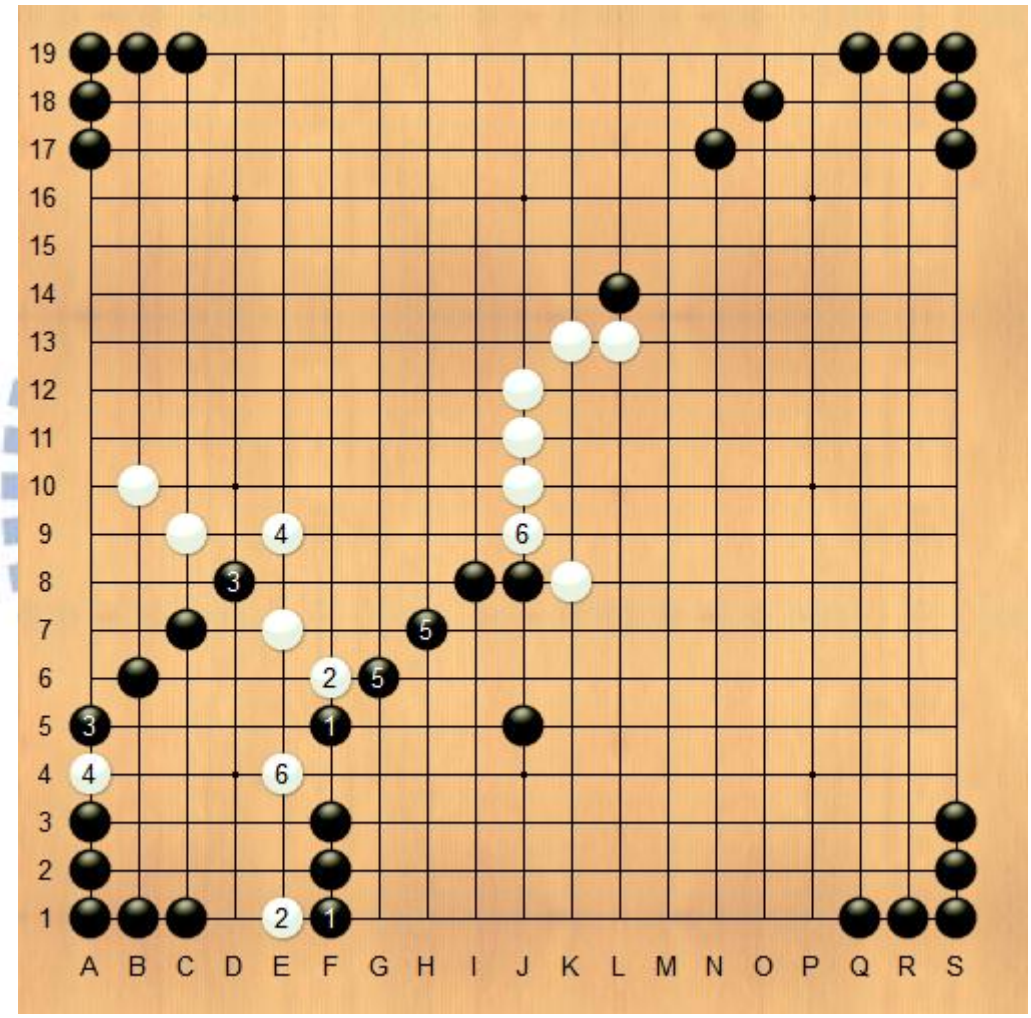


圖 8. 六子棋反手問題

考慮圖 8，在黑方進攻過程中，白 2 造成了反手。黑 3 雖然能擋住白 2 的反手，卻不相依於黑 1。之後黑 5 繼續以相依於黑 1 的迫著進攻，不過白 6 再次造成反手。若以圖 8 的順序進攻至白 6，便因黑方無法找出能擋住反手的攻擊步而須終止搜尋。

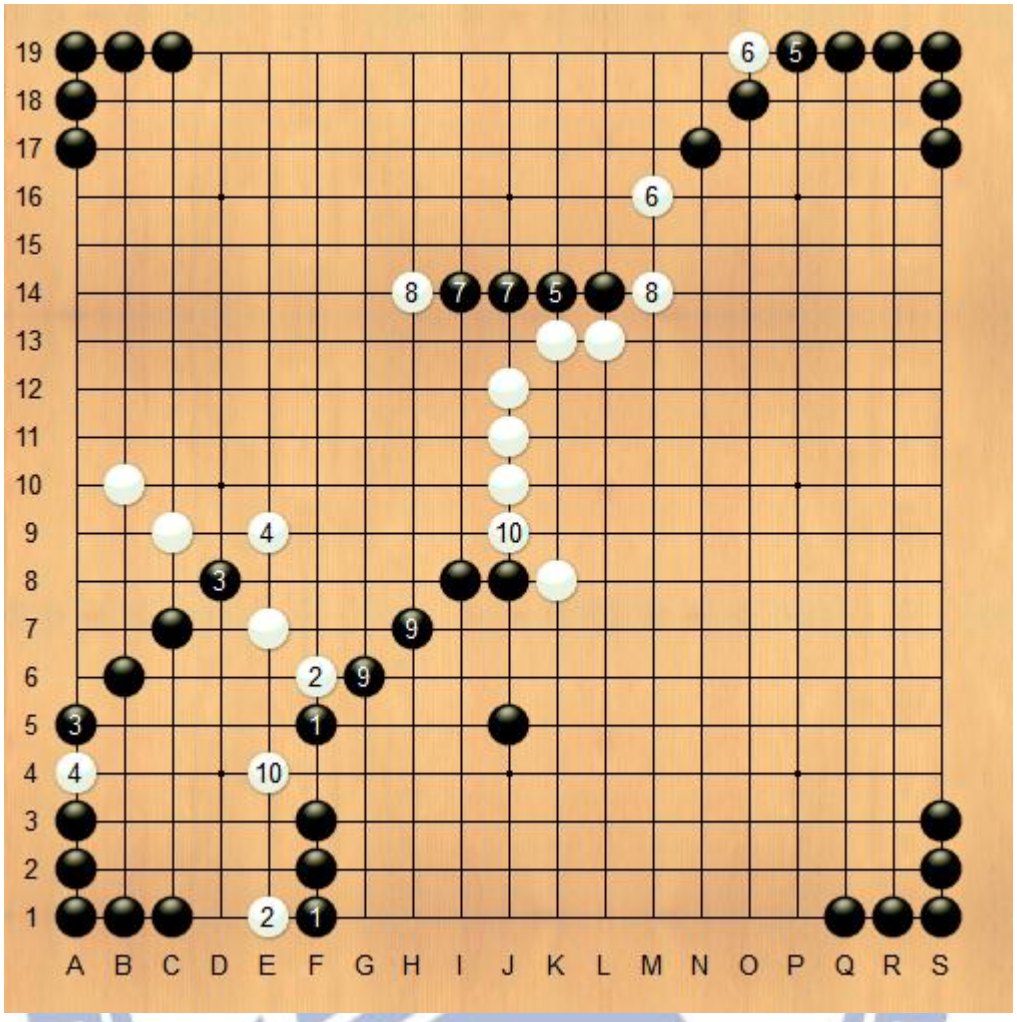


圖 9. 六子棋反手問題

若以圖 9 的順序，相較於圖 8，黑 5 先從盤面右上角進攻，下至黑 7 之後，再次從黑 9 的位置進攻。此次白 10 雖然與圖 8 的白 6 是在相同的位置，但由於黑 7 已經擋住了白 10 原本可能造成的反手，而使得黑方可以繼續從黑 9 進攻。

由此範例可明顯看出六子棋的反手問題搭配雙重區域搜尋，使得迫著搜尋變得更加的複雜，尤其是要考慮攻擊的順序，以及兩顆子搭配的問題，避免重複測試搭配盤面周遭的死三，才能減低搜尋的時間，。

1.4 貢獻

這篇論文的貢獻，主要為以下幾項:

- 將 DB-Search 的概念成功套用在六子棋的迫著空間搜尋。
- 改良 NCTU6 的迫著空間搜尋,成功降低搜尋必勝解時所花費的時間。

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研究以及演算法，例如迫著空間搜尋、DB-Search 等。第三章是研究方法，談論根據六子棋的特性設計的區域性演算法定義，以及實作內容。第四章是實驗，在該章節中會有演算法效能的比較。第五章是結論以及未來展望。

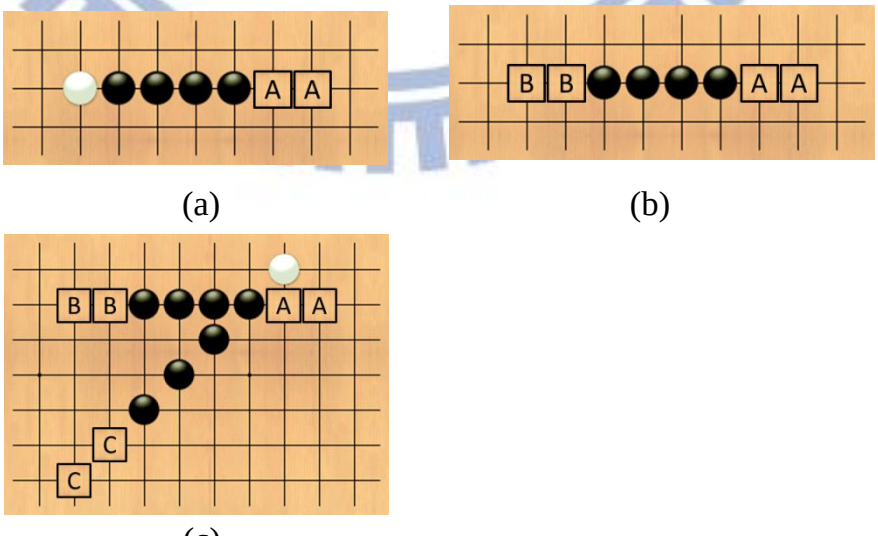
第二章、研究背景

在這個章節中，首先會在第 2.1 節介紹迫著空間搜尋，以及在第 2.2 節介紹保守迫著空間搜尋。在第 2.3 節講解 DB-Search 的理論及概念。

2.1 迫著空間搜尋

迫著的定義如下：我方存在 N 個迫著，則代表對方至少要 N 顆子才能擋住我方下一步勝利。

圖 10 為六子棋的迫著範例。圖 10.(a)是黑方有單迫著(Single-Threat)的範例，白方需花一顆子下在其中一個 A 的位置擋住黑方的迫著，否則黑方可以下兩顆子而形成連六。圖 10.(b)則是黑方雙迫著(Double-Threat)的範例，白方至少須在 A 和 B 各下一顆子才能阻止黑方連六，且下在 A 和 B 之間的距離必須少於 6，否則黑方仍能夠在中間形成連六。圖 10.(c)是黑方有三迫著(Three-Threat)的範例，白方必須花三顆子才能擋住黑方的迫著。若輪到白方下子，除非白方已經有迫著，否則黑方再下一手就能取得勝利。



(a) (b)

(c)

圖 10. (a) 黑方單迫著 (b) 黑方雙迫著 (c) 黑方三迫著

迫著空間搜尋就是攻擊方不斷利用迫著攻擊，找出必勝的路徑。而防守方只有有限的阻擋方法。在六子棋中，迫著空間搜尋可以簡單分成以下三種追殺[4][10]：

- 雙迫著追殺

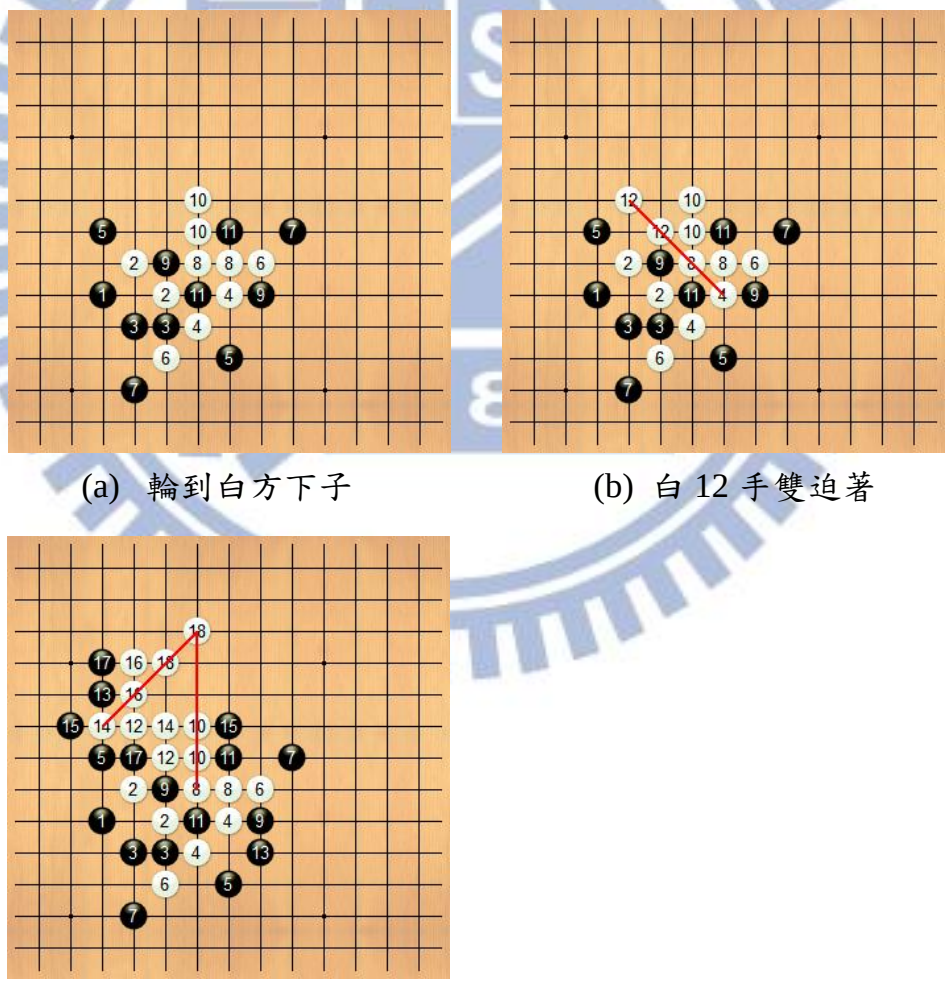
  Victory by Continuous Double-Threat-or-more moves (VCDT)

- 單迫著追殺

  Victory by Continuous Single-Threat-or-more moves (VCST)

- 無迫著追殺

  Victory by Continuous Non-Threat-or-more moves (VCNT)

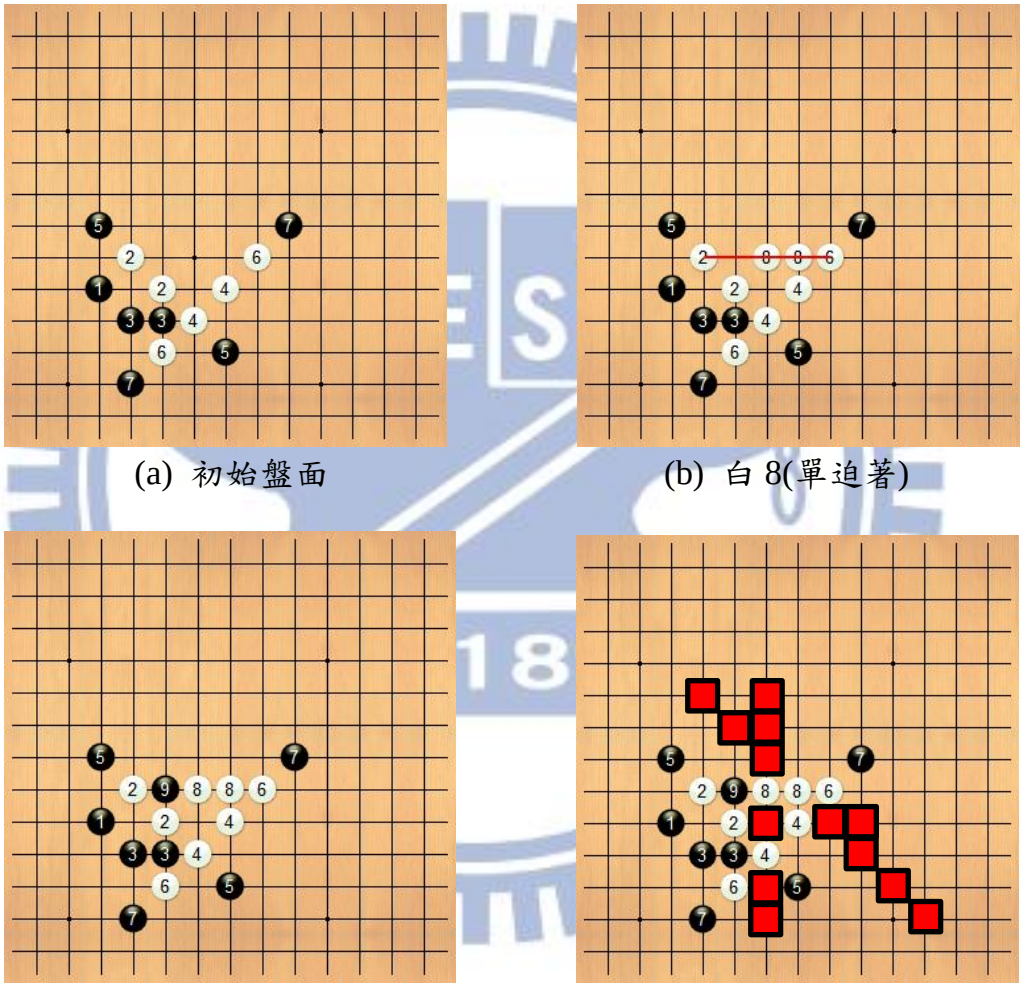


(a) 輪到白方下子　(b) 白 12 手雙迫著

(c) 白 18 手三迫著

圖 11. 雙迫著勝徑範例

雙迫著追殺代表從要搜尋的盤面開始，每次攻擊方下完，盤面至少要存在兩個以上的迫著。以圖 11 為例，攻擊方為白方，在圖 11.(a)時已經存在必勝路徑，可以以白 12 這個下法形成雙迫著，如圖 11.(b)。之後白方以不斷的雙迫著攻擊，下到白 18 這一手時，造成三迫著的盤面，如圖 11.(c)，此時是白方的必勝盤面。由於攻擊方每次下完都會有雙迫著，使得防守方的擋法非常有限，在搜尋時可以大大地減少搜尋複雜度。



(a) 初始盤面　(b) 白 8(單迫著)

(c) 黑只下一子　(d) 黑 9 形成的 RZone

圖 12. 單迫著勝徑範例

單迫著追殺代表從要搜尋的盤面開始，每次攻擊方下完，盤面至少要存在一個以上的迫著。以圖 10 範例而言，其實在圖 12.(b)的白 8 的單迫著就已經是必勝的步。雖然對手只需一顆子就可擋住這個單迫著，另一顆子

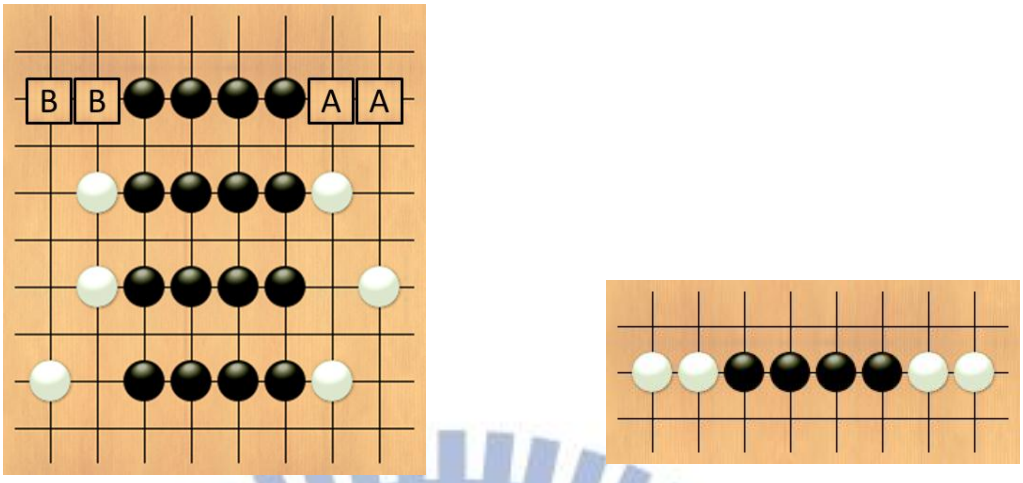
可以任意下在任何位置，但是無論下在何處，攻擊方都有方法能夠獲勝。因此可以讓黑方先下一顆子在黑 9 的位置(圖 12.(c))，另一子不下，接著讓白方進攻，形成區域 Relative Zone(簡稱 RZone[13][15])，如圖 12.(d)的方形區域。可以證明若黑方另一顆子不下在這個區域內，則白方必勝。因此這樣能夠縮小需搜尋的區域,降低搜尋複雜度。黑方另一子無論下在 RZone 內的任何位置攻擊方都有辦法獲勝，因此我們就能證明白 8 是個必勝的下法。

無迫著追殺代表從要搜尋的盤面開始，每次攻擊方下完，可以不存在任何迫著，並且在這樣的條件下搜尋必勝的下法。由於攻擊方沒有任何迫著，防守方的兩顆子都可以自由下，雖然仍可以利用 RZone 限制其搜尋範圍，但搜尋時間仍較雙迫著追殺和單迫著追殺長很多。

2.2 保守迫著空間搜尋

在 VCDT 搜尋時，雖然防守方的檔法因為攻擊方為雙迫著而減少到三到四個種，但是搜尋深度一旦加深，複雜度也是非常高的。在這裡可以用保守迫著空間搜尋(Conservative TSS，簡稱 CTSS)的技巧來更進一步減少防守方的檔法。

保守擋法(Conservative Defensive Move)(圖 13)的意義是一次將所有擋法濃縮成一步，造成防守方絕對優勢的狀況。如果針對每一手雙迫著，防守方都以保守擋法應對，而且最後攻擊方仍然獲勝，便能直接證明攻擊方必勝。但是假如攻擊方無法獲勝，則需檢查所有的保守擋法是否需要被拆開，再次檢查各種擋法。一旦保守擋法造成反手(Inversion)，或是保守擋法擋住了攻擊方的線段，例如活二、死三或活四，則保守擋法需要被拆開。



(a) 三種防守方的擋法　　(b) 防守方的保守擋法

圖 13. 保守擋法

2.3 DB-Search

DB-Search 演算法[1]是由 Louis Victor Allis 於 1994 年在博士論文中所發表的方法。

DB-Search 是一種 Single-Agent 的搜尋演算法，目的主要是縮小 State Spaces。它會將搜尋過的 States 建立成圖形結構，並記錄在記憶體中。DB-Search 主要分為兩個 Stages，Dependency Stage 和 Combination Stage。

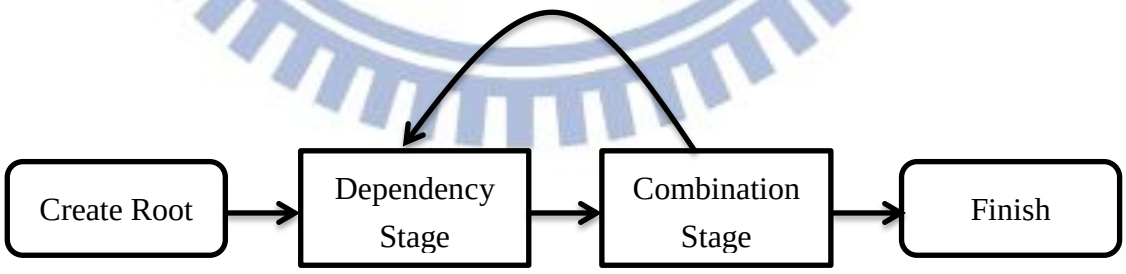


圖 14. DB-Search 的流程圖

以下簡單介紹相關定義：

- s：State Space 中的一個狀態
- f：一個 State 轉到另一個 State 的 Operator，$f = <f^{pre}, f^{del}, f^{add}>$

- f^{pre}：將 f 套用在 State s 時，s 必須包含 f^{pre} 的所有條件。以六子棋的 VCDT 搜尋為例，我們可以將一個 Move 看成是 Operator f，則 f^{pre} 就是活二、兩死三等能夠形成 Double Threats 的 Pattern。
- f^{del}：將 f 套用在 State s 時，需要將目前 s 的某些狀態刪除。以六子棋的 VCDT 搜尋為例，將下子位置的空格刪除。
- f^{add}：將 f 套用在 State s 時，需要新增某些狀態到 s。以六子棋的 VCDT 搜尋為例，將下子位置由刪除的空格改為黑方或白方的子。

- $f_i \prec f_j$：f_i 相依於 f_i，代表 $f_i^{add} \subseteq f_j^{pre}$。
- $F(N, f)$：一個 Meta-Operator。N 代表一個性質的集合，$F(N, f)$ 表示 Operator f 可以套用在一個 Set 的 Operators $\{f_1, f_2, \ldots, f_n\}$。$N \subseteq (f_1^{add} \cup f_2^{add} \cup \ldots \cup f_n^{add})$，並且 $N \nsubseteq$ 任何一個 $\{f_1, f_2, \ldots, f_n\}$ 的 Subset。此外，f 要相依於 $f_1, f_2, \ldots, f_n$。在 DB-Search 的概念中，這是最主要的 Operator。

定義完以上符號，接著講解建立 DB-Search 的圖的步驟：

1. 一開始先建立一個Root node，代表初始的狀態。
2. 在第一個stage時，不斷建立與其Parent有相依性的新Nodes。此時，代表不斷利用Meta-Operator $F(N, f)$去套用到初始的Nodes上，直到沒有新的Nodes產生。在這個階段所產生的Nodes都只有一個Parent。
3. 在第二個Stage時，只要合併某些Nodes之後可以產生新的操作規則，就將它們合併。不斷利用Meta-Operator $F(N, f)$去套用到上一個Stage建立的Nodes上，並可以套用一個以上的Nodes，來達成Combination。
4. 之後便不斷重複以上兩個stages，直到沒有新的Nodes可以產生為止。這兩個Stages也就是在實作DB-Search的兩個重要概念：Dependency和Combination，相關範例可見第1.3節。

以下 Double Letter Puzzle 來說明 DB-Search。首先，假設字母取代條件為以下規則：

$$\begin{aligned} aa &\rightarrow e \mid b \\ bb &\rightarrow a \mid c \\ cc &\rightarrow b \mid d \\ dd &\rightarrow c \mid e \\ ee &\rightarrow d \mid a \end{aligned}$$

則若題目為 *aabdee*，玩家必須想辦法不斷將兩個相鄰且相同的字母，取代為對應的條件式右方任一個字母，直到剩下單一一個字母。

$$aabdee \rightarrow bbdee \rightarrow cdee \rightarrow cdd \rightarrow cc \rightarrow b \mid d$$

當題目為 *aaccadd* 時，可以利用 DB-Search 快速找到解答。首先，需定義應用於此遊戲的 Operator f = (i, j, k, x, y)，代表將 Index i~j 和 j+1~k 的字母 x 取代為 y。

- f^{pre} = Index i~j 和 j+1~k 須為字母 x
- f^{del} = 刪除 Index i~j 和 j+1~k 的字母
- f^{add} = 將 Index i~k 加入字母 y。

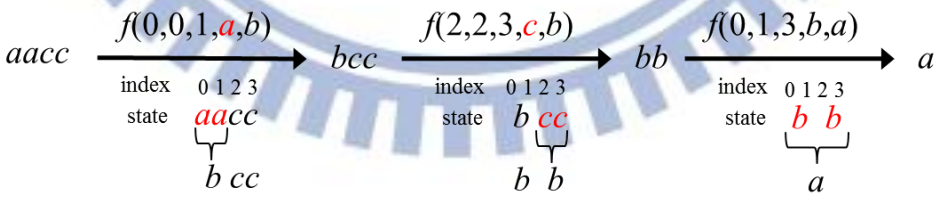


圖 15. Operator f 應用於 Double Letter Puzzle 的範例

DB-Search 一開始先建立 Root Node，代表初始狀態。接著進行第一個 Stage，也就是 Dependency Stage。在結束這一階段後，DB-Search 搜尋樹的圖形如圖 16。舉例而言，aaccadd → Eccadd，代表著兩個 States aaccadd 與 eccadd，可以套用 Operator f = (0, 0, 1, a, e)，代表從 Index 0 和 Index 1

的字母 a 取代為 e。在圖 16 中，大寫的英文字代表被取代後新增的字母。

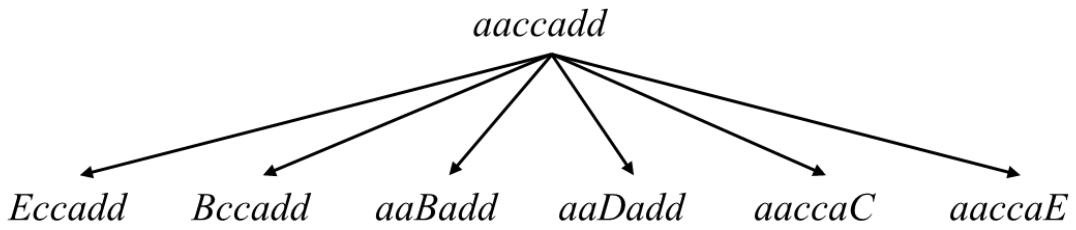


圖 16. 第一階段搜尋後的 DB-Search 搜尋樹

在第二個 Stage，也就是 Combination Stage，便開始尋找是否有兩個 Nodes 可以被合併。在這個遊戲中，由於f^{pre}只有兩個字母，因此最多只能合併兩個 Nodes。完成後的 DB-Search 搜尋樹顯示在圖 17。

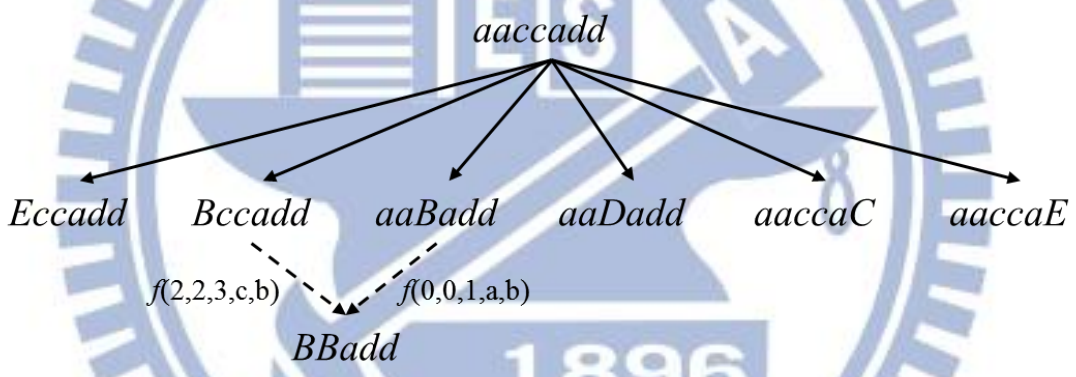


圖 17. 第二階段搜尋後的 DB-Search 搜尋樹

在圖 17 中，BBadd 即是套用 Meta-Operator *F(N, f)*的結果。f_1 = (2, 2, 3, c, b)，f_2 = (0, 0, 1, a, b)，而存在 *f* = (0, 1, 3, b, a)，使得 *f* 相依於 f_1 和 f_2。最後不斷重複這兩個 Stages，就能得到最後的 DB 圖形(圖 18)。在最後的圖形中，搜尋的 Nodes 數量非常少，便可以找到目標。可以觀察到 DB-Search 搜尋過程中，複雜度最高的部分就是在判斷 Combination 的合法性。

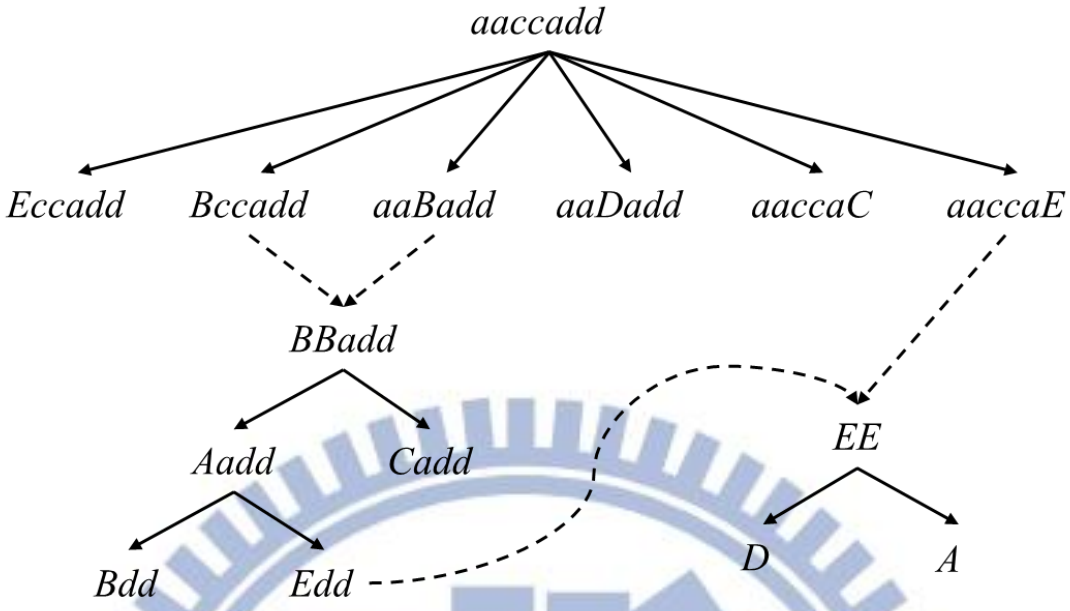


圖 18. 最終的 DB-Search 搜尋樹

第三章、研究方法

在這章節中提出適用於六子棋的 DB-Search 理論架構，在第 3.1 節中介紹 NCTU6 的 TSS 架構，說明要如何設計適用的 DB-Search 演算法。在第 3.2 節中，會依序對使用到的符號作介紹。在第 3.3 節中，將介紹 DB-Search 性質如何套用到 TSS 中。在第 3.4 節中，會介紹用來輔助 DB-Search 的圖形結構：Dependency-Based Hypergraph。在第 3.5 節中，會介紹演算法的性質。實做流程的部分會在 3.6 章節內。

3.1 NCTU6 的 TSS 架構

NCTU6 的 TSS 主要分為兩個階段：VCDT 和 VCST。VCDT 的部分主要是基於 Iterative-Deepening Depth First Search 的架構，搭配 Transposition Table 紀錄已搜尋過的盤面，減少搜尋的複雜度。NCTU6 不會將搜尋過的盤面完整記錄在記憶體中，只有部分會記錄在 Transposition Table 內。要使用 TSS 搜尋證明某個盤面為必勝時，必須整個 Subtree 都符合以下性質：若輪到攻擊方，只需一種攻擊步能獲勝。若輪到防守方，需全部防守步攻擊方都能獲勝(圖 19)。在 Iterative Deepening 的搜尋過程中，一旦找到必勝解，就馬上終止並且回傳必勝步。

(a) 輪到攻擊方下子　　(b) 輪到防守方下子

圖 19. TSS 樹更新勝利的方法

VCST 的部分，NCTU6 的設計是在 VCDT 搜尋時順便蒐集一些分數較高的攻擊方 ST Moves，並將它們記錄在一個 Queue 中。未來若第一次的 VCDT 未能找到必勝解，VCST 便會不斷從 Queue 中找尋最高分數的 ST Move，取出來驗證是否為必勝步，而驗證方法則會再去呼叫 VCDT 的函式去建立 RZone 以及搜尋必勝的路徑。

3.2 定義與符號

在介紹區域性 TSS 搜索(Dependency-Based TSS)之前，需要先做一些相關符號的定義以及介紹。

3.2.1 TSS Sequence

此篇論文在六子棋中定義以下符號：

- State s：一個 State，也就是一個六子棋的盤面。
- State r：TSS 開始搜尋的盤面，也就是 s_0。
- ψ：一個由攻擊方與防守方 States 輪流組成的序列$<s_0, s_1, \ldots, s_{2t}>$，$t \in \mathbb{N}$。
- TSS：{ ψ | 奇數編號的盤面在 VCDT 搜尋時，至少要包含兩個以上的攻擊方迫著。}
- $\psi \in TSS$：一個 TSS Sequence。若ψ的最後一個 State 為 Winning Position，則ψ就是一個 Winning Sequence。

若要證明 r 為必勝盤面，則對於攻擊方盤面，都存在一步攻擊方的必勝步。反之，輪到防守方的盤面，無論防守方如何擋，攻擊方都能獲勝。以符號表示，則$\exists s_1$，$\forall s_2$，$\exists s_3$，…，且所有結束的盤面都是必勝盤面。

在盤面 s_0 到 s_1 的過程中，是由攻擊方下了一步，也就是一個 Move m，定義為 m: $s_0 \rightarrow s_1$。以圖 20 為例，攻擊方的 Move 為 m_1: $s_0 \rightarrow s_1$，防守方

的 Move 為 m_2: $s_1 \rightarrow s_2$。在此篇論文中，我們定義：

- $P(m_i)$：Move m_i 包含的棋子的集合。
- $T(m_i)$：Move m_i 所造成的迫著集合。

因此 TSS Sequence ψ也可以表示為 Move Sequence，$\psi = <m_1, m_2, \ldots, m_{2t}>$。其中奇數編號的攻擊方 Move m_i 在 VCDT 搜尋時至少可以產生兩個以上的攻擊方迫著。

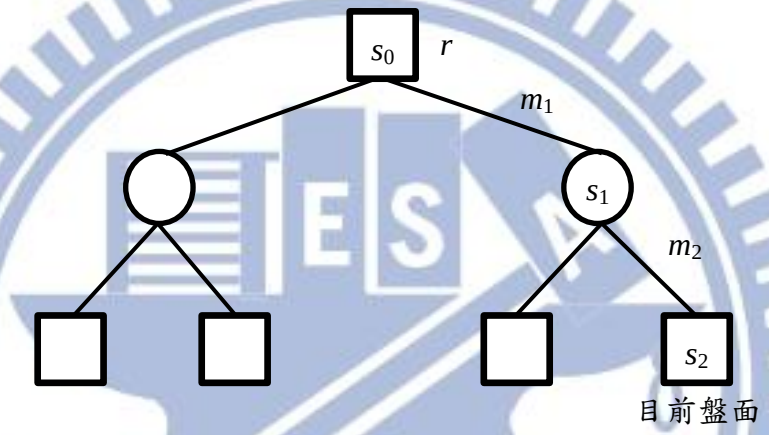
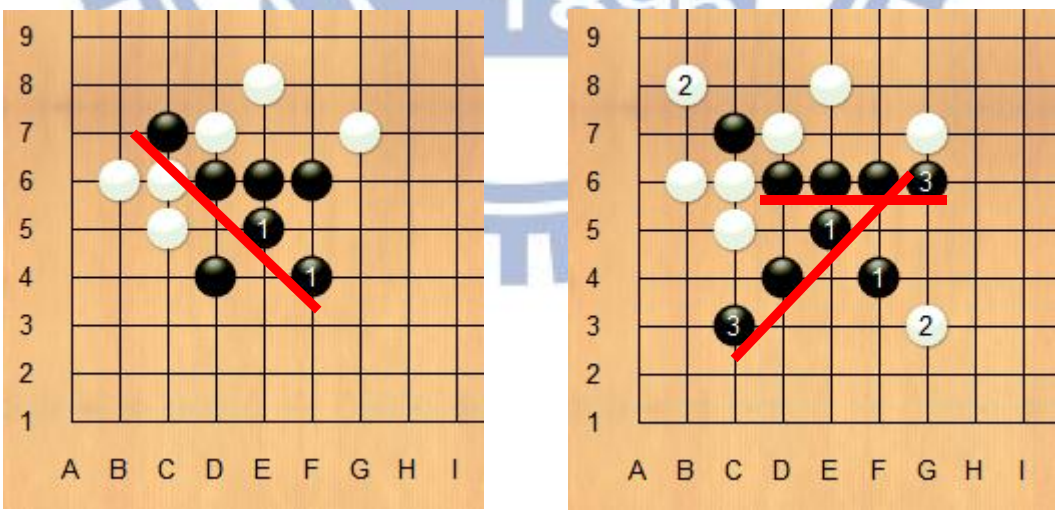


圖 20. TSS Sequence 的範例。$\psi = <s_0, s_1, s_2>$，$\psi \in TSS$



(a) $T(m_1)$　　(b) $T(m_3)$

圖 21. Moves、Pieces 和 Threats 的範例

為了講解方便，以圖 21 的 9 路盤面為例，m_1 為黑方第一手，$P(m_1)$ = {E5, F4}，$T(m_1)$就如同圖 21.(a)所示的線段，是為一個活四。m_3 就是黑方第三手，$P(m_3)$ = {C3, G6}，$T(m_3)$就如同圖 21.(b)所示的兩線段，是為兩個死四。

3.2.2 CTSS Sequence

為了簡化後續的演算法講解，之後將以保守迫著空間搜尋(Conservative TSS，簡稱 CTSS)來表示目前的搜尋。由於 CTSS 搜尋時，除了最後攻擊方造成三個以上迫著的必勝 Move 之外，每個防守方的 Move 都是固定且唯一的，在概念上可以將攻擊與相對應的防守步合併在一起，形成 Macro-Move M。若要證明 r 是必勝盤面，只需找到一條必勝的 CTSS Sequence ψ。

- CTSS：{ ψ | 攻擊方的 Moves 在 VCDT 搜尋時，至少要形成兩個以上的攻擊方迫著，且所有防守方的 Moves 都是保守擋法。}
- $\psi \in CTSS$：一個 CTSS Sequence。若$\psi = <m_1, m_2, \ldots, m_{2t}>$，亦能以 Macro-Move M定義，表示為$\psi = <M_1, M_2, \ldots, M_t>$，$M_i = <m_{2i-1}, m_{2i}>$。
  - M_i^A：表示M_i中攻擊方的 Move m_{2i-1}。
  - M_i^D：表示M_i中防守方的 Move m_{2i}。
- $S_L(\psi)$：代表 CTSS Sequence ψ的最後一個盤面。
- $M_L(\psi)$：代表 CTSS Sequence ψ的最後一個 Move。
- $Prefix(\psi)$：代表 CTSS Sequence ψ的所有前綴序列的集合。

例如$\psi = <s_0, s_1, s_2, s_3, s_4>$，也能以 Macro-Move 表示為$<M_1, M_2>$。則序列$\psi' = <s_0, s_1> \in Prefix(\psi)$， $S_L(\psi) = s_2$， $M_L(\psi) = M_2$。

圖 22 為一個$\psi \in CTSS$ 的範例，$\psi = <M_1, M_2>$。$P(M_1^A)$ = {E5, F4}，M_1^D為一個保守擋法，共有四顆子：$P(M_1^D)$ = {A9, B8, G3, H2}。

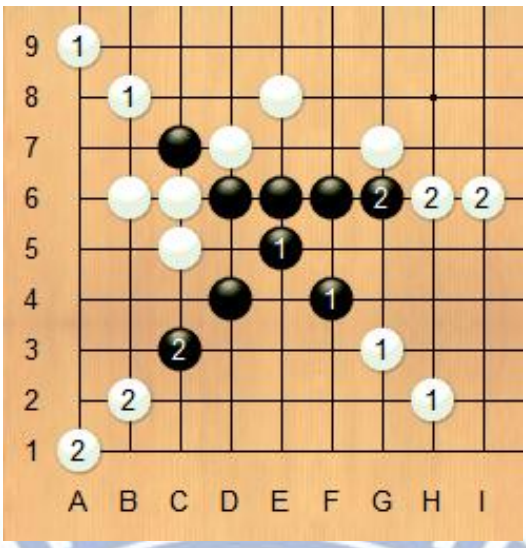


圖 22. 以圖 19 的盤面進行 CTSS

3.3 DB-TSS 的性質

為了將 DB-Search 的性質套用在六子棋的迫著空間搜尋，必須針對六子棋的規則做出相對應的定義。由於 NCTU6 的 TSS 大部分時間都花費在 VCDT 搜尋上，因此主要針對這個部分去套用 DB-Search 的性質。

3.3.1 VCDT 攻擊模式

在開始定義性質之前，需針對 VCDT 的攻擊模式做分類。雙迫著的攻擊 Move *M* 主要可以分為以下三種(參考圖 23)：

1. 兩個獨立的單迫著(ST_I)

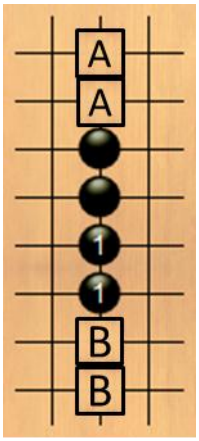
   *M*的兩顆子形成兩個死四，而且兩個死四並沒有共用這兩顆棋子。

2. 兩個相依的單迫著(ST_D)

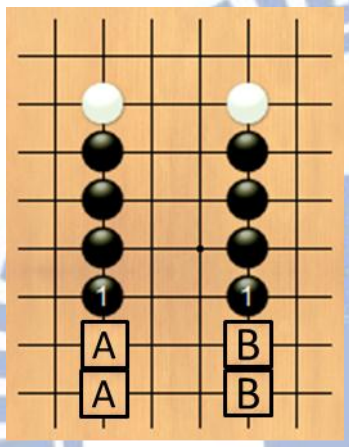
   *M*的兩顆子中，將盤面中的一個死二和一個死三形成兩個共用棋子的兩死四。

3. 兩子造成的雙迫著(DT)

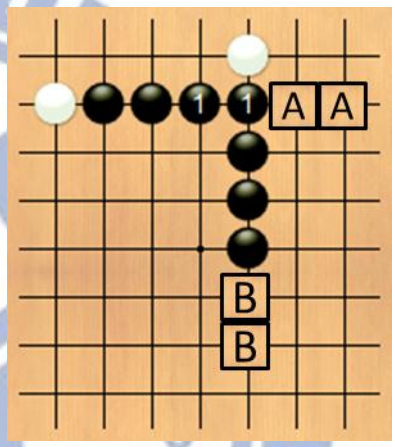
   以上ST_I和ST_D之外的雙迫著情形。



(a) 一子造成的雙迫著(DT_1)



(b) 兩個獨立的單迫著(ST_I)



(c) 兩個相依的單迫著(ST_D)

圖 23. VCDT 的雙迫著攻擊分類

3.3.2 Dependency 性質

首先，針對 Dependency 性質的定義，以$M_i \prec M_j$代表M_j相依於M_i。若M_j相依於M_i，則表示M_j^A產生的迫著包含了M_i的相關棋子(Related Piece, P_R)，可以表示為 $T(M_j^A) \cap P_R(M_i) \neq \emptyset$。相關棋子 $P_R(M_i)$的定義如下：

1. 若$i = 1$，則$P_R(M_i) = P(M_i^A)$。
2. 若$M_i^A \in \mathrm{ST_I}$，則$P_R(M_i) = \{p \mid p \in P_R(M_i)$且$T(p) \cap P_R(M_{i-1}) \neq \emptyset\}$

   但是若$P_R(M_i) = \emptyset$，則代表與上一步無關，因此強制設定$P_R(M_i) = P(M_i^A)$。
3. 其餘情形時，$P_R(M_i) = P(M_i^A)$。

針對 $P_R(M_i)$的定義，只有特別將攻擊步是兩個獨立單迫著的情形獨立出來討論，避免兩個單迫著之間沒有限制距離，而造成攻擊方在攻擊時，會在各區域間跳躍，反而失去了 DB-Search 的意義。

以圖 20 為例，$P(M_1^A)$ = {E5, F4}，由於是第一手，因此 $P_R(M_1)$ = {E5, F4}。$P(M_2^A)$ = {C3, G6}，$M_2^A \in ST_I$，因此需區分這兩顆子造成的迫著，是否各自包含 $P_R(M_1)$。由於只有 C3 造成的單迫著有包含 E5 $\in P_R(M_1)$，因此 $P_R(M_2)$ = {C3}，依此類推。一旦與前一步不相關，代表從新的一個區域開始搜尋，因此 $P_R(M_i) = P(M_i^A)$。

3.3.3 Combination 性質

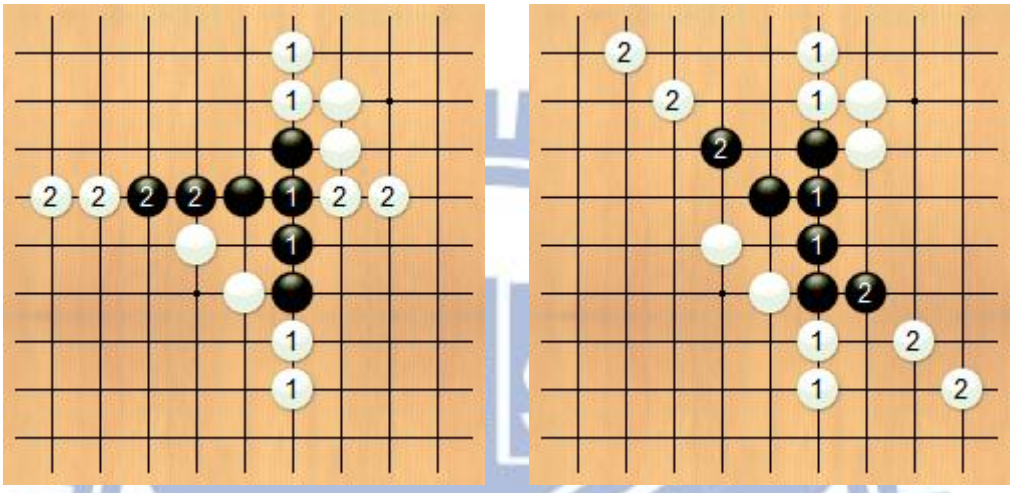
在 Combination 性質方面，需定義盤面間、Move 間以及 Sequence 間合併的合法性。對於兩個 Sequences ψ_1和ψ_2，如果他們沒有衝突(Conflict)，代表存在一個 Sequences ψ_0，$\psi_0 \in Prefix(\psi_1)$且$\psi_0 \in Prefix(\psi_2)$，並且符合以下所有性質時，便能說$\psi_1$和$\psi_2$沒有衝突，可以被合併：

1. $\psi_1 - \psi_0 \neq \emptyset$
2. $\psi_2 - \psi_0 \neq \emptyset$
3. $(\psi_1 - \psi_0) \cap (\psi_2 - \psi_0) = \emptyset$

對於兩個盤面 s_1 與 s_2，如果他們沒有衝突(Conflict)，則表示存在兩個 Sequences ψ_1 和 ψ_2，$S_L(\psi_1) = s_1$，$S_L(\psi_2) = s_2$，且ψ_1和ψ_2沒有衝突。以符號 $s_1 \cup s_2$ 來代表這兩個盤面的合併。

對於兩個 Moves M_i與M_j，如果他們沒有衝突(Conflict)，則表示存在兩個 Sequences ψ_1 和 ψ_2，$M_L(\psi_1) = M_i$，$M_L(\psi_2) = M_j$，且ψ_1和ψ_2沒有衝突。

從圖 24，可以看出ψ_1和ψ_2的第一手M_1是相同的，從以上的定義可得知ψ_0 = <M_1>。兩個 Sequence 各自的第二手是不相同的，各表示為$\psi_1 - \psi_0$和$\psi_2 - \psi_0$，符合以上定義的第一、二點。這兩個 Sequences 各自第二手的六顆棋子(攻擊方兩顆，防守方四顆)的交集是空集合，符合以上定義的第三點。因此這兩個 Sequences ψ_1和ψ_2之間是沒有 Conflict 的。



(a) ψ_1　　(b) ψ_2

圖 24. ψ_1和ψ_2的 Conflict 檢驗概念圖

最後對於 Moves M_1, M_2, …, M_n，如果他們可以在 CTSS 搜尋時被 Move M_{n+1}合併，則表示任兩個 Moves(M_i, M_j)，$1 \le i < j \le n$，彼此之間互相不衝突，且$M_i \prec M_{n+1}$，$1 \le i \le n$。

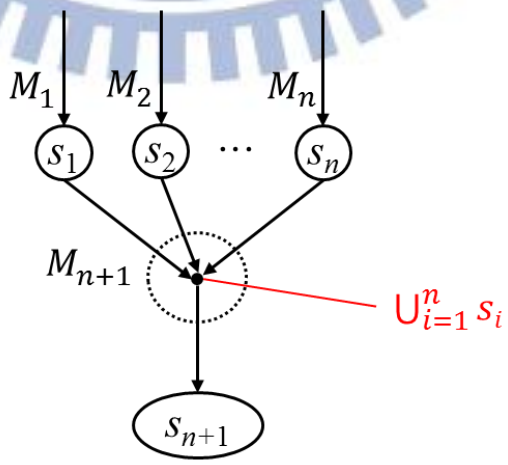


圖 25. M_{n+1}合併M_1, M_2, …, M_n的概念圖

3.4 DB-Hypergraph

由於 NCTU6 的 TSS 是基於 Depth-First Search 的演算法，因此不會將所有搜尋過的盤面紀錄在記憶體中。因此要實作 DB-Search 的性質時，需要額外的一個資料結構來輔助—Dependency-Based Hypergraph(DBH)，中文我們稱為相關攻擊超圖。

3.4.1 DBH 的定義與性質

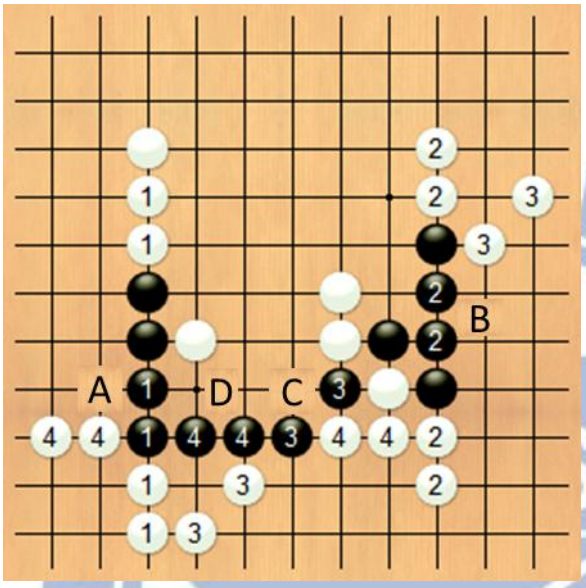
對於 DBH 中的定義如下：

- DBH $G = (\mathbb{V}, \mathbb{E})$：DBH G 是由 Vertex Set $\mathbb{V}$和 Hyperedge Set $\mathbb{E}$所組成。
- V：一個 Vertex，或稱為節點，代表一個盤面。
- $E = (\{V_1, V_2, \ldots, V_k\}, V_d)$：一個有向的 Hyperedge，起始節點為$V_1$, $V_2, \ldots, V_k$，目標節點為V_d。E包含了攻擊與防守雙方的棋子資訊，代表這兩個 Moves 是從盤面 state $V_1 \cup V_2 \cup \ldots \cup V_k$到盤面$V_d$。
  - E^A：代表攻擊方的棋子。
  - E^D：代表防守方的棋子。

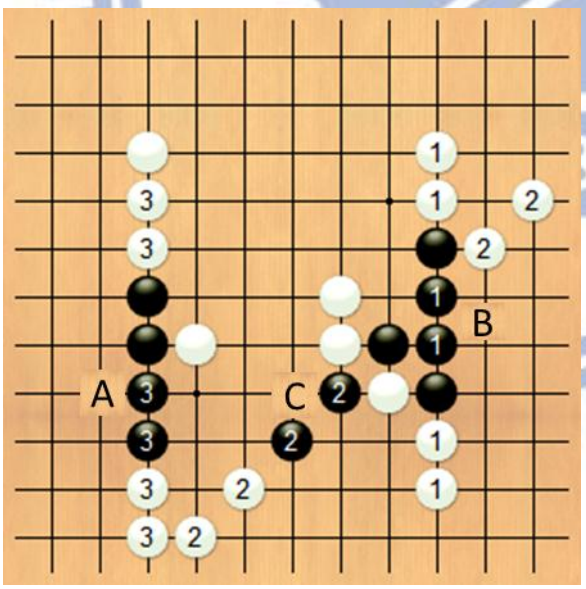
對於 DBH 的性質說明如下：

1. DBH是一個有向無迴圈超圖(Directed Acyclic Hypergraph)，類似拓樸圖(Topological Graph)。
2. 由於DBH內的所有Hyperedges都只有單一個目標節點，因此對於所有Vertex V，V的Indegree都為1。
3. 每一個Vertex代表一個攻擊方的Threat以及相對應的防守方的棋子，而攻擊方的Threat可能為Double Threat或Single Threat。
4. 每一個Vertex底下的Hyperedges表示這個Vertex的未來相關發展區域。

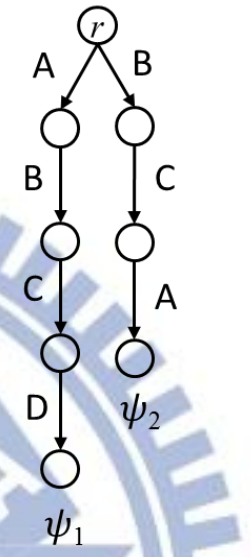
5. 對於每一個Vertex V與相對應的Hyperedge E，以及V的父節點V'與相對應的Hyperedge E'，$T(E^A) \cap P(E'^A) \neq \emptyset$。此性質定義了任兩個連續的Hyperedges都是相關(Dependency)的，同時也使得DBH的Vertex數量會遠少於CTSS搜尋時的Node數。



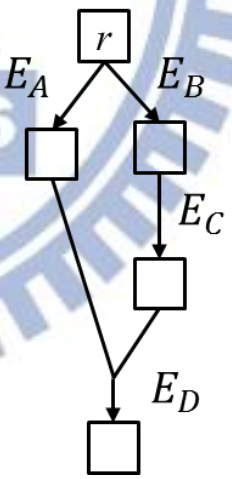
(a) ψ_1



(b) ψ_2



(c) CTSS搜尋樹



(d) DBH

圖26. 二條CTSS Sequences ψ_1和ψ_2及對應的搜尋樹和DBH

從以上兩個盤面，A、B、C、D 分別代表四個位置的攻擊位置，因此ψ_1 的第一手與ψ_2 的第三手是相同的位置，皆以 A 來表示，則 E_A 是位置 A 的攻

擊與防守棋子，依此類推。E_A和 E_B都能直接從 r 攻擊，而 E_C必須在 E_B下完後才能攻擊，因此 E_B為 E_C的 Parent，E_C相依於 E_B。因此要在位置 D 攻擊時，先決條件是 E_D的所有 Precedent Hyperedges 必須存在盤面上。

圖 26.(c)的 CTSS 需以七個 Nodes 來表示，但是相對應的 DBH 圖 26.(d)卻只需四個 Vertices 便能記錄這些攻擊的相關性，因此 DBH 的 Vertex 數會遠少於相對應的 CTSS 的 Node 數。

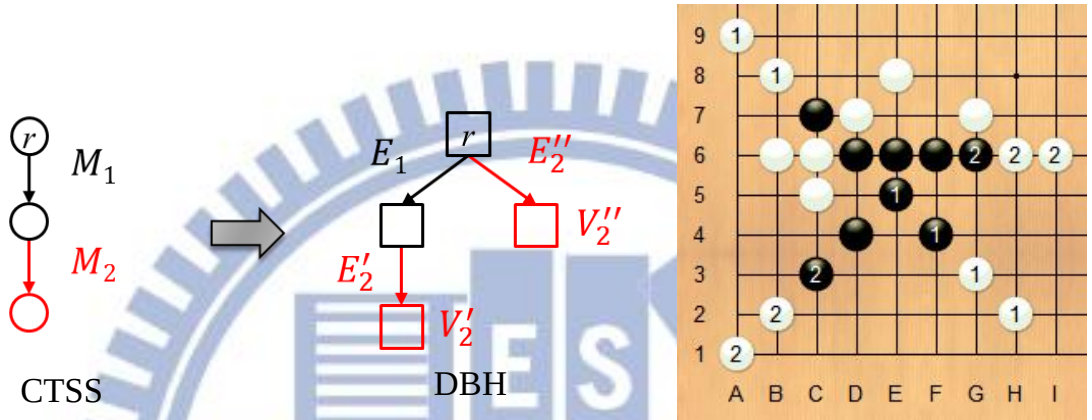
3.4.2 DBH 的建立流程

在 CTSS 搜尋中，每下了一個 Move，就將它加入 DBH。以一個 CTSS Sequence ψ為例，將整串 Sequence 建立到 DBH 的流程為依序對於每個 Move M，執行以下三個流程：

1. 若M^A為兩個單迫著的Move，建立相對應單迫著的兩個Vertices V'和V''；否則，建立一個Vertex V。
2. 對所有在第一個步驟建立的Vertex V，建立其相對應的Hyperedge E = <$\mathbb{V}_s$, V>，其中$\mathbb{V}_s$初始為一個空的Vertex集合。若E已經存在於DBH中，可省略此步驟。
   - 對所有在DBH內已被標記的Hyperedge E'，若$T(E^A) \cap P(E'^A) \neq \emptyset$，則將$E'$加入Set $\mathbb{V}_s$中。
   - 若$\mathbb{V}_s$為空集合，則將Root加入$\mathbb{V}_s$。
3. 標記所有新建立的Hyperedges。

第一個步驟建立 Vertex 時，若攻擊步M^A為兩個單迫著，$P(M^A) = (p_1, p_2)$且 $P(M^D) = (d_1, d_2, d_3, d_4)$，需建立相對應單迫著的兩個 Vertex V'和V''以及 Hyperedge 為E'和E''，其中M^A可以拆成E'^A和E''^A，$P(E'^A) = (p_1)$, $P(E''^A) = (p_2)$；M^D可以拆成E'^D和E''^D，$P(E'^D) = (d_1, d_2)$, $P(E''^D) = (d_3, d_4)$。

以圖 27 為例， Move M_1相對應 HyperedgeE_1已經存在於 DBH 中，而 Move M_2^A是兩個單迫著的攻擊，正要加入 DBH。要將M_2加入 DBH，首先須將M_2拆解成兩個 Vertex V_2'和V_2''及相對應的兩個 Hyperedges E_2'和E_2''，接著分別找尋它們的來源節點集合。假設E_1 = <{Root}, V_1>，則 $P(E_2'^A)$ = {C3}，E_2' = <{V_1}, V_2'>；$P(E_2''^A)$ = {G6}，E_2'' = <{Root}, V_2''>。



(a) M_2由 CTSS 加入 DB-Graph　　(b) M_2下完後的盤面

圖 27. 建立 DBH 的範例

3.5 DB-TSS 的方法

在 CTSS 搜尋中，本篇論文欲將不合理的攻擊方候選步所排除，只搜尋合理的攻擊步。以下將定義四種合理攻擊候選步的性質，以用來判斷一個候選步是否應該被排除。

3.5.1 Dependency

由前面章節的定義，我們可以針對$\psi \in CTSS$ 定義一個 Move Set $\mathbb{M}_D$：

- $\mathbb{M}_D = \{M_i \mid M_{i-1} \prec M_i$或 $i = 1\}$。

此 Move Set 代表ψ內第一手以及所有跟前一手相關的 Moves。對於 CTSS 搜尋，可以將攻擊模式歸類為以下的 Property 1。

Property 1：

如果 CTSS 中，存在一個 winning $\psi \in CTSS$，且ψ包含以下所有性質：

- 不存在 Moves 之間的 Combination
- 不存在反手
- 不存在兩個單迫著的 moves

則只需要搜尋 Moves $\in \mathbb{M}_D$，必能找到一個 winning $\psi \in CTSS$。

對於 Property 1，以較白話的解釋，即是存在一條必勝的 CTSS 路徑，可視為一連串的 Moves，且 Moves 之間不需合併，攻擊方的 Moves 順序即使隨意調換對方也不會形成迫著，以及不存在兩個單迫著的攻擊。在此前提之下，只需搜尋所有屬於$\mathbb{M}_D$的 Moves，也就是不斷找尋與上一步相關的 Moves，最終就能找到一個必勝的 CTSS Sequence，

3.5.2 Combination

在此節中，將針對$\psi \in CTSS$定義另一個 Move Set $\mathbb{M}_C$及補充相關定義：

- $E_i \prec E_j$：E_j相依於E_i，表示在 DBH 內E_i為E_j的 Parent。
- $E_i \ll E_j$：E_i可到達E_j，表示 DBH 中存在一條 Path 從E_i到E_j。
- $\mathbb{E}(M)$：一個由 Move M建立的 Hyperedges 的 Set。
- $\mathbb{M}_C$:$\{M_i \mid \exists\ E_c,\ E_i \in \mathbb{E}(M_i),\ E_{i-1} \in \mathbb{E}(M_{i-1}),\ E_i \ll E_c$且$E_{i-1} \prec E_c\}$

簡單來講，一個 Move 若屬於$\mathbb{M}_C$，則代表這個 Move 的前一個 Move 與這個 Move 未來的發展區域可以被 Combine。以 DBH 來表示，則可以參考圖 28。E_i代表 Move M_i所建立的其中一個 Hyperedge，E_{i-1}代表 Move M_i的上一手M_{i-1}建立出來的其中一個 Hyperedge，且最後會由E_c將這兩條 Path 合併，形成 Combination。M_i可能不相依於M_{i-1}，因為在此情形之下，M_i是在另一個區域的 Move，並且在未來能與M_{i-1}形成新的攻擊線段。

此外，$E_{i-1} \prec E_c$代表E_{i-1}必須是E_c的 Parent，代表著E_i只有在E_{i-1}下完之後才能從其他區域開始搜尋，不可在E_{i-2}下完就先跳躍到其他區域，否則形同沒有 DB-Search 的 TSS，造成 TSS 在不同區域間不斷來回搜尋，浪費搜尋的時間。

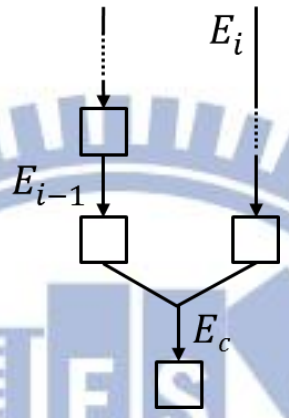


圖 28. $\mathbb{M}_C$ 的 DBH 概念圖

對 CTSS 搜尋，可以將攻擊模式更進一步地歸類為以下的 Property 2。

Property 2：

如果 CTSS 中，存在一個 winning $\psi \in CTSS$，且ψ包含以下所有性質：

- 不存在反手
- 不存在兩個單迫著的 moves

則只需要搜尋 Moves $\in (\mathbb{M}_D \cup \mathbb{M}_C)$，必能找到一個 winning $\psi \in CTSS$。

3.5.3 擋反手

接著，將討論反手的議題。在六子棋的 VCDT 中，一旦防守方在防守的過程中產生防守方的迫著，就稱作反手(Inversion)。此時攻擊方須花費額外的棋子去擋住反手，同時須造成雙迫著以便繼續攻擊。否則，攻擊即因反手的產生而終止。

- INV：Inversion，形成反手的盤面。
- ~~INV~~：Inversion，反手已被之前的攻擊 Moves 擋住的盤面。
- $M_i \rightarrowtail M_j$：表示M_i^A擋住了M_j^D造成的反手。

依照反手造成的方式，可以將擋反手分為兩類：

1. Post-Block：擋住上一手造成的反手，如圖29。
2. Pre-Block：擋住未來可能形成的反手，如圖30。

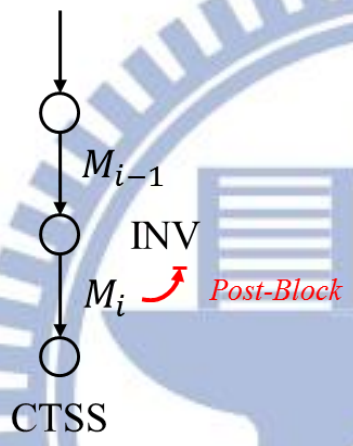


圖 29. Post-Block

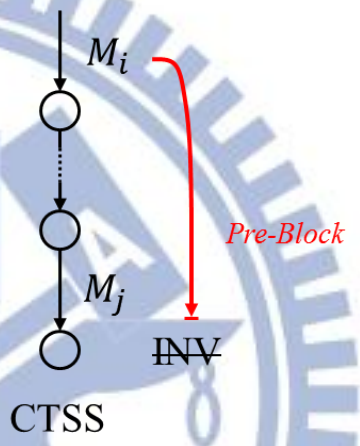


圖 30. Pre-Block

以 Post-Block 而言，$M_i \rightarrowtail M_{i-1}$表示$M_{i-1}^D$形成了防守方的迫著，而攻擊方在下一步$M_i^A$阻擋了這個迫著。在圖 31 中，$T(M_1^D)$造成了一個單迫著，但是黑方仍能靠著$M_2^A$擋住這個反手。以定義而言，$M_2^A \rightarrowtail M_1^D$表示$M_2^A$擋住$M_1^D$造成的反手。因此可以針對$\psi \in CTSS$定義一個 Post-Block 的 Set：

- $\mathbb{M}_{post\text{-}block}$：$\{M_i \mid M_i \rightarrowtail M_{i-1}\}$

以 Pre-Block 而言，$M_i \rightarrowtail M_j$雖表示M_i可以擋住之後M_j造成的反手，但是在M_j下完的盤面中，是不存在迫著的，因為迫著已經被M_i^A擋住。若將 $P(M_i^A)$從盤面上取下，則對方至少會存在一個迫著。

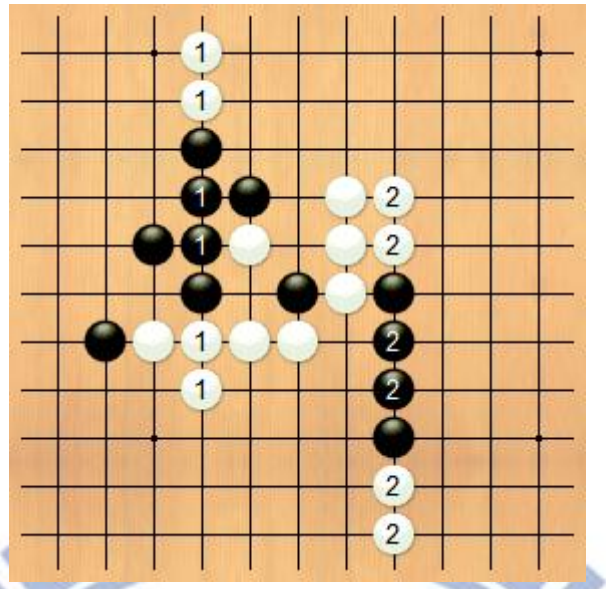


圖 31. Post-Block 反手範例

以圖 32 為例，$\psi \in CTSS$，$\psi = <M_1, M_2, M_3>$，雖然M_1^D、M_2^D和M_3^D都沒有形成防守方的迫著，但若攻擊方在初始盤面先下 $P(M_3^A)$，則防守方下了 $P(M_3^D)$之後，就會產生防守方的迫著。此時攻擊方再去下 $P(M_1^A)$想要擋住反手便已經來不及，因為攻擊方要擋住反手必須下到 $P(M_2^A)$的位置，也就是要兩手後，而防守方只要再一手便可因攻擊方未擋反手而形成連六獲勝。唯有照圖 32 的順序攻擊，才有可能在 VCDT 的攻擊中擋住這個反手。

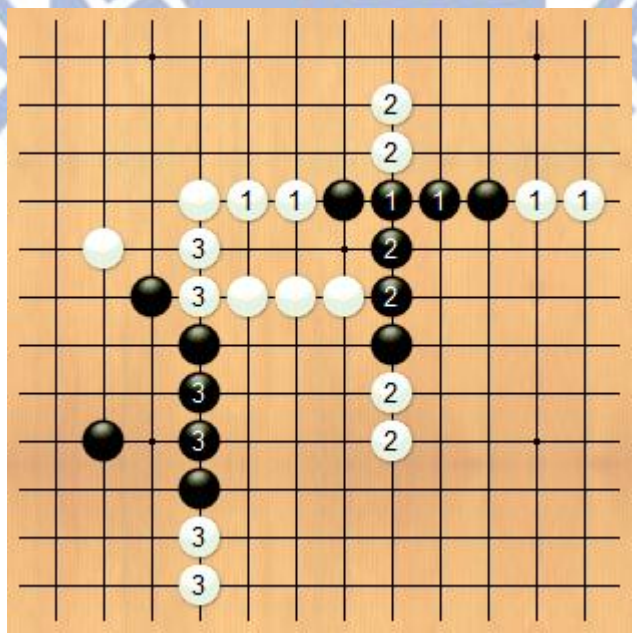
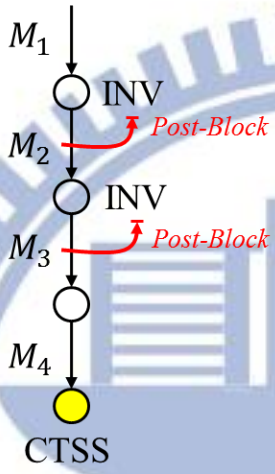


圖 32. Pre-Block 反手範例

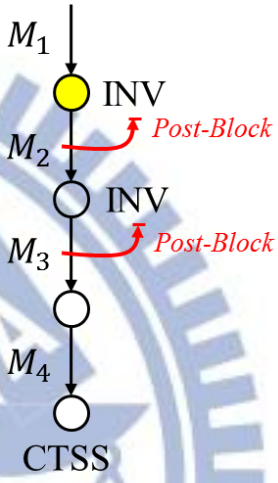
對於 Pre-Block 的 Move Set 定義較為複雜，需先對存在的$\psi \in CTSS$定義以下兩種連續反手的 Move Set：

- $I_{pre}(M_i)$：$\{M_{i-k} \mid$ 對於每個M_{i-n}，$2 \le n \le k$，$T(M_{i-n}^{D}) \neq \emptyset\} \cup \{M_{i-1}\}$。
- $I_{post}(M_i)$：$\{M_{i+k} \mid$ 存在$\psi' \in CTSS$，對於每個$M'_{j+n} \in \psi'$，$0 \le n \le k$，$T(M_{i+n}^{'D}) \neq \emptyset$且$P(M_{i+n}) = P(M'_{j+n})\}$。



$I_{pre}(M_4) = \{M_1, M_2, M_3\}$

圖 33. 之前的連續反手



$I_{post}(M_1) = \{M_1, M_2\}$

圖 34. 未來的連續反手

$I_{pre}(M_i)$表示M_i之前形成反手的連續 Moves，如圖 33。假設目前位置在M_4下完之後的盤面，而M_3之前連續兩步M_2^D和M_1^D都形成反手，則$I_{pre}(M_4) = \{M_1, M_2, M_3\}$，代表攻擊方在$M_3$之前無法自由選擇要從何處攻擊，被反手所限制攻擊的位置，直到M_4才有機會自由選擇要攻擊的位置。

$I_{post}(M_i)$表示M_i之後形成反手的連續 Moves，如圖 34。假設目前位置在M_1下完之後的盤面，M_1^D和M_2^D都將形成反手，則$I_{pre}(M_1) = \{M_1, M_2\}$。

針對$\psi \in CTSS$定義一個 Pre-Block 的 Set：

- $\mathbb{M}_{pre\text{-}block}$：$\{M_i \mid \exists\ M_j \in I_{pre}(M_i)$，$M_k \in I_{post}(M_i)$且$M_j \rightarrowtail M_k\}$。

簡單來講，假設目前是 Move M_i(圖 35)，若M_i之前為連續的反手，則攻擊方別無選擇，只能一邊阻擋反手一邊進攻，且這些攻擊方的 Move 不一定是互相相關的。只要其中某一步有可能擋住M_i之後連續反手的其中一個反手，因此即使M_i與其前一手並無相關，M_i仍是一個合理的攻擊步。

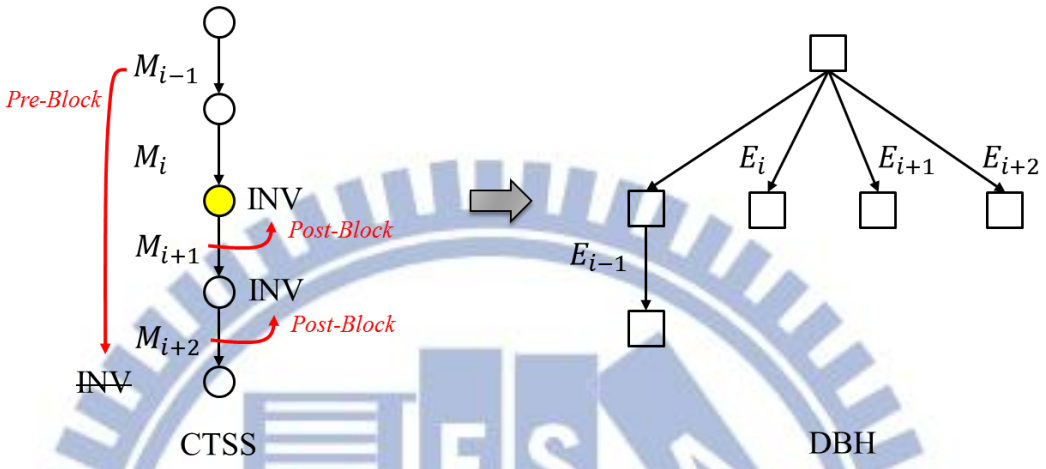


圖 35. Pre-Block 概念圖

雖然 Pre-Block 與 Post-Block 考慮了大部分的反手問題，但是要完整解決反手問題，需要相當大的複雜度，且設計極為複雜。因此本論文仍有一些反手情形沒有被考慮。

3.5.4 2ST 搭配

以目前的 TSS 搜尋，只要是在同一個盤面中的 ST 都能被互相搭配而形成攻擊方的候選步(Candidate Moves)。在大部分的情形，這些候選步已足夠找到必勝路徑，不需要特別到其他區域尋找額外的 ST 來搭配，但是在極少的情形下，只靠同一個盤面的 ST 來搭配是無法必勝的。必勝路徑內的 ST 無法在同一個盤面找到合適的 ST 來搭配的原因有：

1. 目前盤面只有一個死三。
2. 其餘搭配的ST將使得防守方造成無法阻擋的反手。
3. 其餘搭配的ST將使得防守方擋住必勝路徑。

以上的三項原因使得要完整解決2ST搭配問題，需要相當大的複雜度，且設計極為複雜。因此本論文仍有一些2ST搭配情形沒有被考慮。

3.6 實做流程

這個章節則實際討論 TSS 如何進行 DB-Search 的剪枝。剪枝的詳細流程如下：

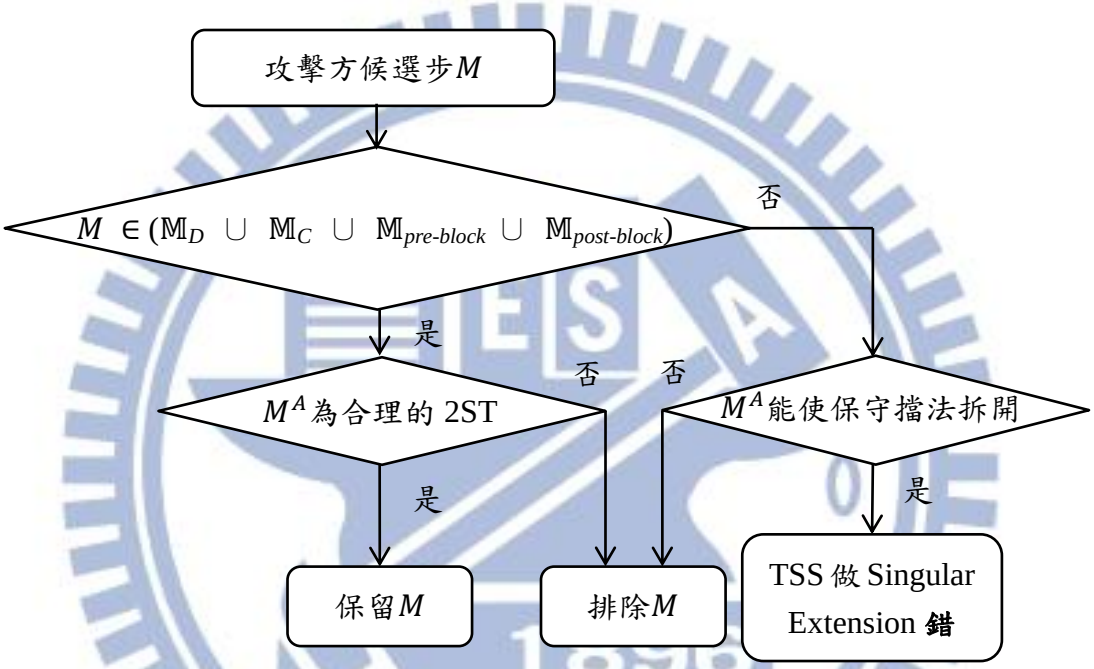


圖 36. TSS 剪枝流程圖

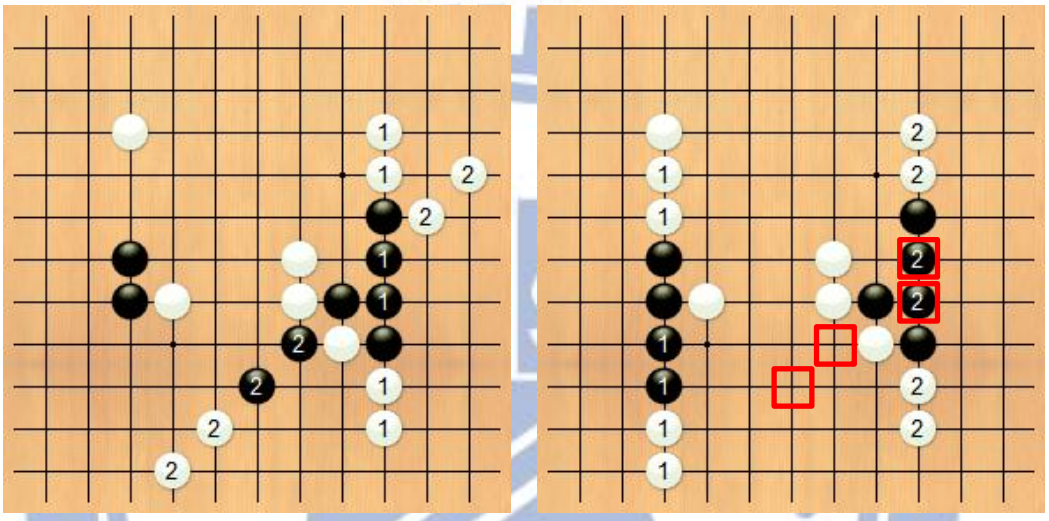
3.6.1 Dependency

要判斷一個 Move $M_i \in \mathbb{M}_D$，只需利用 3.3.2 定義的 Related Piece 性質檢查M_i是否相依於前一手即可。

3.6.2 Combination

由於 NCTU6 的 TSS 是 Iteration Deepening 的方式，即使沒有 Combination，各個區域仍會因為一開始的 Iteration 搜尋過而已經被建立在 DBH 內。要判斷$M_i \in \mathbb{M}_C$，需以 DBH 來輔助檢查。

- $Z_T(M_i)$：M_i未來的發展區域，從所有$E_i \in \mathbb{E}(M)$的 Subtree 所有攻擊方棋子的聯集。
- $Mi(M_i)$：由M^A建立的米字形區域的聯集。建立米字形區域的方法為：
  - 對每顆攻擊方的子，往八個方向延伸 6 顆子的長度的區域聯集。
  - 延伸時，一旦遇到防守方棋子，就停止延伸。
  - 若有某個方向的區域不足 6 顆，則不考慮此方向的線段。



(a) 之前搜尋過的區域　　(b) $Z_T(M_2)$

圖 37. DBH 紀錄的未來發展區域

$Z_T(M_i)$代表M_i未來的發展區域，圖 37 為未來發展區域紀錄的範例，可見圖 37.(a)從$P(M_1)$搜尋過右方的區域，未來若從左方開始搜尋(圖 37.(b))，且又下到同樣的位置時，可以從 DBH 中紀錄的區域得知未來的發展。

$Mi(M_i)$代表若攻擊方下子在此區域中，有可能與M_i產生新的攻擊線段，例如死二、活三等，如圖 38。攻擊方位於 C3 位置的棋子由於左上－右下方向的空間不足以形成連六，因此米字形不建立此方向的線段。

若存在$M' \in I_{pre}(M_i)$且$Z_T(M_i) \cap Mi(M') \neq \emptyset$，就代表$M_i$有可能屬於 set $\mathbb{M}_C$，表示M_i未來的發展區域有機會與之前的 Moves 去 Combine 形成新的攻擊線段。在此情形之下，M_i是一個合理的攻擊步。

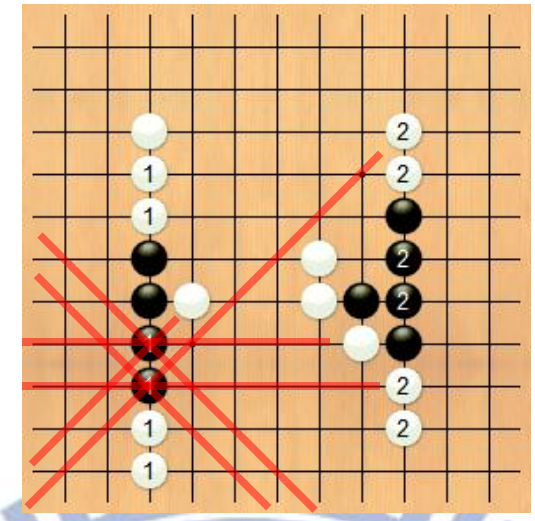
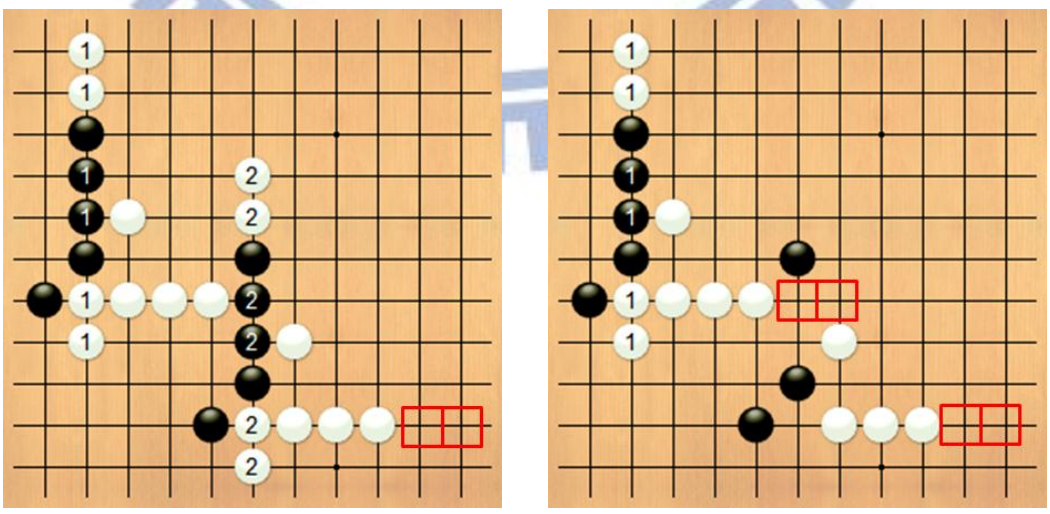


圖 38. 米字形區域範例：$Mi(M_1)$

3.6.3 擋反手

要判斷 Post-Block 是非常簡單的。但是要判斷 Pre-Block 則需要使用到 DBH 來紀錄可以擋住反手的區域。

- $D(M)$：一個區域，聯集了所有能夠擋住M^D形成的反手的位置。
- $Z_B(E)$：一個區域，聯集了所有 $D(M')$，$M' \in I_{post}(M_i)$且$E \in \mathbb{E}(M_i)$。DBH 中的每一個 Hyperedge 都記錄一個 Z_B，表示可以擋住未來產生的連續反手的區域。



(a) $Z_B(E_1)$　　(b) $Z_B(E_2)$

圖 39. Z_B 的範例

以上圖 39 的範例中，存在一個 CTSS Sequence ψ = <M_1, M_2>，且M_1和M_2分別對應到 Hyperedges E_1和E_2，則當 CTSS 搜尋到M_2時 $Z_B(E_2)$的區域為 $D(M_2)$，而且由於M_1也是反手，因此 $D(M_2)$也會與 $Z_B(E_1)$聯集，使 $Z_B(E_1)$更新為圖 39.(b)的方框區域，如此未來再次搜尋到M_1的位置時，便能從 $Z_B(E_1)$直接取得可擋住未來的連續反手區域，以判斷未來的連續反手是否已經被目前盤面上的棋子所擋住。

若$M_i \in \mathbb{M}_B$，則存在$M' \in I_{pre}(M_i)$，$E \in \mathbb{E}(M_i)$且 $P(M'^A) \cap Z_B(E) \neq \emptyset$，表示 Move M_i之前的攻擊方的棋子有機會擋住M_i之後的其中一個連續反手。

至於更新 Z_B 的方法，則是在 CTSS 搜尋時，一旦遇到反手的 Move M_i，就將可擋住這個反手的區域 $D(M_i)$去 DBH 中聯集所有之前造成連續反手的 Moves 的 Z_B。，如圖 40 所示。假設$E_1 \in \mathbb{E}(M_1)$，$E_2 \in \mathbb{E}(M_2)$，$E_3 \in \mathbb{E}(M_3)$，由於目前的 Move M_3造成反手，因此將區域 $D(M_3)$聯集到 $Z_B(E_1)$、$Z_B(E_2)$與 $Z_B(E_3)$。

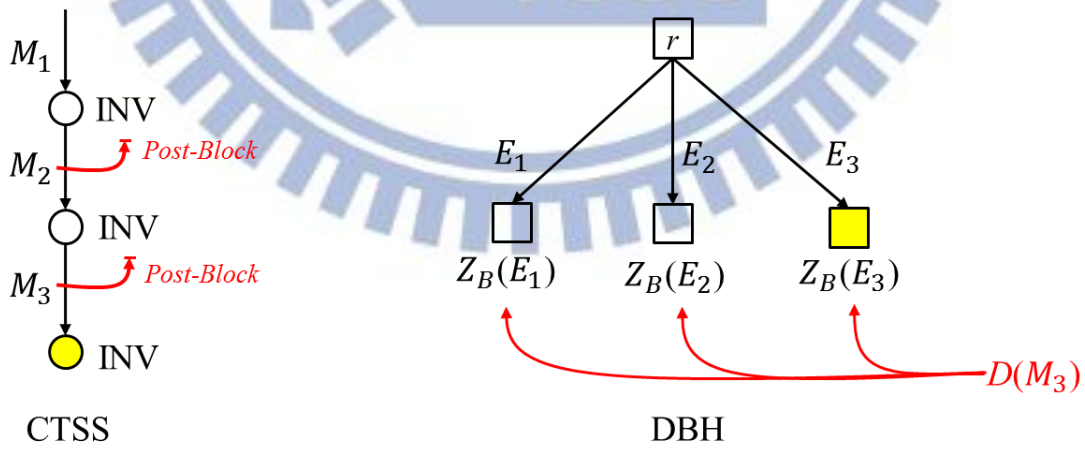


圖 40. 更新 Z_B 的範例

3.6.4 拆開保守擋法

要判斷M_i^A是否能使保守擋法拆開，首先目前的 TSS Sequence 中須至少存在一個保守擋法。接著檢查M_i^D是否會造成反手。若會，則讓 TSS 多搜尋一層，也就是 Singular Extension[3]，讓保守擋法因為造成反手而有機會被判斷為需要拆開。

3.6.5 合理的 2ST

最後是關於不合理的兩死三的判斷。以圖 41 作為範例：

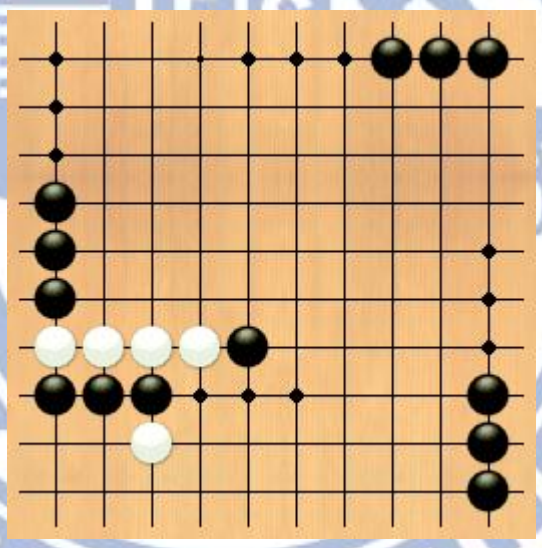


圖 41. 不合理的兩個死三範例

從這個範例可以看出，真正有機會發展的死三只有盤面左下角死三，例如 E3 或 F3，其餘三個死三只是用來搭配。但是在 TSS 搜尋時，若無法從盤面左下角的死三攻擊獲勝，則 AI 會繼續嘗試使用其他三個死三不同的搭配方法來繼續搜尋，造成時間上無謂的浪費。假設$P(M^A) = \{p_1, p_2\}$，$\mathbb{E}(M) = \{E^1, E^2\}$且$M^A \in \mathrm{ST_I}$，則若以下所有條件都成立，就將$M^A$從 CTSS 的候選步中排除。

- $T(M^A)$的所有死三已經被搜尋過。

  只要一個死三，其可能造成死四的其中一個位置被攻擊過，就將這個死三標記為攻擊過，降低重複搭配同一個死三的不同位置的狀況。

- $Z_T(E^{1A}) \cap Z_T(E^{2A}) = \emptyset$。

  這兩個死三未來的發展不會有任何交集，無法產生新的攻擊線段。

- $Z_T(E^{1A}) \cap Mi(E^{2A}) = \emptyset$

  E^1未來的發展區域無法與E^2的棋子直接合併產生新的攻擊線段。

- $Z_T(M^{2A}) \cap Mi(M^{1A}) = \emptyset$

  E^2未來的發展區域無法與E^1的棋子直接合併產生新的攻擊線段。

- p_1無法產生額外的攻擊線段，例如活二等。

- p_2無法產生額外的攻擊線段，例如活二等。

一旦利用以上的性質來進行判斷，在搜尋某些含有大量死三的盤面時，能夠排除非常多不合理的 2ST 候選步，有效地降低搜尋的時間，避免在一些不合理或重複的搭配中來回搜尋。

第四章、實驗

在這一章節中，將實驗由第三章所講解的方法。在第 4.1 節中，會介紹實驗所使用的設備、環境以及各種參數。在之後的章節中，會列出三種不同的實驗方法以及相關結果。

4.1 實驗環境

在 NCTU6 運行的系統，使用的是個人桌上型電腦，CPU 為 Xeon E31225，時脈為 3.1GHz，搭配 4.0GB 的 DDR3-1333 記憶體，運行的作業系統為 Windows 7 x64 版本。

在 NCTU6 參數的部分，之前有說明過 NCTU6 的 TSS 主要包含 VCDT 和 VCST。VCDT 的演算法是依據 Iterative Deepening Depth First Search 所設計，而 VCST 演算法內則會執行多次 VCDT。因此參數部分主要有三種：Iterative Deepening 的深度、單次 Iteration 的時間限制，以及 TSS 的總時間限制。時間限制的部分，主要是依據 Piece Count 來模擬時間，也就是 NCTU6 在模擬下子的數量。

用來實驗的盤面總共有 260 個，分別是從 LG[6]網站上或歷年來學長姐整理的一些困難的盤面。其中有些是 VCDT 的解答，有些是 VCST 的解答，也有無必勝解的題目。

4.2 原始參數比較

在第一個實驗中，將比較套用了 DB-Search 的 TSS 演算法和直接的 TSS 演算法。所有的參數都是依照 NCTU6 的預設參數，重要的有下列的參數：

- Iterative Deepening 深度最深 20 層，一旦找到必勝解就馬上結束。
- 單次 Iteration 的時間限制為 130,000 個 Piece Count，約 1.3 秒
- 總 TSS 的時間限制為 7,200,000 個 Piece Count，約 72 秒

實驗結果如下：

Run VCDT & VCST	Move Count	Time (Sec.)	Solve
Original Search	**62,865,372**	**4,678**	**180**
DB-Search	**45,717,611**	**3,719**	**199**
Speed Up	**1.37**	**1.25**	**19**

表 3. 原始參數的總比較表

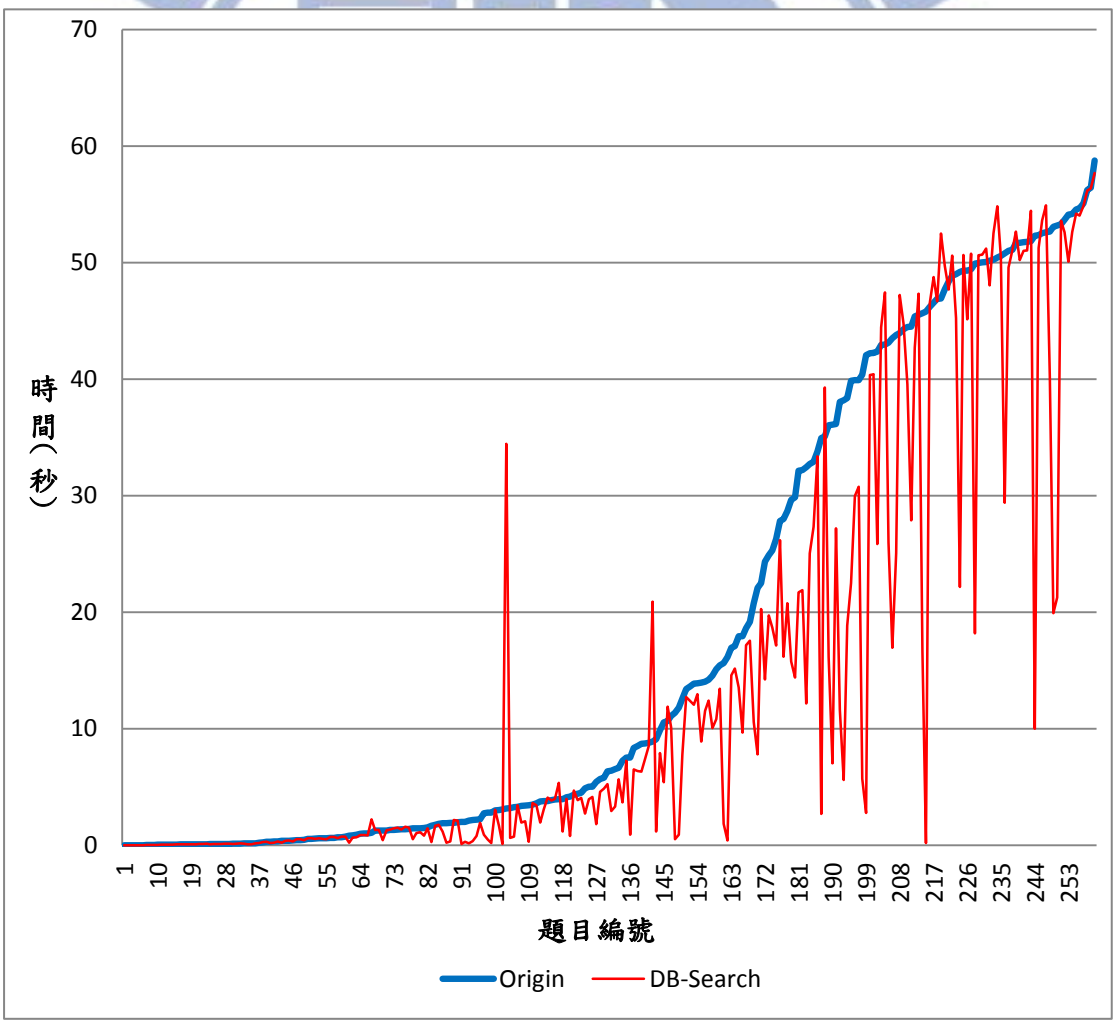


圖 42. 原始參數的比較圖(依照原始 TSS 花費的時間排序)

由比較圖可以看出，使用了 DB-Search 的 TSS 在時間上並沒有特別優異於原始版本的 TSS。在總比較表中也可以發現速度上快了原本的 TSS 約 25%，但是解出來的總題數卻多了 19 題。

時間上沒有太多的優化，主要原因是單次 Iteration 的時間限制是在每次 Iteration 完成才進行檢查。如果原始版本在第 N 次的 Iteration 結束後會因為 Piece Count 超過 Limit 而中止搜尋，而 DB-Search 的 TSS 卻因為減少搜尋一些不必要的盤面，使得 Piece Count 在第 N 次 Iteration 結束後未超過 Limit，進而開始搜尋第 N+1 次的 Iteration，則在時間花費上就有可能大大地超過原始版本的 TSS。

4.3 調整參數比較

在第二個實驗中，將調整一些參數，使得在解題數量接近相同的情況下，盡量壓低所需時間。調整的有下列的參數：

- Iterative Deepening 深度最深 20 層，一旦找到必勝解就馬上結束。
- 單次 Iteration 的時間限制為 70,000 個 Piece Count，約 0.7 秒
- 總 TSS 的時間限制為 3,800,000 個 Piece Count，約 38 秒

實驗結果如下：

Run VCDT & VCST	Move Count	Time (Sec.)	Solve
Original Search	**62,865,372**	**4,678**	**180**
DB-Search	**30,007,630**	**2,620**	**181**
Speed Up	**2.09**	**1.78**	**1**

表 4. 調整參數的總比較表

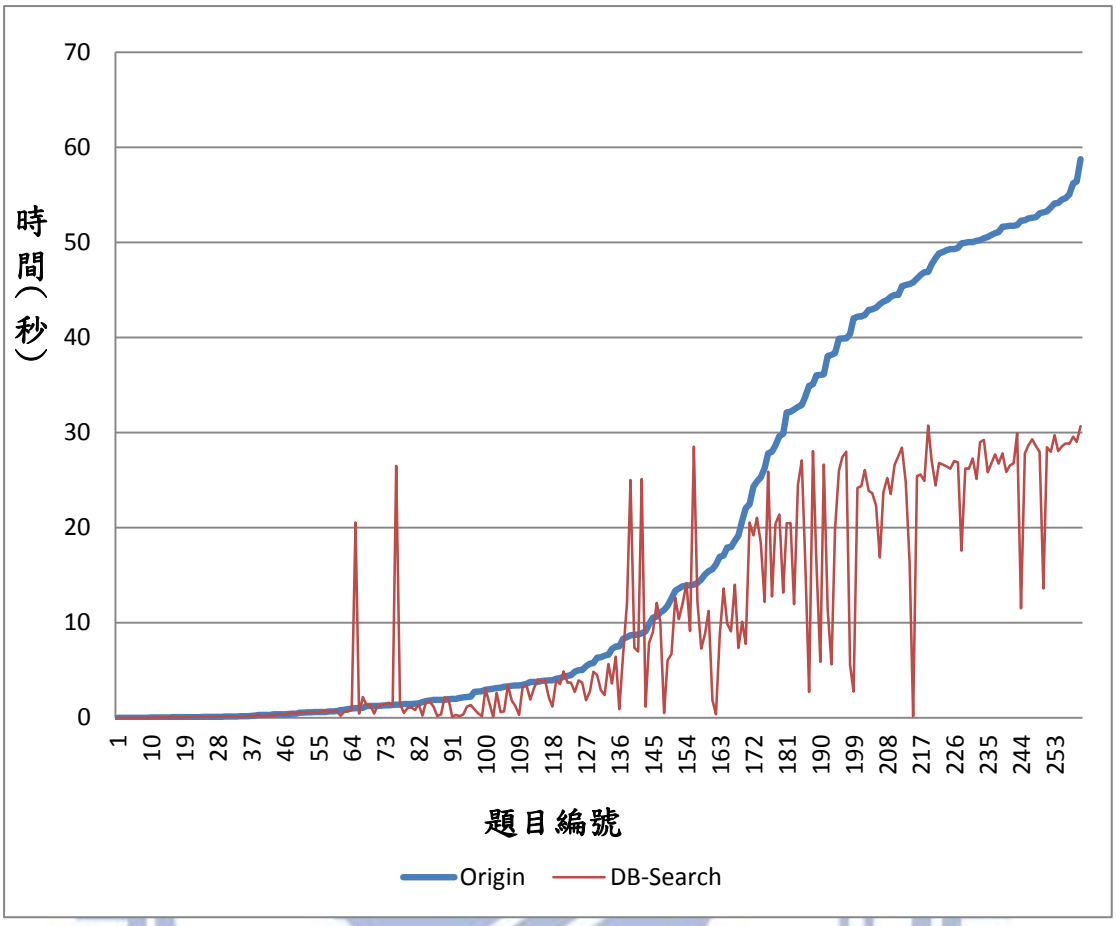


圖 43. 調整參數的比較圖(依照原始 TSS 花費的時間排序)

由比較圖可以看出，使用了 DB-Search 的 TSS 在時間上大約是原始版本的 TSS 的 1.8 倍快，而解出的題目數量只多解一題，幾乎是相同的。

4.1 公平參數比較

在第三個實驗中，為了公平且完整地比較效能上的差異，將把一些變因去除來進行實驗。參數設定如下：

- Iterative Deepening 深度最深 20 層，一旦找到必勝解就馬上結束。
- 不限制時間
- 只進行 VCDT 的搜尋，減少 VCST 所造成的變因

在 VCST 的搜尋中，所有的 Single Threat 都是在 VCDT 搜尋時蒐集而來，放入一個 Queue 中，依序取出進行 RZone 的搜尋。因此若 VCDT 搜尋的路徑有所不同，蒐集到的 Single Threat 也會有差異，造成比較上的不公平性。因此在這個實驗中，將參數設定為只進行 VCDT 搜尋。實驗結果如下：

Run VCDT	Move Count	Time (Sec.)	Solve
Original Search	**45,536,423**	**2183**	**72**
DB-Search	**7,169,103**	**507**	**72**
Speed Up	**6.35**	**4.30**	**0**

表 5. 公平比較的總比較表

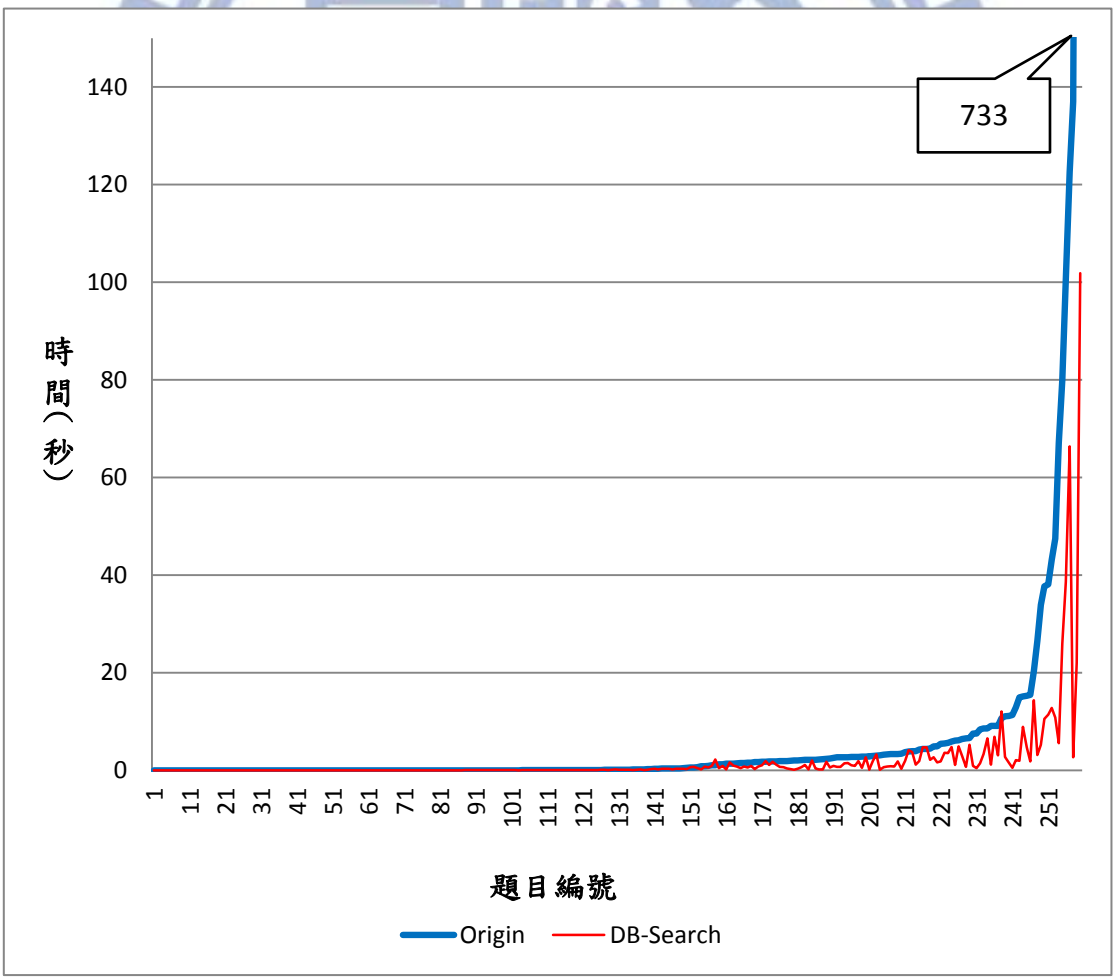


圖 44. 公平參數的比較圖(依照原始 TSS 花費的時間排序)

從比較表格中，可以看出使用了 DB-Search 的 TSS 在時間上平均是原始版本的 TSS 的 4.3 倍快，而解出的題目數量是相同的，只要是存在 VCDT 必勝解的題目都可以在時間沒有限制的情形下被解出。但是，由於有一半以上的題目是 VCST 必勝解的題目，因此 VCDT 幾乎都是不到 1 秒便結束搜尋，導致曲線圖中有不平均的現象。另外，有些搜尋複雜度非常高的盤面，也能利用 DB-Search 的 TSS 有效降低搜尋時間，有些題目甚至能夠加速到 50 倍以上。

最後計算 Overhead。由以上三個實驗，原始版本的 TSS 總共搜尋了 171,267,167 個 Moves，花費了 11,539 秒。而 DB-Search 總共搜尋了 82,894,344 個 Moves，花費了 6,846 秒。可以比較出在 Move 數量的比較方面，DB-Search 約為原始版本的 TSS 的 0.48 倍，而時間的比較卻是 0.59 倍，因此 Overhead 大約在 22.58%左右。

第五章、結論與未來展望

在本篇論文之中提出了適用於六子棋的 DB-Search 演算法，以及其相關的定義、符號和實作方法。此外，有別於 Allis 所提出的 DB-Search，將所有盤面都記錄在記憶體中的演算法，此篇論文提出的 DB-Search 演算法能應用於深度優先搜索(Depth-First Search)的迫著空間搜尋演算法，增加 DB-Search 的應用範圍。對於六子棋迫著空間搜尋的問題，本篇論文提出的 DB-Search 主要能夠解決以下幾種：

- Dependency，相關攻擊問題
- Combination，合併問題
- 2ST 搭配問題
- 擋反手問題

從實驗的結果可以看出使用了 DB-Search 的 TSS，在調整過參數之後較原本的搜尋快了 1.7 倍，解出的題目也多了約 20 題。在完全公平的比較之下，快了 4 倍之多，甚至是一些較為困難的盤面，這些原本的 TSS 需 10 秒以上搜尋時間，DB-Search 加速的倍數甚至超過 5 倍之多，成功改良了 TSS 演算法的搜尋效率。另外我們也測量了套用 DB-Search 的 Overhead，用三項實驗總 Move 數的改進效率與總時間的改進效率，大約是 22.58%。

在未來展望的部分，主要有兩項工作需繼續研究與加強。由於 DB-Search 在 2ST 的搭配以及反手的判斷上仍非完美解決所有的情形，雖然這些情形非常罕見，在 260 題測試題目中並無影響，目前無法確定是沒有這些情形存在，抑或是剛好有其他的必勝路徑，但這些未來仍需解決。繼續降低 Overhead 也是其中一項未來需要繼續加強的部分。

參考文獻

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

[3] Thomas Anantharaman, Murray Campbell and Feng-hsung Hsu. "Singular Extensions: Adding Selectivity to Brute-Force Searching", Artificial Intelligence, vol. 43, no. 1, pp. 99-109, 1990.

[4] T. Cazenave, A Generalized Threats Search Algorithm. Computers and Games, Vol. 2883 of Lecture Notes in Computer Science, pp. 75–87, 2003.

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

[6] Little Golem website. Available at http://www.littlegolem.net/

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

[8] Renju International Federation, The International Rules of Renju, http://www.renju.net/study/rifrules.php

[9] Taiwan Connect6 Association, Connect6 Homepage, available at http://www.connect6.org/

[10] T. Thomsen, Lambda-search in game trees - with application to Go, ICGA Journal, Vol. 23 203–217, 2000.

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

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

[13] 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.

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

[15] 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.

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