國立臺北大學資訊工程學系專題報告

麻將 AI 之研究與實作

THE MAHJONG AI

專題組員:王淳暘、陳亮筑、劉得恩、楊智凱

專題編號:PRJ-NTPUCSIE-112-011

執行期間:2023 年 09 月 至 2024 年 05 月

1.摘要

本專題實作麻將 AI，主要著重於捨牌決策的演算法及審局函數之數學模型研究，並加速程式，以提升程式於比賽中有限時間內的表現。

2.簡介

在麻將的遊戲中，當我方摸牌或吃碰牌後，由於手牌增加一張，因此需要捨棄一張手牌，以保持手牌數量維持在(3n+1)，此時的麻將 AI 便會進入捨牌環節。

麻將本身為機率型遊戲(Stochastic game)，對手及我方每輪摸進的牌皆為變數，盤面的可能性極多。若用一般的樹狀結構進行搜尋會導致樹的大小過於龐大。因此本專題使用蒙地卡羅樹搜尋(Monte Carlo Tree Search, MCTS)，用模擬的方式代替大量的子節點拓展，並些微修改樹的架構，使演算法能更合適地應用於麻將遊戲中。

除此之外，本專題亦以機率為基礎，發展出一套能快速逼近統計結果的數學模型作為審局函數，以求得更加準確的審局結果。

然而，AI 功能實作完畢後，我們發現程式效率不盡理想，導致演算法無法於時限內發揮最大的效果。分析原因後，本專題以同形表(Transposition Table, TT)省略重複的節點模擬，並且使用 CUDA 將程式平行化，大幅提升程式效率，更使蒙地卡羅樹的演算法能有更佳的表現。

3.專題進行方式

3.1 程式架構

程式主要被分為以下五個部分管理：

◆**自家暗牌管理**：紀錄我方的暗牌資訊，包含手牌及眼牌。

◆**剩餘未知牌管理**：管理所有場上尚未出現的牌。用以計算場上每張牌出現的機率或模擬抽牌及對手捨牌後場上的情況。

◆**各家明牌管理**：管理各玩家吃、碰、槓的組合。

◆**牌海管理**：紀錄所有玩家捨棄的牌，以實行防放槍機制。

◆**玩家行為管理**：玩家的決策分析，包含吃、碰、槓牌與捨牌選擇。

3.2 開發工具

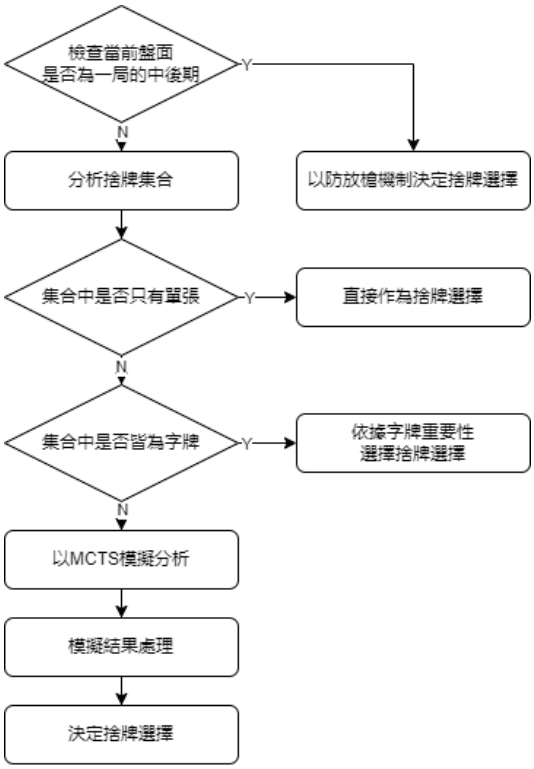
本專題程式的麻將 AI 用 C++撰寫而成，而開發工具為 Visual Studio。

3.3 捨牌決策與 MCTS

於對局中，我方和其他玩家的吃、碰、槓牌策略主要採用機率與進胡數計算的方式粗略估計並做決策。然而，捨牌選擇對麻將而言至關重要，因此需要更精準的分析，相關流程見圖一。

當牌局進入到中後期時，多數玩家會進入聽牌階段。此時若我方尚未進入聽牌階段，能胡牌的可能性不如其他玩家，如果沒有考慮手牌的放槍風險而仍以追求胡牌為目標，便會導致非常高的放槍率。因此麻將 AI 中的防放槍機制主要以跟打、分析打出後被吃碰風險的方式，希望在牌局不利於我方時盡可能降低放槍率，減少

失分。



圖一：捨牌環節之流程圖

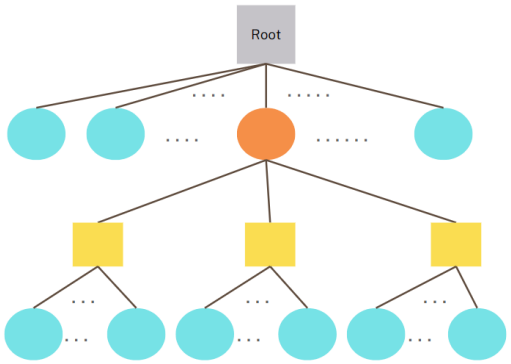
若對局尚未進入中後期，則麻將 AI 將以追求己方胡牌為目標進行分析。首先，為了避免將模擬資源浪費於明顯不理想的捨牌選擇，如已能湊齊一組的牌，本專題分析人類打牌的邏輯，將其以數學方式實作，粗略地從手牌中挑選出幾張擾亂手牌有序性的牌作為捨牌集合，以進行下一步模擬。其中挑選捨牌集合的方式如下：

現實中，人類不僅會思考場上尚未出現的牌對手牌的影響，也會考慮每張手牌是否有機會能和附近的手牌以許多不同的組合結合。因此對於每張手牌，本組會考慮能用該牌組成的所有可能組合，個別計算湊成組的機率並加總。將此機率總和定義為該手牌之 score。

例如：若手牌有二萬、三萬、四萬，計算二萬的 score 時會考慮其與其他手牌支組合。二萬可與手牌中三萬及四萬組成完整的一組，也可與三萬組成搭子，亦可與四萬組成中洞搭，還能將本身視為孤張。根據這些組合算出能成組的機率並加總即為二萬的 score。

得各張暗牌的 score 後，會依據 socre 低至高將暗牌排序。排序越靠前代表該牌對手牌來說越不重要，因此會將與排序第一位與和其無顯著差異的牌都納入捨牌集合，以盡量減低 score 衡量本身的誤差。

捨牌集合挑選完畢後即進入 MCTS 的環節。MCTS 由 Selection、Expansion、Simulation、Back-Propagation 四步驟組成，經過多次迭代形成蒙地卡羅樹。由於本專題將 MCTS 應用於捨牌策略，因此每往下拓展一層，皆代表捨棄一張暗牌。若一味地拓展，將導致手牌數量不斷減少，限制了樹的深度，也有失模擬準確性。為了解決此問題，本組在每層捨牌節點之間插入抽牌節點，每個抽牌節點代表一張有可能摸進的牌。示意圖見下方圖二。



圖二：本麻將 AI 中 MCTS 架構示意圖。其中圓形節點為捨牌節點，方形節點為抽牌節點。

經過不同的嘗試與測試，AI 麻將中 MCTS 每步驟的實作細節如下：

◆Selection:

使用 upper confidence bound(UCB)選擇該輪將要拓展的節點。UCB 公式如式一：

$$UCB_i = \frac{W_i}{N_i} + c \times \sqrt{\frac{\log N}{N_i}} \qquad \text{（式一）}$$

其中 W_i 為節點 i 之模擬總分；N_i 為節點 i 之模擬次數；N 為整棵搜尋樹的

總模擬次數。

UCB 的公式可分為開發項 $\frac{W_i}{N_i}$ 和探勘項 $c \times \sqrt{\frac{\log N}{N_i}}$。開發項為該節點模擬之平均分數，而從探勘項的計算式可推得當節點 i 模擬次數越小，其探勘項將越大。因此以 UCB 作為選擇標準能使模擬表現普通但得到較少模擬資源之節點能有機會被探勘，進而提升模擬準確度。其中探勘速度由常數 c 控制。

◆Expansion:

由於第一輪的拓展從樹開始，此時手牌已經經由吃碰牌或摸牌多得一張，因此不需要拓展摸牌節點。第二輪之後的拓展，為確保手牌數量的平衡，每次拓展皆會先根據場上尚未出現的牌拓展抽牌節點，從捨牌節點走訪到抽牌節點時代表手牌多了一張，此時會計算在該狀態下之捨牌集合，依捨牌集合再從抽牌節點拓展一層捨牌節點。

◆Simulation:

重複模擬從葉節點捨牌與摸牌直到遊戲結束。由於抽牌本身帶有機率性，因此每次模擬結果不盡相同。每次模擬之分數將根據模擬盤面結束時的結果決定，其中胡牌得 1 分，自摸得 3 分，流局得 0 分。

本程式基於時間考量，不模擬對手的打牌策略，所有對手皆打出摸到的牌，故對手不會胡牌，模擬分數為介於 0 和 3 的數而不會為負。經測試後本組發現對手以摸打模擬即能有效提高勝率，因此以此方式降低模擬時間。

◆BackPropagation:

由於本組調整 MCTS 之架構，因此在往回更新分數之部分亦做出了相對應調整。

本組將抽牌節點之分數定義為其下方所有捨牌子節點分數之平均，而非所有捨牌子節點之最大值，以減低模擬誤差。而抽牌節點具有走訪機率，即摸到該牌之機率。因此得一層中所有抽牌節點的分數後，將計算出該層期望值，再往上更新至抽牌層。如此反覆更新至根節點。

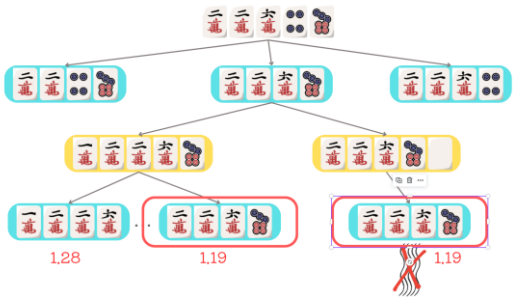
MCTS 結束後，AI 僅會在最高模擬分數與其他有顯著差異時採用模擬結果，否則將根據相近的捨牌選擇進行經驗法則分析，以適當避免模擬誤差。

3.4 同形表

於實作時我們觀察到於搜尋樹中，相同或相近的節點可能由不同的捨牌抽牌路徑到達，即實際上搜尋的過程為一個對局圖而非一對局樹，因此可使用雜湊表作為資料結構，建構同形表將先前已得到的資訊重複使用。

麻將可觀察的盤面資訊龐雜，同時包含各玩家的明牌、牌海和我方手牌。而經由實測可發現，不同的盤面但相同的手牌時常會得到相近的捨牌選擇，即影響捨牌策略最顯著的因子為我方的手牌，因此我們於實作時將我方手牌相同的情況視為相同的盤面。

每次進行模擬前，程式會先到同形表中確認該節點的盤面是否已被模擬過，若沒有則於模擬後將結果儲存回同形表中；若該節點已被模擬，則直接將儲存於同形表中的結果回傳，節省模擬的時間。



由於搜尋時展開的節點數量極大，因此如何有效率的索引至同行表中的位置以供程式快速地讀出、寫入結果便顯得至關重要。我們使用了 Zobrist hashing 來快速地將手牌轉為 hashkey，藉此得到同行表中對應

的位置。

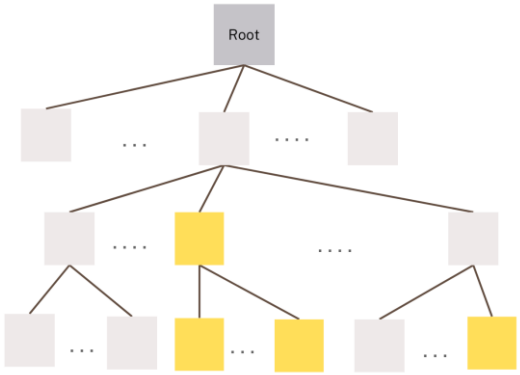
3.5 審局函數

於此先定義進胡數為「至少需再進幾張牌使得手牌可以胡牌」，即於最理想的進牌方式中，仍需幾次進牌才能胡牌。於需要審局時，如判斷是否吃碰槓，過往的程式往往會使用進胡數作為判斷的依據，並搭配事先建表進行加速。但事先建的表並無法依據於對局中獲取的資訊做出調整以得到更精確的結果，甚至可能會錯誤的等待一張早已無法取得的牌。此外，由於進胡數本身的數值過於離散，難以表達牌局連續性的變化，因此我們引入了一個基於機率的審局函數—胡牌機率，並透過事先建表和平行化的運算來快速地進行審局。

理論上而言，要求得於對局中特定時間點能夠胡牌的機率，只需以樹狀搜尋依序展開捨牌層與抽牌層即可。然而由於麻將本身的特性使得搜尋的 branching factor 過大，難以於有限的時間內展至足夠的深度。因此我們將問題簡化，於計算胡牌機率此一近似機率時，只讓玩家持續的摸牌而不進行捨牌，即玩家的手牌數量將持續上升，當玩家的手牌包含可胡牌的牌型時，我們便稱其為可胡牌的節點。

令胡牌機率P_k´為摸入 k 張前可胡牌的機率，展開至深度 k 時，透過簡單的 counting 便可求得近似的胡牌機率。由於只進牌不捨牌，代表父節點手牌的多重集必然包含於其子節點中，因此若某節點為可胡牌的節點，其所有子節點必然亦為可胡牌的節點。據此我們可以歸納出胡牌機率P_k´的兩個特性:

1. 胡牌機率為摸牌數的遞增函數
2. 胡牌機率為實際胡牌機率的上界



圖三:胡牌機率概念圖。黃色的節點為包含可胡牌牌型的節點。與圖二相比，圖三僅含方形節點—即抽牌節點。可以注意到，其中可胡牌的節點之所有子節點亦為可胡牌的節點。

定義了胡牌機率P_k´為摸入 k 張前可胡牌的機率之後，我們便可得到P_k如式二，為恰於摸入 k 張時胡牌的機率。

$$P_k = \begin{cases} P'_k - P'_{k-1} & \forall k \in \mathbb{Z}^+ \\ 0 & k = 0 \end{cases} \quad \text{（式二）}$$

有了P_k之後便能依期望值的概念得出平均胡牌數H_k´如式三，其意義為我們期望中恰於摸第幾張時胡牌。

$$H'_k = \sum_{x=1}^{k} x \cdot P_x \quad \text{（式三）}$$

然而，給定摸 k 張時P_k´並不一定等於 1，即最後不一定可胡牌，且不同盤面間的P_k´亦往往不相同。為求不同盤面間的可比性以避免錯誤的審局結果，我們應將樣本空間補至 1。修正後的平均胡牌數H_k如式四，其中常數 C 可視為為於 k 張前無法胡牌的風險或懲罰項，其合理的值應>=k。

$$\begin{aligned} H_k &= C \cdot (1 - P'_k) + \sum_{x=1}^{k} x \cdot P_x \\ &= C \cdot (1 - P'_k) + H'_k \end{aligned} \quad \text{（式四）}$$

有平均胡牌數之需求來自於麻將是 4 人遊戲，我們除了在乎我們最終能否胡牌，也就是胡牌機率P_k´的

意義，同時也會希望能盡快、比其他玩家更早胡牌，而平均胡牌數便能將此考慮在內。

最後由條件機率的概念可再導出條件胡牌數H_k^{cond}如式五，意義為於可胡牌的前提下，預期中的所需的期望進牌數。其能夠同時考量盡快胡牌和最終能否胡牌這兩個指標。

$$H_k^{cond} = \frac{\sum_{x=1}^{k} x \cdot P_x}{P_k^{'}} = \frac{H_k^{'}}{P_k^{'}} \text{(式五)}$$

3.6 主要困難與解決之道

由圖二之架構可推得，因為抽牌節點的加入，使樹的大小遽增。尤其於牌局剛開始時，幾乎所有牌種都有可能出現，樹總是十分龐大，大幅降低搜尋效率。若未來使用此AI比賽，比賽中現實的機制將導致MCTS迭代次數不足，甚至連搜尋一輪的時間都遠超出限制時間，嚴重影響決策的準確性。

原程式中模擬的部分以 thread pool 平行進行。為了解決此問題，本組進一步更改程式，將模擬的部分交由 GPU 執行。程式中這部分所使用的語言為較底層的CUDA C，將資料手動載入到 GPU 的記憶體，經過平行運算後再取回至 CPU 之記憶體進行處理。雖然 GPU 核心效率較差，但其極高的的核心數仍能大幅提升整體模擬效率。

4.主要成果與評估

本專題測試AI使用4次迭代的MCTS，每個葉節點模擬次數為兩萬次。

◆勝率測試

	自摸數	胡牌數	放槍數	勝率
本組程式	187	596	497	52%
對照程式	148	463	562	41%

對照程式為三年前麻將TAAI比賽中獲得銀牌的程式，因此結果具有足夠的參考價值。

其中勝率總和不為100%之原因為1500場中有106場留局。扣除流局部分後本組程式勝率為56.17%，對照組勝率為43.83%。

根據標準差公式：

$$\sigma = \sqrt{\frac{P \times (1-P)}{N}}$$

可計算得標準差為1.33%，本組程式與對照程式之勝率差距為13.67個標準差，因此本組對於勝率的提升有超過99.9%之信心。

◆搜尋平均時間測試

	平均搜尋時間
MCTS with thread pool	37.97sec
MCTS with CUDA	2.84sec

本組CPU使用i9-9900K，而顯卡使用RTX 2070 SUPER。可明顯發現使用CUDA加速後的程式平均搜尋時間大幅縮短，順利解決演算法的效率問題。

5.結語與展望

總括來說，我們於此次專題中實作了MCTS使得勝率與其他程式相比有了顯著的提升，並使用了CUDA進行加速使得AI程式能於時限內進行更多次的模擬已得到更準確的結果，同時使用同形表避免重複模擬節點，節省模擬的資源。此外，也引入了胡牌機率以得到更準確的審局結果。

在未來針對MCTS的模擬部分，若是能夠找到足夠快速並相較摸打更為準確的對手模型，則能於模擬的過程中使對手的決策更加合理，得到更為接近實際牌局的模擬結果。針對程式效能的部分，若能夠找到更適合的方式使用同行表，搭配CUDA更進一步的最佳化，便能於同樣的時限內進行更多次的模擬。於審局函數的部分，

則希望找到能同時考量各種於牌局中玩家能進行的行動，於足夠有效率的同時得到更準確的審局結果。

6. 銘謝

感謝指導教授於專題的製作過程中給的建議與幫助，引導我們進行方向的同時保留足夠多的空間讓我們嘗試各種可能的方法，除了定期的討論外也撥出了額外的時間協助專題的進行。也感謝實驗室的學長們，於製作中遇到問題時與我們進行討論，提供了許多不同面向的想法。

7. 參考文獻

[1]徐讚昇等著,2017,《電腦對局導論》