臺中一中/陽明交通大學科學班

個別科學研究成果報告

2048AI 決策樹樹搜尋開展與剪枝探討

學年度： 111 學年度

學 生： 游承熹

指導教授審核通過簽名：

中 華 民 國 112 年 4 月 13 日

目錄.

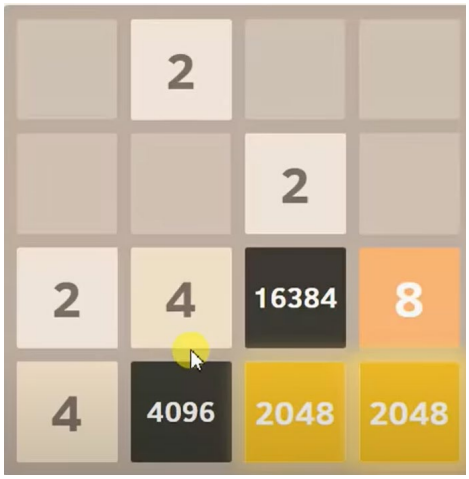
壹、前言

延續高中兩年的專題有關棋類等對稱公平透明賽局，有感於決策樹在現今類神經網路當紅而較少被討論，看重其潛在的研究價值，決定跨入具隨機性的 2048 遊戲更深入的探討有關決策樹的不同搜尋方法，以期大幅提升決策樹的搜尋速度與能力，也可以找到更多相關的決策樹應用。

2048 遊戲描述:

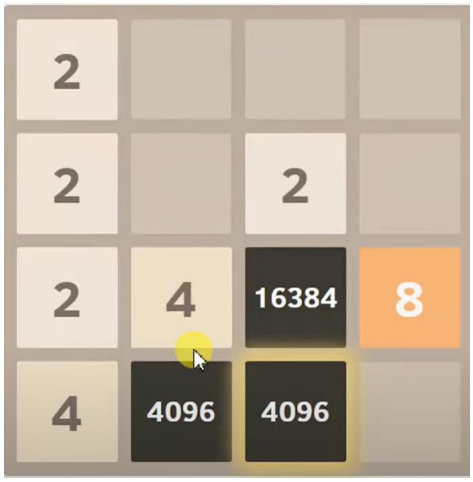
該遊戲每次操作可使方塊整體以上下左右其一方向移動。如果兩個帶有相同數字的方塊在移動中碰撞，則它們會合成為一個方塊，其所帶數字變為兩者之和。每次移動時，會有一個值為 2 或者 4 的新方塊出現(機率為 0.9、0.1)，所出現的數字都是 2 的冪次，當合成時，即得到方塊分數，例:將兩個 2048 方塊合成為 4096，則分數+4096 分。

圖示:



圖一、2048 遊戲介紹圖示(合成前)

合成 2048 後



圖二、2048 遊戲介紹圖示(合成後)

貳、研究目的

現今的人工智慧研究主流常常圍繞著類神經網路學習，然而決策樹其實有其應用的價值，比起類神經網路容易造成的黑盒子問題(容易形成 garbage in,garbage out、對於 input data，output 皆處於不了解)，決策樹往往可以在清楚的邏輯架構中討論問題，速度上也可以比類神經網路模型快上許多，因此本次研究將依現有的評估函數，進一步討論有關決策樹開展和剪枝的方法以提升現有電腦對局程式的效率和分數，期望能找到提高決策樹效率的普遍方法。

參、文獻探討

在決策樹的文獻探討中，我也發現有論文研究結合決策樹和類神經網路學習，預測美國大聯盟職棒比賽賽果(Valero,2016)，在一些問題上決策樹可以比類神經網路等人工智慧更精確及快速地處理問題。在賽局理論下的基礎下延伸討論，現實中出現的問題也可以使用賽局理論解決，只要找到適當的轉化方式將問題轉化成決策樹型式解決。

而在 2048AI 的文獻探討中，查閱了以往 2048 相關的論文，發現有(Szubert,2014; Matsuzaki,2016; Yeh,2016;)等論文有效率與精確性很高的盤面評估函數，但主要探討的偏向機器學習方式的細節，如盤面評估函數的制定，學習方式，學習率(learning rate)等細節較多，而決策樹僅在實驗機器學習效果時被提及，而樹搜尋方法往往也是最簡單的 expectimax，缺乏剪枝法的討論，因此本研究希望可以利用文獻中良好的盤面評估函數有效剪枝達到良好的效果，提升樹搜尋效率，減少不必要的枚舉，以達到更快更好的決策樹。

決策樹是一種樹狀圖，在樹上遍歷順序為由現有情況的父節點連到可能的情況的多個子節點，是一種良好的決策輔助工具，也可以用來做數據挖掘，對數據做分類，其中一種著名進階應用隨機森林可透過多種不同類型的決策樹的輸出綜合判斷的演算法。

肆、研究方法

此次研究採用賽局討論中利用決策樹常用的結構，包含決策樹、盤面評估方式、minimax 演算法。將賽局拆解成一步步行動，並模擬每次決策者做出不同行動開展不同局面的子節點，

由於 2048 複雜性較棋類遊戲較高，盤面評估函數決定使用 n-tuple network 結合 temporal difference (TD) learning 和 multi-stage TD (MS-TD) learning。此技術是將盤面特定位置劃分成一個特徵(feature)，以此得到許多個不同的 feature，一個 feature 中不同的方塊之間的大小關係、位置等會透過加權回傳一個回饋值(value)，將不同 feature 經過特殊加權後得到一個預測值預測此盤面的「潛力」-在適當操作後可得的預估最終分數，feature 之間的加權權重則會透過機器自我不斷遊戲以強化式學習型式修改優化 feature 的加權模式，在對局後將盤面評估函數更新的訓練公式為:

$V'_j(S_t) = V_j(S_t) + \alpha * \Delta$

r = 盤面操作後的分數-盤面操作前的分數

St =當前盤面

S′t =當前盤面做最佳操作(基於評估函數)後，未出現新方塊的盤面

V = 對局程式的盤面評估函數

Vj = 對局程式的盤面評估函數的其中一個 feature

$\Delta = r + V(S't) - V(S't-1)$ (回饋值)

α = 學習率，避免 AI 過度學習

如此反覆對局即可訓練出一個好的評估函數模型。

而本次研究主要著重的研究面向是剪枝法的討論，以原先最基礎的 expectimax 演算法作為對照組，提出從賽局觀察中發想直觀的 limit_popup 與從 alpha-beta 剪枝法中得到啟發的 startimax 剪枝法，並比對其數據與 expectimax 相比成果。

本次實驗中的分析數據取自電腦對局程式在 100 場中得分表現與耗時作為判斷依據。另一方面，在分析實驗結果上，在初步檢測後發現各種剪枝法在 1-ply 和 3-ply(即樹搜尋深度為 1 和 3)時表現皆不明顯，推測是決策樹大小本身不足，造成程式運行時常數影響大於剪枝演算法所致，故以下研究結果皆以 5-ply(深度 5)為研究目標。

以下結果分析將會從三個面向探討:

1.平均分數

2.每局方塊出現分布

3.程式運行時間

，本次研究因為著重於剪枝法提升效率，故目標在於維持機器人分數與原 expectimax 算法相同，並降低耗時達到樹搜尋效率的提升。

伍、結果與討論

一、expectimax:

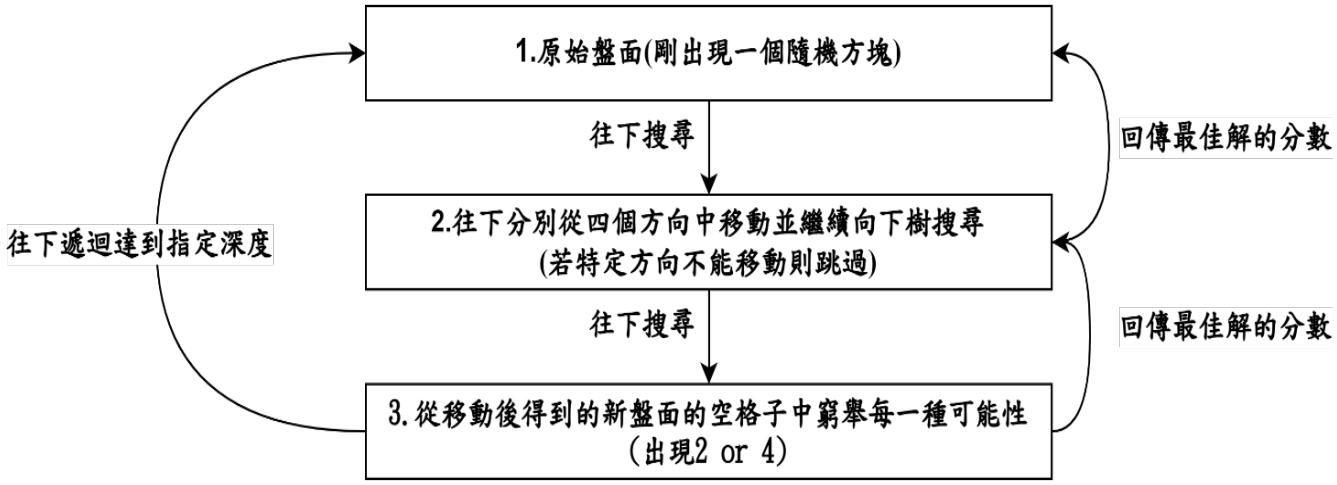
演算法介紹:

基於樹搜尋概念和機率，往下樹搜尋模擬每種情況並以機率加權找出最佳移動方塊方式

搜尋流程:

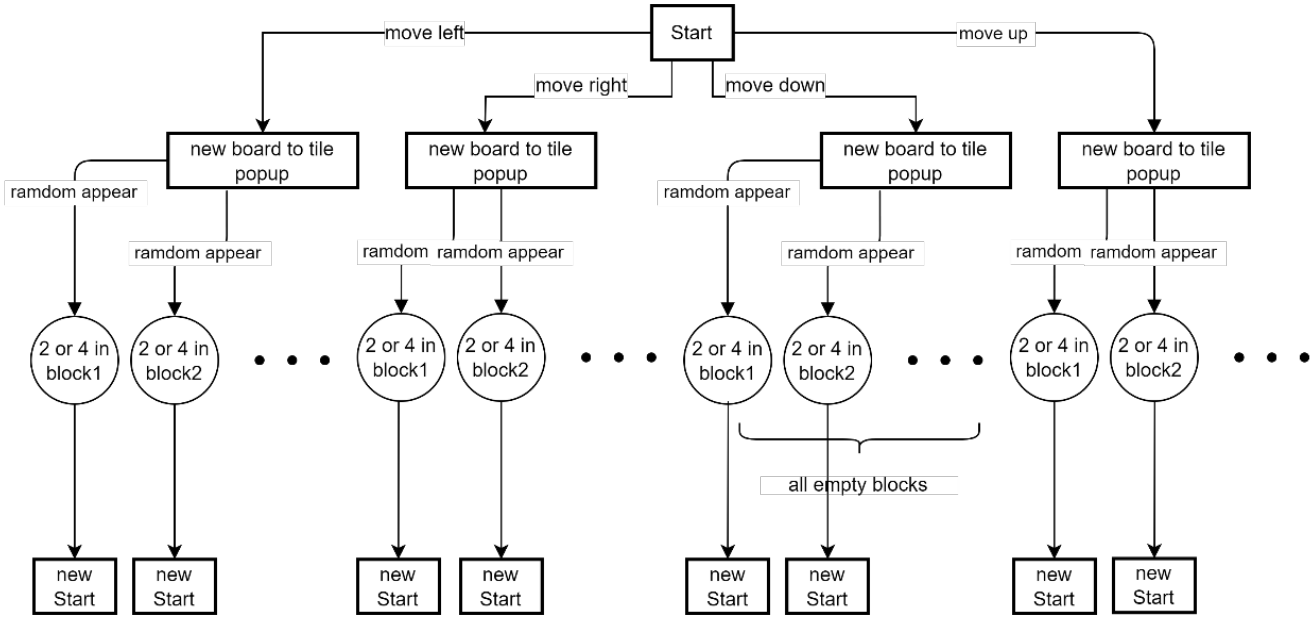
1. 原始盤面(剛出現一個隨機方塊)
2. 往下分別從四個方向中移動並繼續向下樹搜尋(若其中一個方向不能移動則跳過)，並回傳最好的一步的分數
3. 從移動後得到的新盤面的空格子中窮舉每一種可能性(出現 2 or 4)，
   依盤面出現機率(出現機率方塊 2:方塊 4= 9:1，再依方塊出現在不同方格得到的分數取平均)加權回傳此盤面得分期望值
4. 返回 1.再往下遞迴達到指定深度後回傳回父節點
5. 最後回傳最佳解的移動方向

流程圖:



圖三、expectimax 演算法流程圖

樹搜尋圖示:



圖四、expectimax 演算法樹搜尋樹狀圖

實際盤面例子:

原始盤面:

```
+-------------------------+
| 512     8     0     0|
| 1024    32    0     0|
| 2048    8     2     0|
| 8192    4     2     2|
+-------------------------+
```

圖五、實際盤面舉例(原始盤面)

窮舉移動方向:

向上:

```
+-------------------------+
| 512     8     4     2|
| 1024    32    0     0|
| 2048    8     0     0|
| 8192    4     0     0|
+-------------------------+
```

盤面評估值:70141.7

向右:

```
+-------------------------+
| 0     0       512   8|
| 0     0       1024  32|
| 0     2048    8     2|
| 0     8192    4     4|
+-------------------------+
```

盤面評估值:21808.5

向下:

```
+-------------------------+
| 512     8     0     0|
| 1024    32    0     0|
| 2048    8     0     0|
| 8192    4     4     2|
+-------------------------+
```

盤面評估值:70042.8

向左:

```
+-------------------------+
| 512     8     0     0|
| 1024    32    0     0|
| 2048    8     2     0|
| 8192    4     4     0|
+-------------------------+
```

盤面評估值:70111.5

圖六、實際盤面舉例(四方向移動後盤面)

窮舉出現方塊(以向上後盤面為例):

右下角出現2:

```
+-------------------------+
| 512     8     4     2|
| 1024    32    0     0|
| 2048    8     0     0|
| 8192    4     0     2|
+-------------------------+
```

右下角出現4:

```
+-------------------------+
| 512     8     4     2|
| 1024    32    0     0|
| 2048    8     0     0|
| 8192    4     0     4|
+-------------------------+
```

...(尚有其他盤面需窮舉，此處只舉兩例)

圖七、實際盤面舉例(出現新方塊)

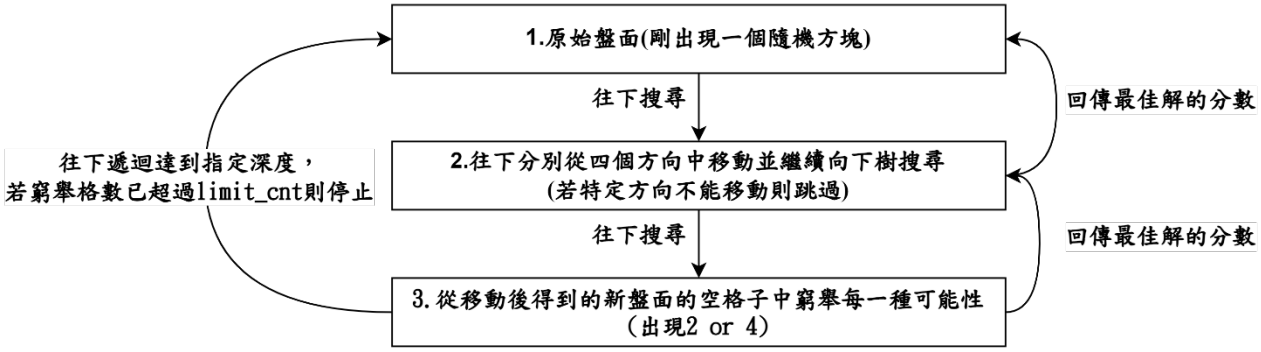
結果:
平均分數:170353
每局方塊有無出現分佈(一百場統計):

128(%)	100%
256(%)	100%
512(%)	100%
1024(%)	100%
2048(%)	100%
4096(%)	99%
8192(%)	89%
16384(%)	11%

表一、expectimax 演算法每局方塊有無出現分佈(一百場統計)
耗時:2143.7 秒
分析
平均分數已經到達約 170000，可見 n-tuple 結合 5-ply 的 AI 已經能玩出超越常人的成績。而其中耗時平均一場約 21 秒

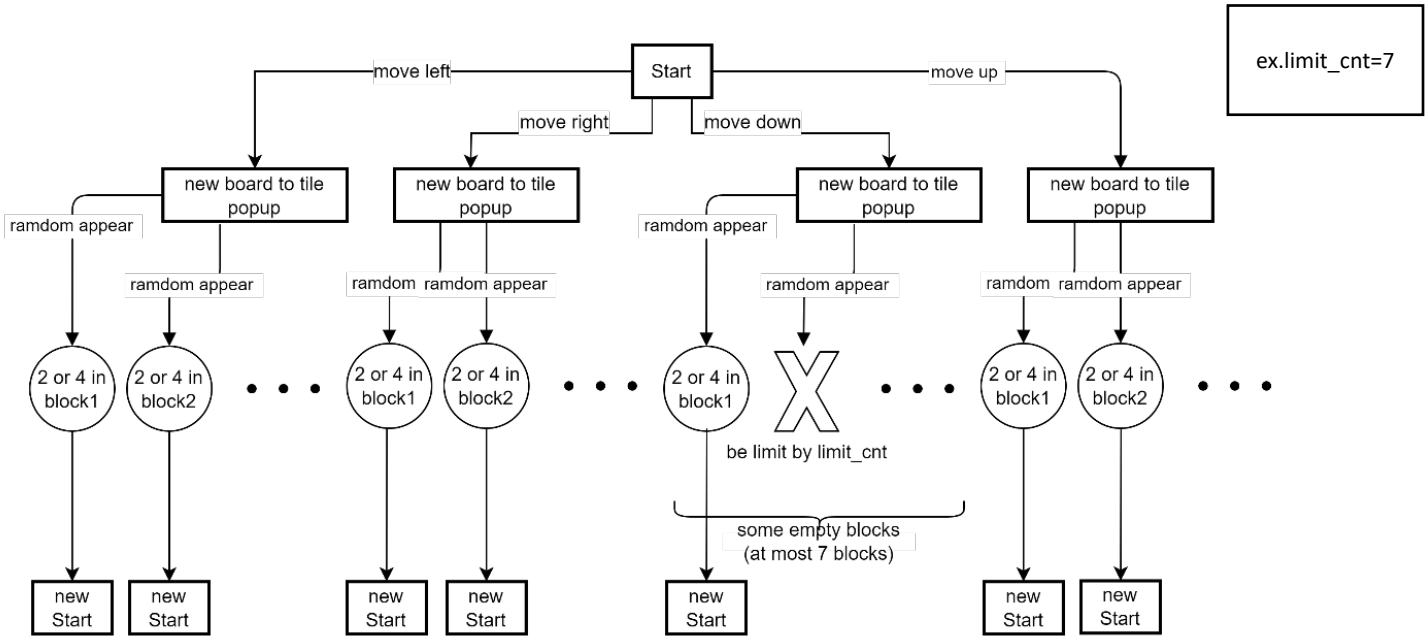
二、limit popup:
演算法介紹:
當現在樹搜尋處理的盤面有太多空格，導致要搜尋的空格太多，定義一格常數 limit cnt 為窮舉空格數上限，意即從現有空格中挑出不超過 limit cnt 個數做搜尋。
參數:
limit cnt:限制窮舉空格個數
搜尋流程:
1.原始盤面(剛出現一個隨機方塊)
2.往下分別從四個方向中移動並繼續向下樹搜尋(若其中一個方向不能移動則跳過)，並回傳最好的一步的分數
3.從移動後得到的新盤面的空格子中隨機窮舉最多 limit cnt 個空格子出現方塊的可能性(出現 2 or 4)，依盤面出現機率(出現機率方塊 2:方塊 4= 9:1，再依方塊出現在不同方格得到的分數取平均)加權回傳此盤面得分期望值
4.返回 1.再往下遞迴達到指定深度後回傳回父節點
5 最後回傳最佳解的移動方向

流程圖:



圖八、limit popup 演算法流程圖

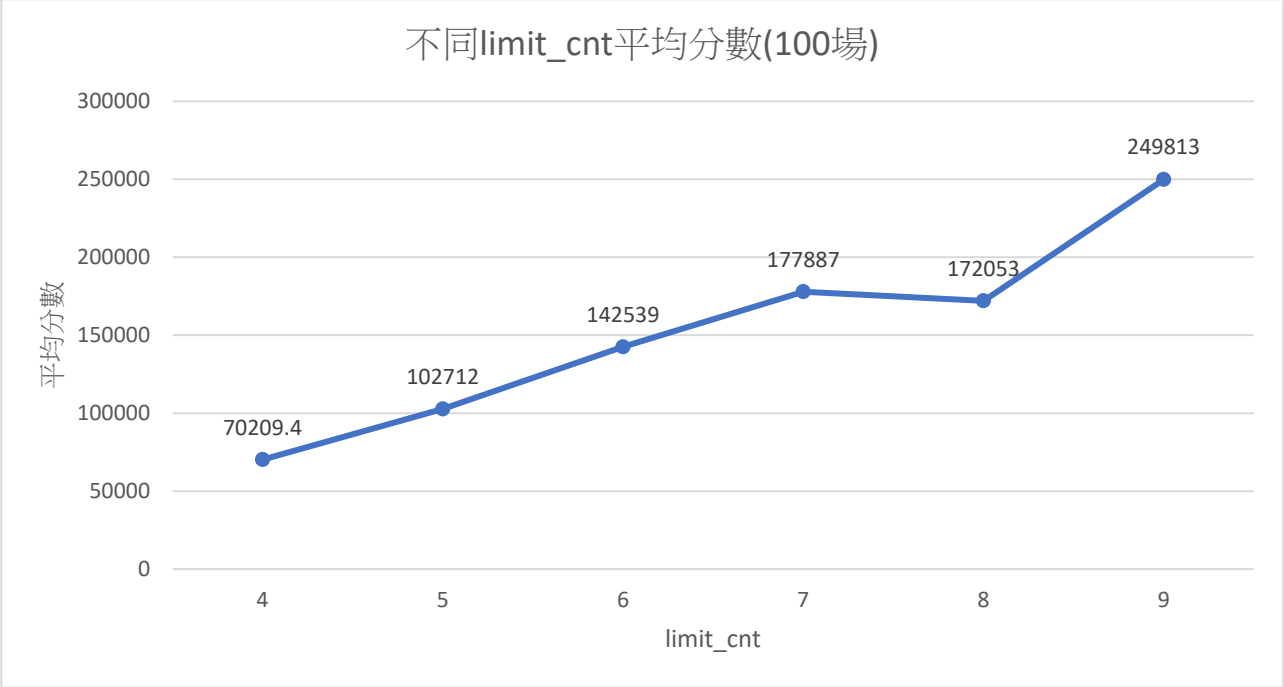
圖示:



圖九、limitpopup 演算法樹搜尋樹狀圖

結果:

平均分數:

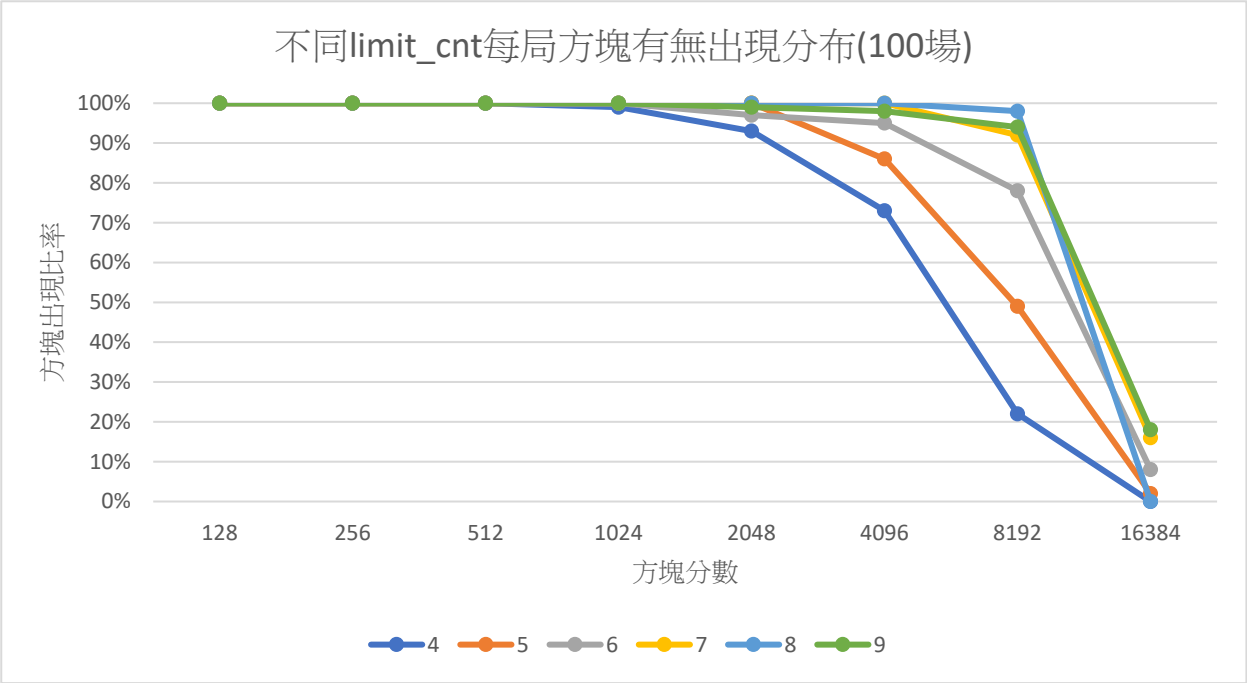


圖十、limit popup 演算法之不同 limit_cnt 平均分數(100 場)

不同 limit_cnt 每局方塊有無出現分佈(一百場統計):

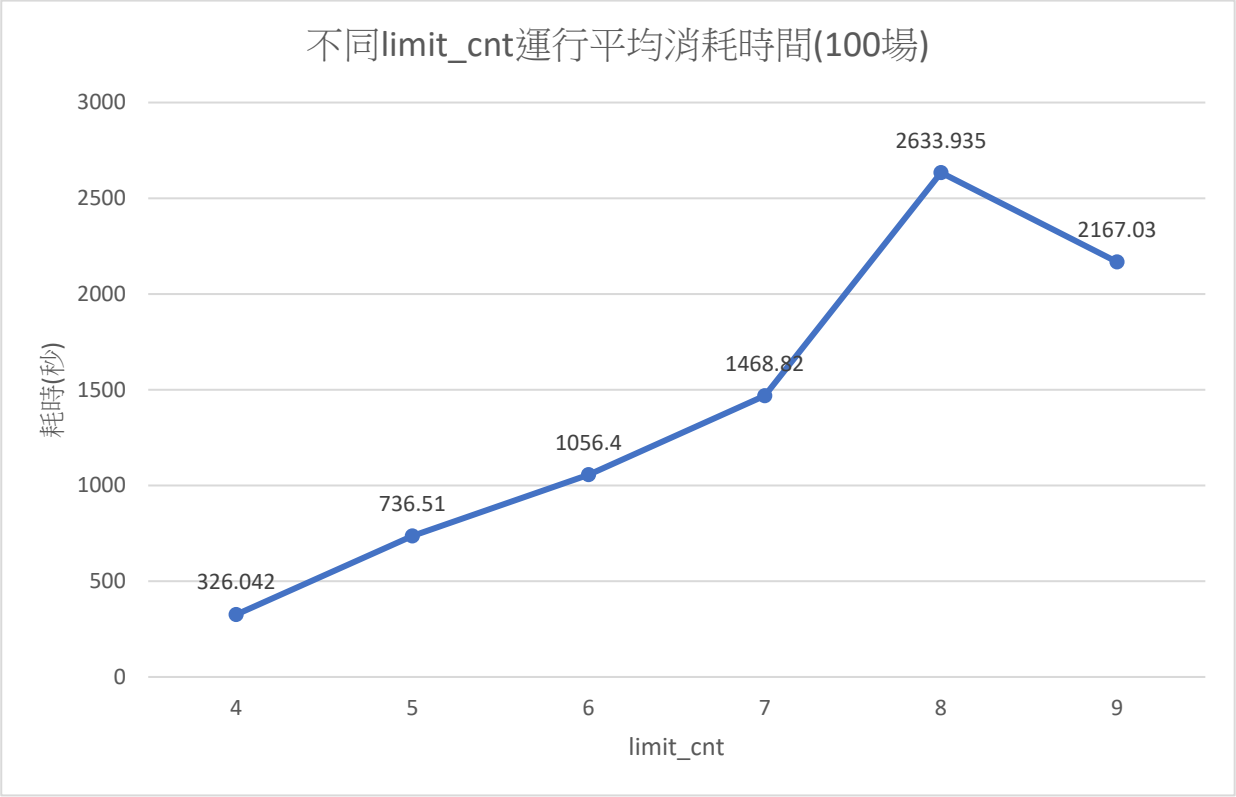
limit_cnt	4	5	6	7	8	9
128(%)	100%	100%	100%	100%	100%	100%
256(%)	100%	100%	100%	100%	100%	100%
512(%)	100%	100%	100%	100%	100%	100%
1024(%)	99%	100%	100%	100%	100%	100%
2048(%)	93%	100%	97%	100%	100%	99%
4096(%)	73%	86%	95%	100%	100%	98%
8192(%)	22%	49%	78%	92%	98%	94%
16384(%)	0%	2%	8%	16%	0%	18%

表二、limit popup 演算法不同 limit_cnt 每局方塊有無出現分佈(一百場統計)



圖十一、limit popup 演算法不同 limit_cnt 每局方塊有無出現分佈(一百場統計)

耗時:



圖十二、limit popup 演算法不同 limit_cnt 運行平均消耗時間(一百場統計)

分析:

在 limit_cnt = 7 時有最佳表現，可以達到平均分數仍維持在約 170000 的同時將時間縮減至原耗時的 68%左右，在 6 以下即有分數嚴重降低情形，8 以上則耗時與原 expectimax 相比無太大改變。推測是在大部分的盤面中出現的空格子不會超過 8 個，因 limit_cnt 若大於 8 則沒有剪枝作用，小於 6 則窮舉太少情況考慮不周，而精確度不足。

三、startimax:

演算法介紹:

當樹搜尋時，如果已經找到一個相當差的結果，即可停止搜尋這個方向，因為高機率這個移動不會是最優解，worst rate 即是在搜尋最優解時，如果目前移動方法之盤評估函數分數乘上 worst rate 小於目前遇到的最優解分數，則此移動方式大概率不會是最優解，因此不往下搜尋。

搜尋流程:

1.原始盤面(剛出現一個隨機方塊)

2.往下分別從四個方向中移動並繼續向下樹搜尋(若其中一個方向不能移動則跳過)，若此方向的盤面評估值小於目前搜索到的最大值*worst_rate 則跳過，最後回傳最好的一步的分數

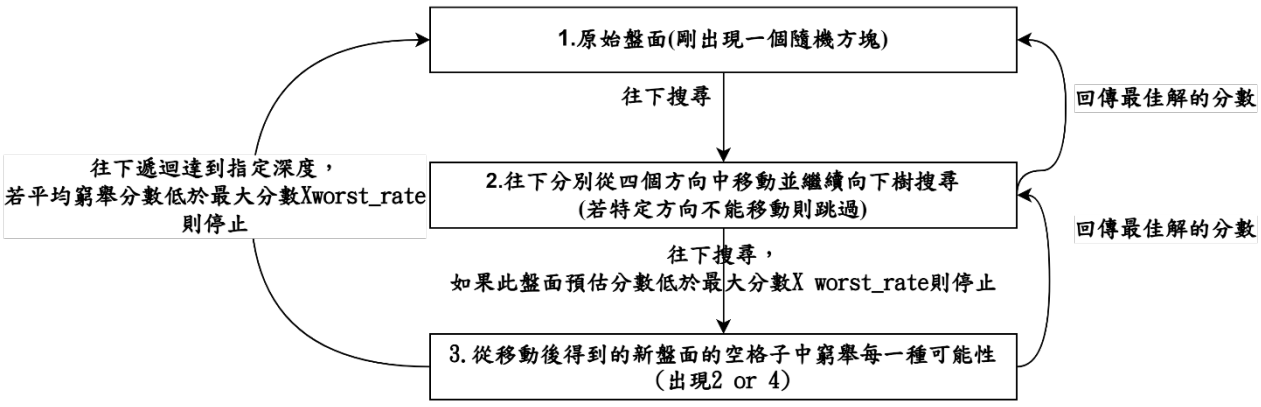
3.從移動後得到的新盤面的空格子中窮舉每一種可能性(出現 2 or 4)，

依盤面出現機率(出現機率方塊 2:方塊 4= 9:1，再依方塊出現在不同方格得到的分數取平均)，若目前搜尋窮舉過的節點分數加權平均小於最大分數*worst_rate 則結束搜尋並回傳此盤面得分期望值。

4.返回 1.的步驟再往下遞迴達到指定深度後回傳回父節點

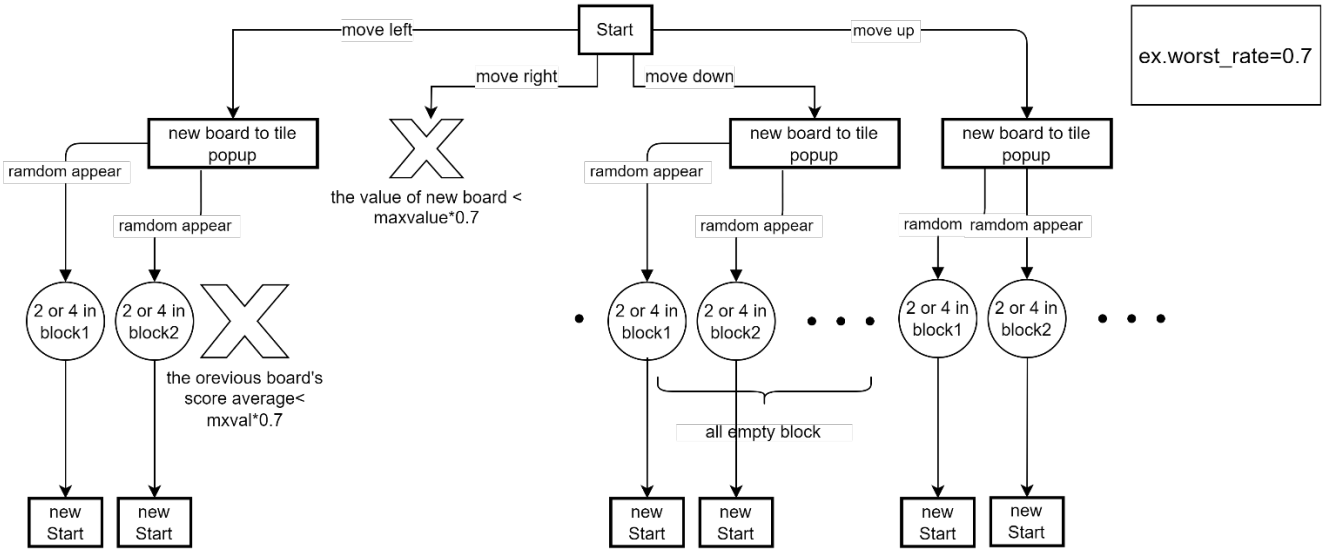
5.最後回傳最佳解的移動方向

流程圖:



圖十三、startimax 演算法流程圖

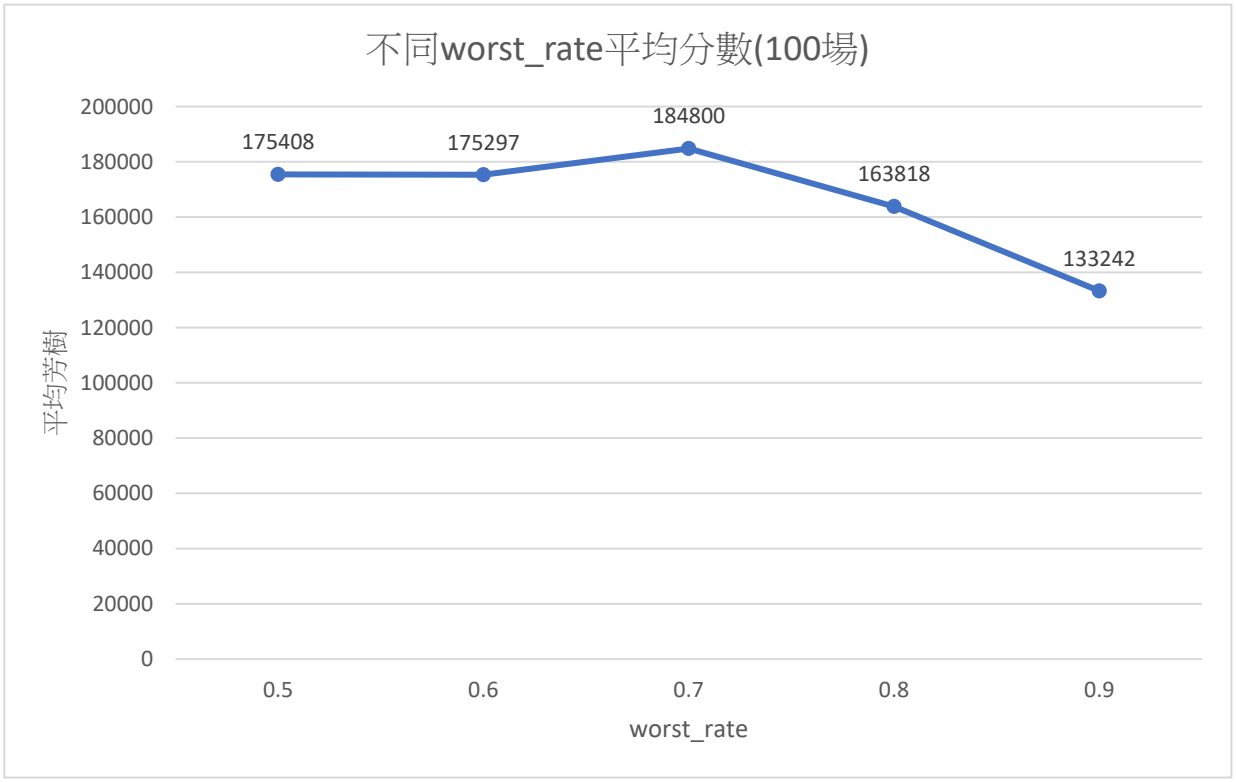
圖示:



圖十四、startimax 演算法樹搜尋樹狀圖

結果:

平均分數:

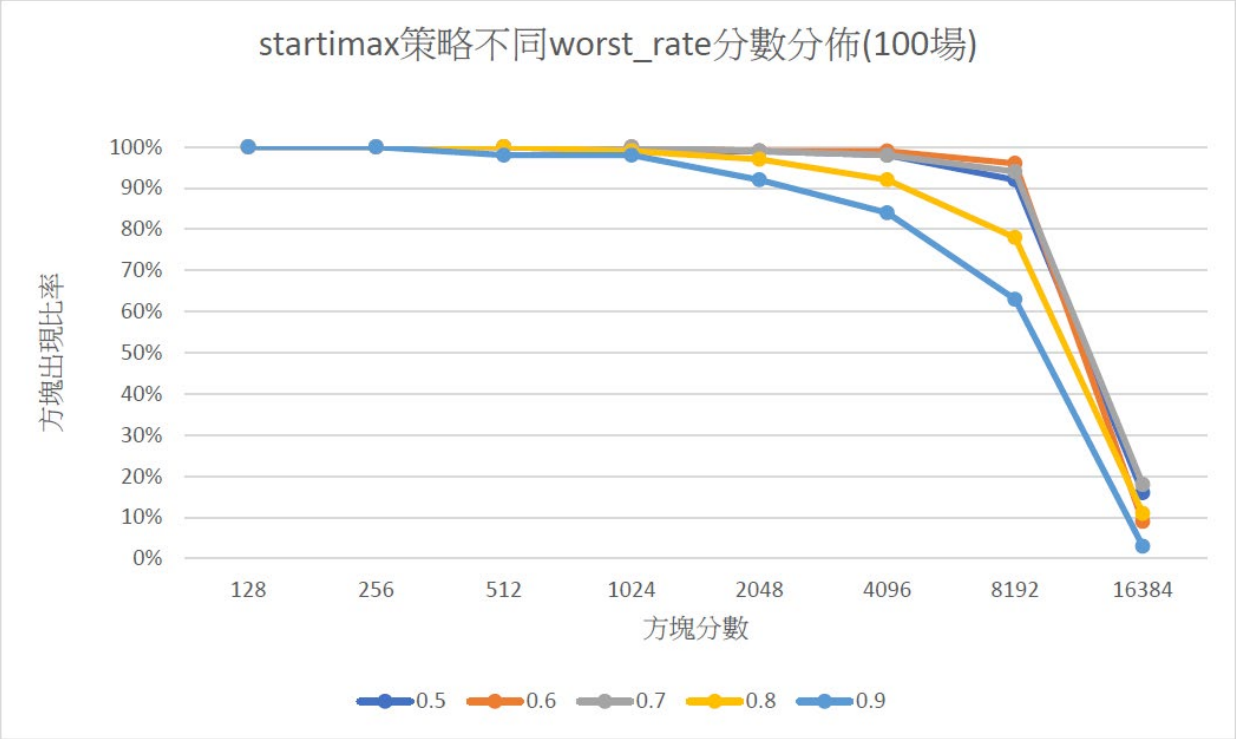


圖十五、startimax 演算法之不同 worst_rate 平均分數(100 場)

不同 worst_rate 每局方塊有無出現分佈(一百場統計):

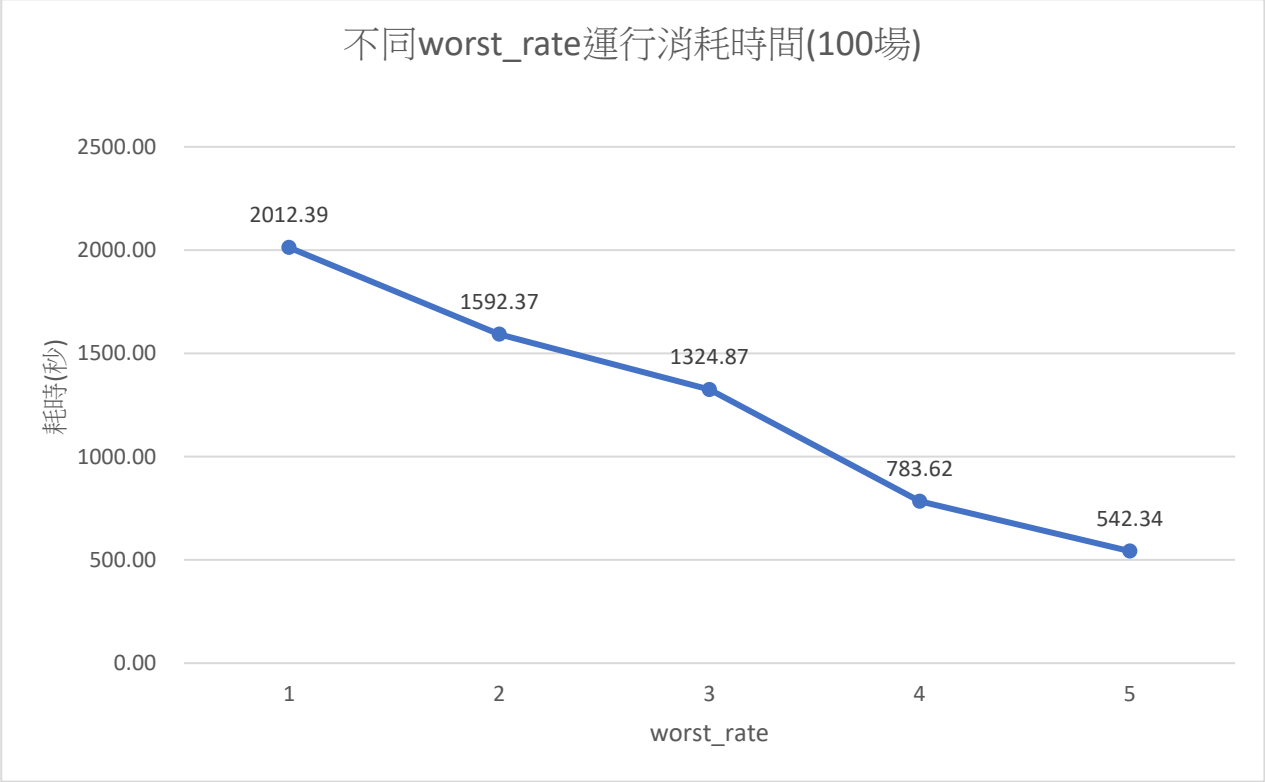
worst_rate	0.5	0.6	0.7	0.8	0.9
128(%)	100%	100%	100%	100%	100%
256(%)	100%	100%	100%	100%	100%
512(%)	100%	100%	100%	100%	98%
1024(%)	99%	100%	100%	99%	98%
2048(%)	99%	99%	99%	97%	92%
4096(%)	98%	99%	98%	92%	84%
8192(%)	92%	96%	94%	78%	63%
16384(%)	16%	9%	18%	11%	3%

表三、startimax 演算法不同 worst_rate 每局方塊有無出現分佈(一百場統計)



圖十六、startimax 演算法不同 worst_rate 每局方塊有無出現分佈(一百場統計)

耗時:



圖十七、startimax 演算法不同 worst_rate 運行平均消耗時間(一百場統計)

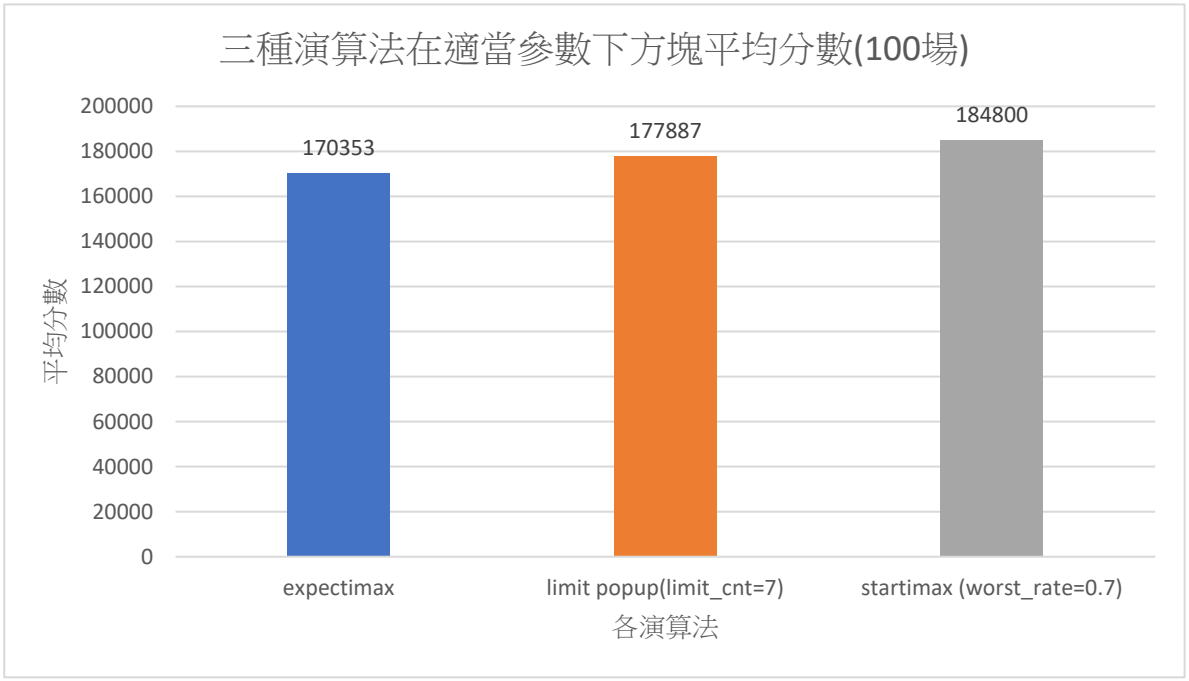
分析:

在 worst_rate 為 0.5、0.6、0.7 時皆無分數降低的情況，0.8 後分數開始嚴重下降，因此已能剪枝最多而不影響分數的 worst_rate=0.7 的效率最高，能將時間縮減至原耗時的 61%左右。原因與 limit popup 的結果分析類似，也是因為 0.8 以上考慮太少情況，而 0.6 以下則因考慮太多情況幾乎無剪枝效果。

結論:

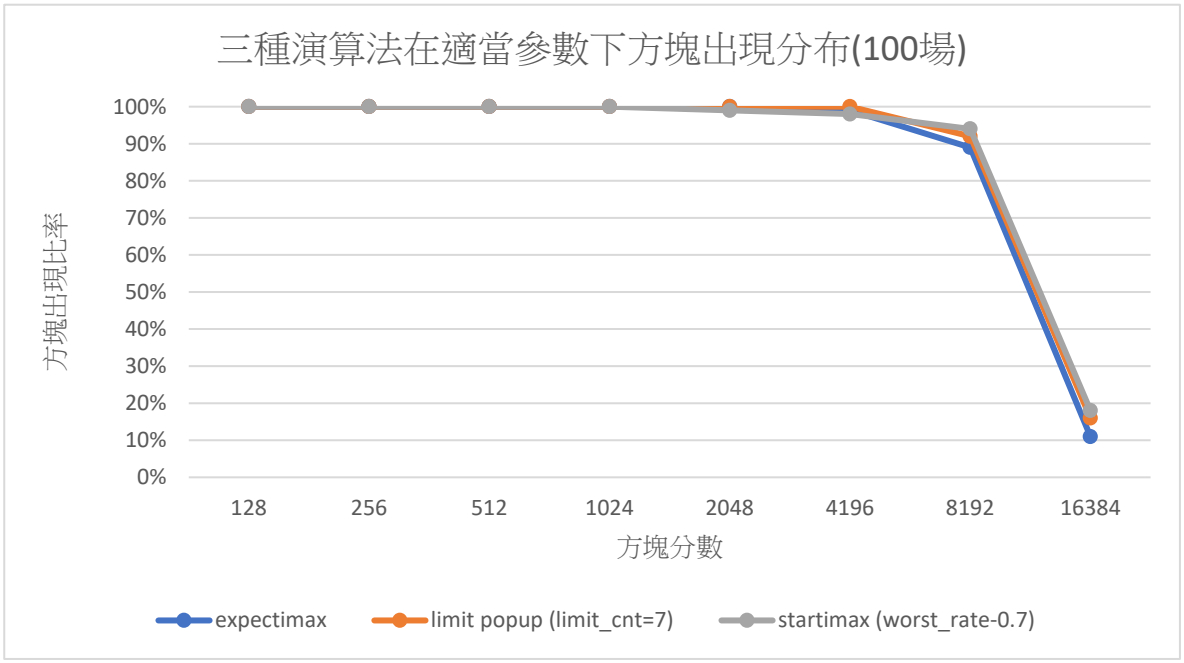
將各演算法表現最好的成果作比較:
(expectimax,limit popup (limit cnt = 7),startimax(worst_rate=0.7)

平均分數:



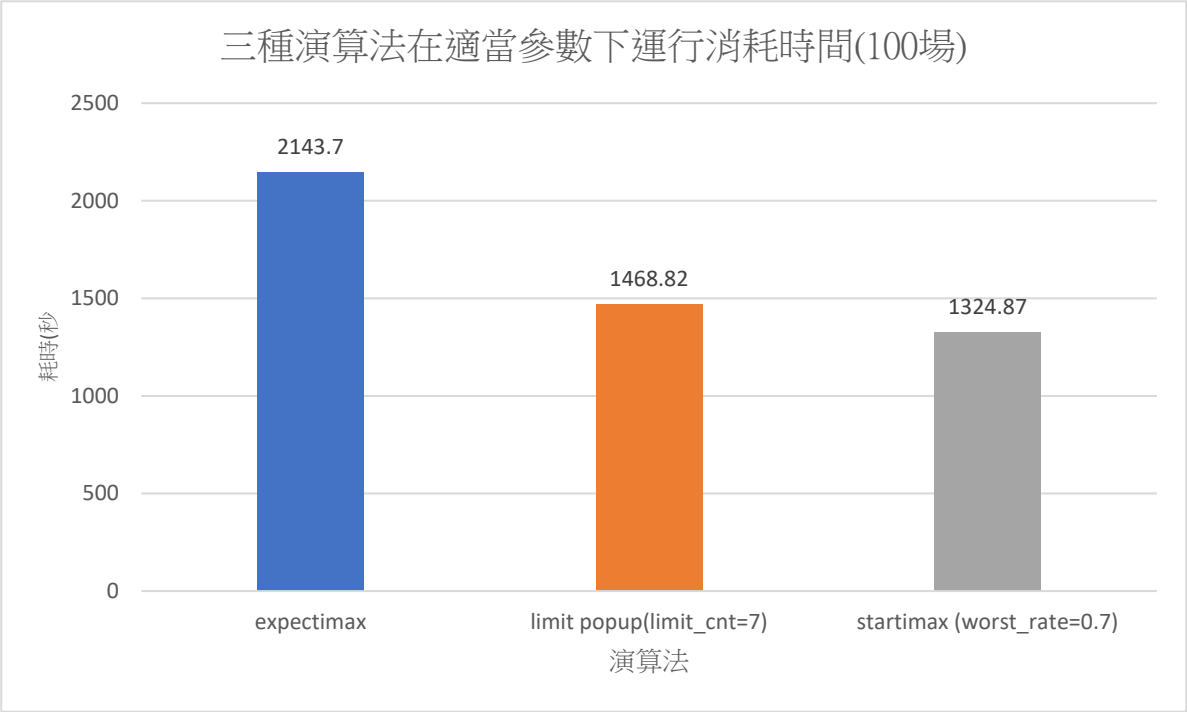
圖十八、三種演算法在適當參數下方塊平均分數(一百場統計)

單局最大方塊分數分佈:



圖十九、三種演算法在適當參數下方塊出現分布(一百場統計)

耗時:



圖二十、三種演算法在適當參數下運行消耗時間(一百場統計)

總結而言，各樹搜尋剪枝法在適當參數皆有不錯的表現，在平均分數下能和原 expectimax 演算法相差不多，方塊出現比率曲線皆能貼合原演算法，且皆能有效縮減運行時間至 60%到 70%，而以二分搜最佳參數數值也得到良好的成果，估計是因為無論是 worst_rate 或 limit_cnt，皆有設定偏離最佳值可能會考慮太少情況而造成分數低落或者搜尋太多盤面而剪枝不利造成耗時無降低。因此最後找到的參數應幾乎為演算法的最佳優化參數，也得很不錯的結果。

- 建議與展望:

1.盤面評估函數:

本次研究著重在於決策樹的開展與剪枝，如果進一步提高盤面函數的準確率(例如更換、增加 feature 或調整盤面評估函數)也可以增加仰賴盤面評估函數做剪枝的 startimax 演算法效率。

2.結合兩種演算法:

此次研究雖然探討了兩種剪枝方向，但兩種演算法是從不同面向作搜尋剪枝，若將兩種演算法結合，並調整不同的參數組合，相信能得到更高效率和得分能力的 AI。

3.多人賽局:

在本次研究中探討的還是著重於雙人賽局的情況，若要更進一步討論決策樹在現實世界中的應用(如:公衛、經濟等問題)，那即需要考慮較貼合現實的多人賽局情況，因此多人賽局轉化成決策樹問題也是值得研究的方向。

陸、參考文獻

Valero, C. Soto. "Predicting Win-Loss outcomes in MLB regular season games–A comparative study using data mining methods." International Journal of Computer Science in Sport 15.2 (2016): 91-112.

Yeh, Kun-Hao, et al. "Multistage temporal difference learning for 2048-like games." IEEE Transactions on Computational Intelligence and AI in Games 9.4 (2016): 369-380.

Matsuzaki, Kiminori. "Systematic selection of N-tuple networks with consideration of interinfluence for game 2048." 2016 Conference on Technologies and Applications of Artificial Intelligence (TAAI). IEEE, 2016.

Szubert, Marcin, and Wojciech Jaśkowski. "Temporal difference learning of n-tuple networks for the game 2048." 2014 IEEE Conference on Computational Intelligence and Games. IEEE, 2014.

柒、附錄

程式碼附錄(github 連結):https://github.com/tyrso/nycu-project