Final Project

R09922188 曾泓硯

January 5, 2023

1 How to compile your code into an agent

1. > $ make

The agent will be compiled into file "negascout".

2 What algorithms and heuristics you've implemented?

在這一次的專案中，我實作了 NegaScout, Star1 algorithm, transposition table, IDAS, time control module 以及 heuristic 的審局函數。

3 Experiment results and findings of your implementation

在實作的過程中，我發現自己的 IDAS 搜尋五層的結果，與直接搜尋五層的結果不同，在這樣的前提之下我比較了 IDAS 搜尋不同層數與直接搜五層的差別。

Agent	IDAS depth 6	IDAS depth 5	IDAS depth 4	direct depth 5
IDAS depth 6	X	10W10L	12W8L	8W12L
IDAS depth 5	10W10L	X	11W9L	7W13L
IDAS depth 4	8W12L	9W11L	X	6W14L
direct depth 5	12W8L	13W7L	14W6L	X

Table 1: Win rate under different agent

從表格中，我們可以發現直接搜五層的結果會比 IDAS 搜尋不同層數還要好，但是還是可以看出相差兩層時，搜尋越深的會比較有利。而 IDAS 的結果會出現這個的原因可能跟 transposition table 有關，因此我把骰子的資訊也一起加到 transposition table，來增加正確性。

4 Some detail about your implementation

4.1 NegaScout

這一次的作業中規定要實作 NegaScout，我在實作的過程中原本是使用 Nega 版本的，並且也成功的作出了 Star0 版本，但是到了 Star0.5 版本時發現剪枝後的結果與 Star0 不同，才發現說 chance node 也要寫成 Nega 版本的，也因為改不太出來，所以最後就寫成 Min-max 版本。另外，NegaScout 演算法中有一行當 $depth < 3$ 時就不用 research，這件事情因為在 chance node 計算時可能因為 pruning 導致結果不精準，所以不符合原本可以 pruning 的假設，因此我把它改成 $depth < 1$ 時才能夠剪枝。

4.2 Chance Node

這一次我實作了 Star1 的演算法，與 Star0 相比速度有感提升，原本 star0 搜尋五層可能需要花費 5 秒左右，但現在可以穩定限制在 2 秒內。基本上大致程式碼與講義上相同，但在實作的過程中，因為計算過程中會有誤差，會造成小數點後 7 位數字可能會不同，因此我將算出的數字先乘上 10^6 取 floor 以後再除回去，如此一來就可以得到完全相同的結果。

4.3 Transposition table

我實作的方法基本上與講義上相同，只是在 Min node 的時候 Upper bound 的 flag 要跟 Lower bound 交換，並且也利用 XOR 的特性在 O(1) 的時間內計算出下一步的 hash value。此外，參考講義上的公式，我把 table 的數量設定在 2^17，hash 的數值範圍則是 2^25，以此來降低 hash miss。另外，我在經歷第一輪比賽後，就把骰子的資訊加入到 hash 的計算中，也讓 transposition table 在整場比賽中共用，不會每一步就刷新一次。

4.4 Heuristic evaluation function

在經過測試以後，我發現用整個盤面的數據來進行統計其實是沒有太大的意義的，因此我利用雙方離勝利最近的棋子來進行統計，結果發現對各棋子而言，最近的距離以及它可以移動的機率會對結果影響最大。

4.5 Time control

由於目前的演算法在搜尋五層的時間花費不超過 1 秒鐘，因此我的目標是讓演算法搜尋得更深，或者退而求其次，在關鍵步的時候搜尋久一點。搜尋 6 層的時間花費大概介於 15 40 秒之間，因此，我利用 這一步的時間 × 這一步的時間/上一步的時間來估計下一步所需的時間，如果有機會做完的話，就會再搜尋一層。另外，我也設定了一個時間的硬上限 (剩餘時間的/3)，如果超出的話會直接回傳到根結點。

4.6 IDAS

基本上按照講義上去實作，threshold 的設定上就是憑感覺而已。

4.7 Verification

在驗證的時候，我先用 Star0 打了 20 場，並且把 Star0 每一步的選擇都記錄下來，在實作 Star1 的時候，去對照每一步的選擇是否都跟 Star0 相同，也因此抓到了，NegaScout depth < 3 以及小數點不精確等問題。程式的使用方式如下。

4.7.1 Verification 實作

a. 把所有對局紀錄匯集在一個檔案中, ex. verifiy/All_log_new_5.txt
b. 跑 generate.py 生成指定數量的走步 -> input.txt, gt_ans.txt
c. 用 make debug 編譯程式，並且執行，此時 agent 會吃 input.txt 的輸入，並且把答案記錄在 myans.txt
d. 用 diff 比較 myans.txt 及 gt_ans.txt

4.8 Analysis

從 Table. 1 中，我發現 IDAS 搜尋越深的結果不一定會比前一層還要好 ex. IDAS depth 5 vs. IDAS depth 6，因此我有讓直接搜五層跟搜六層比賽，雖然沒有比滿 20 場，但是大致上搜六層還是有比較好的表現，但是一定會超時。在這樣的前提下，我認為找到一個好的審局函數才是關鍵的所在。

我利用 Week1 的版本 (最近棋子還有幾步勝利、最近棋子能夠移動的機率) 進行了 2000 場左右的 self-play，試圖依靠 Machine Learning 的方法來找哪一些因素會影響到最終的預測。首先是特徵的選擇，可以分成全局的特徵以及局部的特徵。

4.8.1 Global Feature

所謂的全局特徵，指的是考慮所有的棋子。以下列出我搜尋過的特徵：第幾行/列有幾個棋子，剩下哪幾個棋子 (64 種排列組合)，平均距離，棋子的數量，是否有棋子在角落... 等。然而，從統計上的意義來看，這些特徵都沒有明顯的分佈，因此我也沒有使用這些特徵。

4.8.2 Local Feature

局部的特徵在這邊指的是距離終點最近棋子的各種特徵。包含，距離終點的距離，是否為先手，該棋子被骰到的機率，該棋子點數為何，有多少其他的棋子與該棋子同行/列，有多少友方的棋子保護，有多少敵方棋子有所威脅... 等。從統計上來看，最有感的就是距離終點的距離，以及該棋子被骰到的機率。

我提取這些特徵後匯入到一個 csv 檔案，並且用 Logistic Regression 來進行分類的任務，希望能夠從這些特徵獲得更高的分類準確率。另外，我提取特徵的盤面是從第五步以後開始，因為我的程式最少會搜五層。

4.8.3 ML 實作的程式

1. 利用 analysis/main.py 來提取特徵並且寫入 CSV 檔案
2. 用 Jupyter notebook 打開 Training.ipynb 來跑分類任務

4.8.4 結論

從分類的準確率上來看，提取終點距離、骰到的機率、是否為先手這三個特徵就可以達到快 68% 左右的準確率，比較讓我意外的是是否有敵人以及是否被保護竟然對分類任務沒有明顯的幫助。

然而，上述特徵所得到的結果是否能夠轉移到審局函數的提升也是需要實際去 self-play 才能有結論。與 Week1 對打的結果來看，『是否為先手』這個特徵會造成不穩定，導致在藍方時可以獲得 13 勝 7 敗，在紅方時卻為 8 勝 12 敗，因此可能不太適合當作競賽的版本。

另外，我也猜測審局函數的評估可能不能用分類任務的結果來斷言，因此有試著加入『有多少友方的棋子保護』,『有多少敵方棋子有所威脅』這兩個評估標準來進行 self-play，然而分別加入或是一起加入這兩個標準依然敵不過 Week1 版本。

從結果上來看，Machine Learning 帶給我的幫助比較像是能夠快速刪除掉不合理的選擇，以及提供超參數，我最終用的 1: 0.17 就是從 Logistic Classification 中得到的，只可惜沒有因此找到更多有用的審局函數。

5 Discuss benefits of various enhancements

Week1 - 在第一週的時候，我把 NegaScout, Star1, transposition table, heuristic evaluation function 都實作出來，發現從 Star0 - Star1 的過程中搜尋的時間會有 2 倍左右的進步，而 transposition table 也還會再有 1.5 倍左右的加速。至於 Evaluation function 我一開始是只使用距離當作參考，也讓我可以穩定贏過 random baseline。

Week2 - 經過第一週的比賽後，我將 transposition table 引入了骰子點數的資訊，並且每一步結束後不清空，讓下一步可以利用先前搜尋的結果，再來加上了時間控制的參數，讓 IDAS 可以在指定的時間內盡可能走搜尋越深越好。此外，我也補上當只有一種走步方式時，就直接選擇沒有必要再搜尋的方法。最終與第一週的版本比大概只有 60% 勝率，所以有點糾結要用哪一個版本。

Week2 最終戰績：我在進行完第一場後發現用 IDAS 增加收尋深度的比賽結果並沒有特別好，所以在第二場之前就把深度調回 5 層，雖然會造成搜尋深度不夠深，但是就結果來看效果其實還算不錯。

1. 3 勝 3 敗
2. 6 勝 0 敗
3. 5 勝 1 敗
4. 5 勝 1 敗
5. 4 勝 2 敗

Final Score: 23 勝 7 敗

6 Reference

- 課程講義