Report Group 1 SNA HW1

r03922096 洪立遠, b00902104 楊勛, b00902057 陳煥元

前言

我們作此作業，可以 10/21 為分界，分為前期，與後期。而進入的後期的關鍵，在於一個我們解決了 graph reset 的 bug。因為跑 greedy 需要作模擬的 propagate，在模擬的 propagate 之後又要 reset 回原本的 state。這裡有一些複雜性，所以造成 greedy 的錯誤。

解決 bug 進入後期後，配合正確的 greedy 已可打贏 MaxWeight Player 1，很多的努力放在了 greedy 的加速，與找出時限內的最佳策略配對。

另外，我們基於 DiffusionModel 的 code，自己加了很多的功能如 export 整張圖、simulate propagate 等，變成 MyDiffusionModel，這樣我們才可以做到跟上整個遊戲的進程與跑自己的演算法。

前期

以下是我們所實作的 strategy，我們作為 Player 2 對決助教 Player 1 MaxWeight Strategy。統計數據是在 egofb.txt 這張圖上跑出來的結果。

1. DegreediscountIC:

參考 KDD 2009 的論文所得到的方法單純用他跑的話對上 S1，會輸 974 點。

2. MaxWeight:

利用助教的 code 直接互相對打輸 900 多點。

3. MaxWeight version 2:

考慮到要阻擋對手的 Activation，我們會先把敵人在沒有我們干擾的情況下，一回合能夠 activate 的所有的點，變成一個 set，讓 Maxweight Strategy 每次都只在那個 set 中選擇點。輸 578 點。

4. DegreediscountIC Version 2:

利用同樣的阻擋對手 Activation 的概念，讓 DegreeDiscountIC 在一回合能夠 activate 的所有的點，去做篩選，輸了 600 多點。

5. Mix_Heuristic:

一半的點用 MaxWeight 去選 一半的點用 DegreeDiscount 去選，這樣的方法會輸 800 多點。

小結論

在我們一開始輸的情況下，限制選點在一個比較小的，對方會 activate 的點的集合，效果會比較好。

後期

Greedy:

在 largest connected component 的點集合中，一次選擇一個點，在每一次選擇點的考慮都是，選擇那個點後，選擇的點(包含已確定選擇的點)的集合，能造成最多的 activate 的點。

使用了正確的 greedy 之後，只要第一回合有用 greedy，之後配合其他的 heuristic，就會獲勝。只是 greedy 太過耗時，所以接下來花了時間在 greedy 加速上面。

另外，我們捨棄前期「利用阻擋對手 Activation 的概念」，因為 Greedy 在對方會 activate 的點的集合跑起來，發現效果較差。

Greedy 加速

1. 刪掉圖中的 selected 與 activated node

   因為每一回合，會有點變成 selected 或是 activated，而這些點對整個遊戲已經沒有影響了，繼續在 graph 中記錄他們及其 edge，會造成 overhead。所以我們會刪掉這些點及其 edge，然後才 export 成檔案，下次 restore 建 graph 時，就會比較快。

2. Giant component

   因為我們有刪掉 nodes，所以會造成 graph 變成許多 weakly connected components，而我們認為主戰場就是 giant connected component，不需要在那些其他的小小的 weakly connected component 上選擇，所以我們會限定 greedy 選擇的點的 set 在 giant connected component 上。

3. peek Window

greedy 選擇完第一個點後，會得到一個 activate node 數由大排到小 node list，這時選定這個 list 的前 n 個，也就是在 peeking window 內的這 n 個，做為之後選擇 2~10 點的 set。

也就是說，之後的 2~9 點，只會從這個 peeking window 中的點中產生，而依舊是以 greedy 的方式來選擇，這樣的好處就是可以控制 n 為一定數，目前設為 600，跑得 node 數量少，速度就快了。

這個效果甚至有時候會比原本的全部點的 Greedy 還好。

4. data structure 優化

   把許多 list, dict, reset 時重複建立的 data structure 都去除掉。

最終的 Strategy

第一回合 10 個點使用加速的 Greedy 去跑，後面 2-10 回合，會將一半的點給 Greedy，一半的點給 Max Weight 去跑。效果沒有全 Greedy 好，例如：在 hepth.txt 中，每一回都用 Greedy 會贏 2200 多個點，但是使用一半一半會在 30 秒內選擇完畢並贏 1500 多個點。

這樣的考量還是因為時間限制，怕全部都用 greedy 超時。

總結

Greedy 真的很強，能夠讓後手得到許多優勢，滿意外的而且也不知道為什麼，像是如果兩個 Player 都全部使用 greedy，也是後手會贏。在 Future work 上面，我們希望再把 greedy 加速(雖然感覺已經加速到不能再加速了，而且又不能用 multi thread)，然後再去實作更多 paper 中的演算法。