**Game Theory Homework 1**

**311551094 廖昱瑋 資科工所碩一**

1. **Simulation Environment (WS model)**

首先，先介紹如何建置WS model，這個模擬環境會用在接下來requirement 1-1、1-2、2的部分。以python中networkx package提供的 “watts\_strogatz\_graph” function建立WS model，其中參數n代表node數目，k代表初始時每個node的neighbors數目，p代表rewiring的機率。

接下來的題目皆設置：、、。並且每種rewiring機率要各跑100次，以呈現之後實作部分的performance。

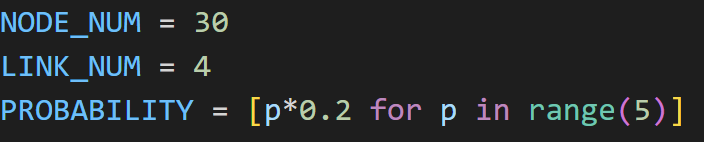


圖1. WS model的參數

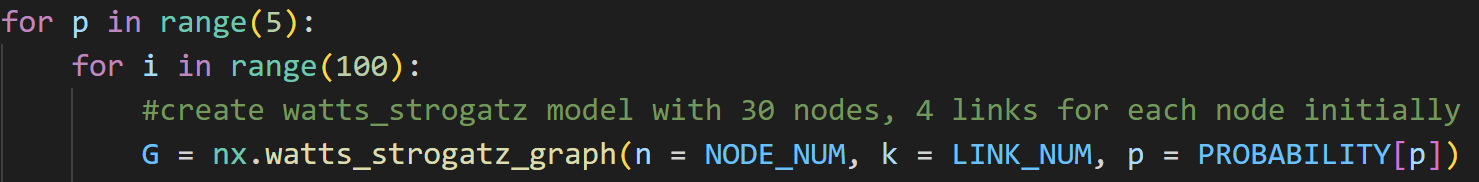


圖2. 以networkx建置WS model

1. **Requirement 1-1 (Weighted MIS Game)**
   1. **Utility Function and Best Response**

Weighted MIS Game的每個node都有一個整數weight，介於[0, 29]。我定義其priority function：

其中，pi為node i、Ni為pi的open neighbor、為node i的weight。

Utility function：

其中，、為大於1的常數、Li為priority大於等於pi priority的neighbors。

由以上utility function，我們可以推論出best response：

* 1. **Code**

在 “add\_node\_weight” function randomly assign[0, 29]範圍的weight給每個node，然後藉由 “calculate\_node\_priority” function以上述公式計算各node的priority。並以 “initialize\_strategy\_profile” function randomly給予每個node strategy{0, 1}。

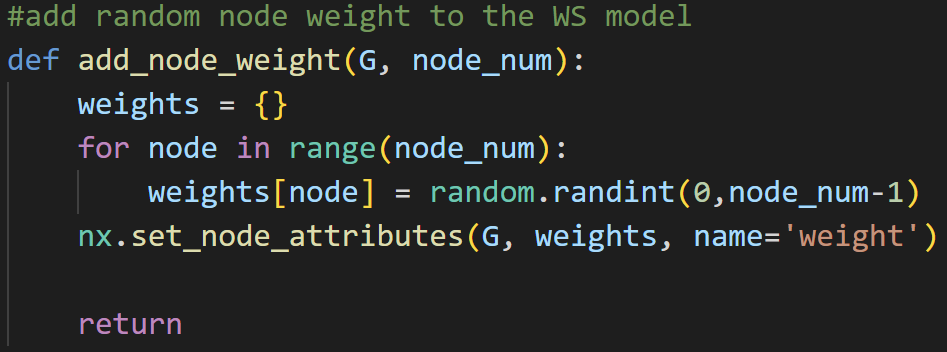
****

圖3. “add\_node\_weight” function

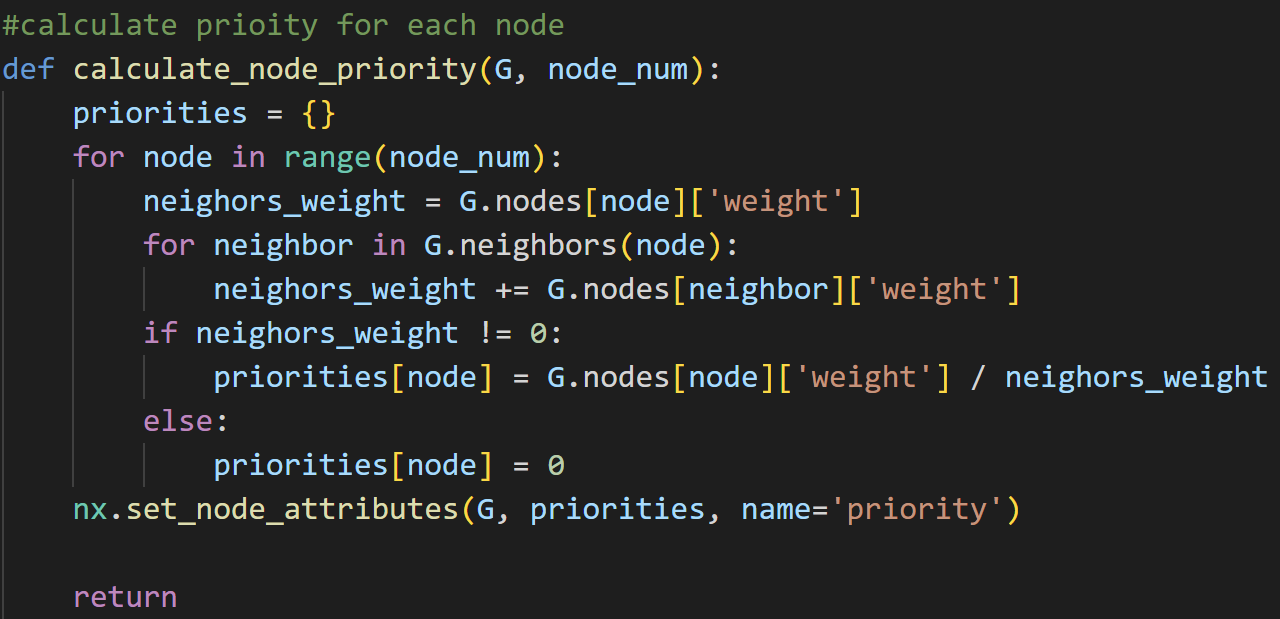
****

圖4. “calculate\_node\_priority” function

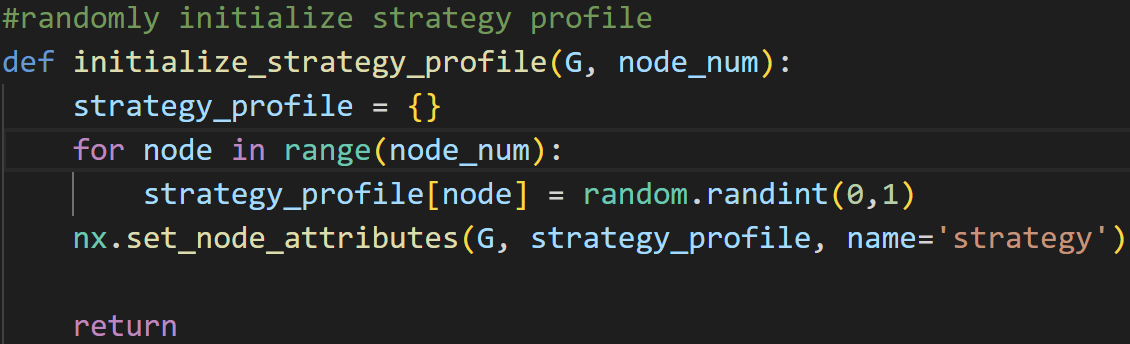
****

圖5. “initialize\_strategy\_profile” function

接著，建立一個迴圈，每次迴圈隨機挑選一個可以增加自身utility 的node，並把它的strategy改為其best response，一直重複執行，直至沒有任何node可以透過更改strategy增加其utility。

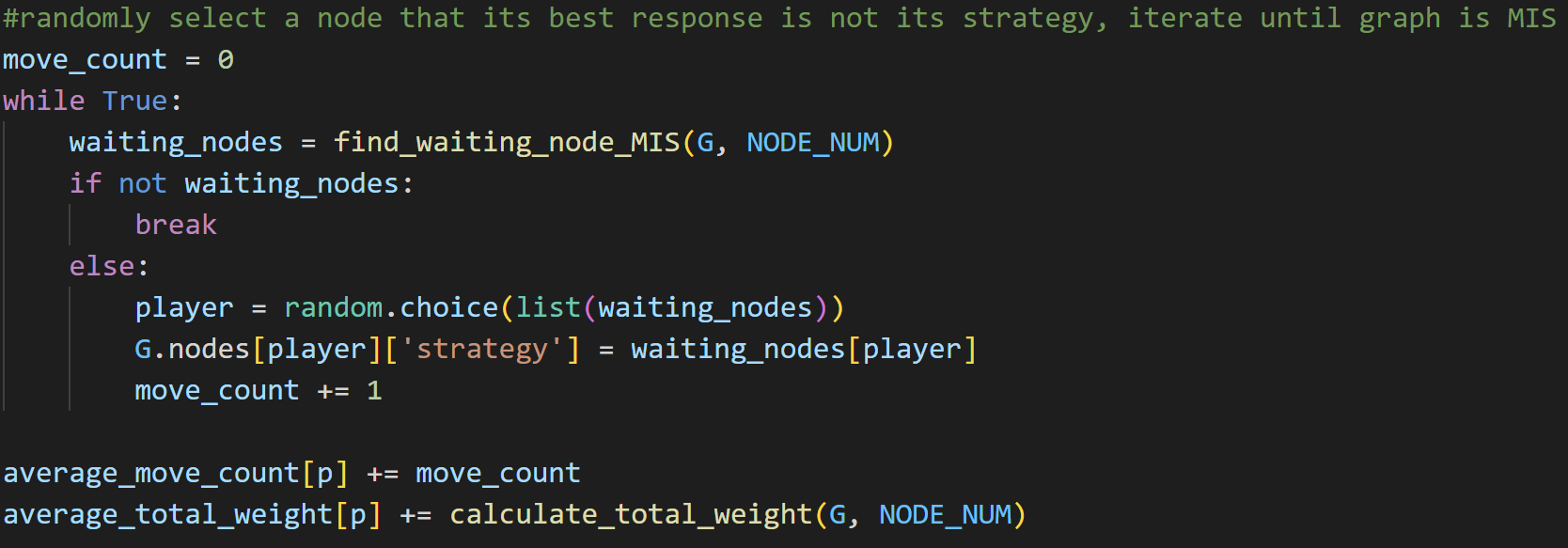


圖6. Weighted MIS Game主迴圈架構

透過 “find\_waiting\_node\_MIS” function尋找可以更改strategy增加其utility的node set。“find\_waiting\_node\_MIS” function中運用上述best response公式，先得到每個node的best response，並與它當前strategy做比較，若不相同，就把node number當成key；best response當成value存進 “waiting\_nodes” dictionary中。

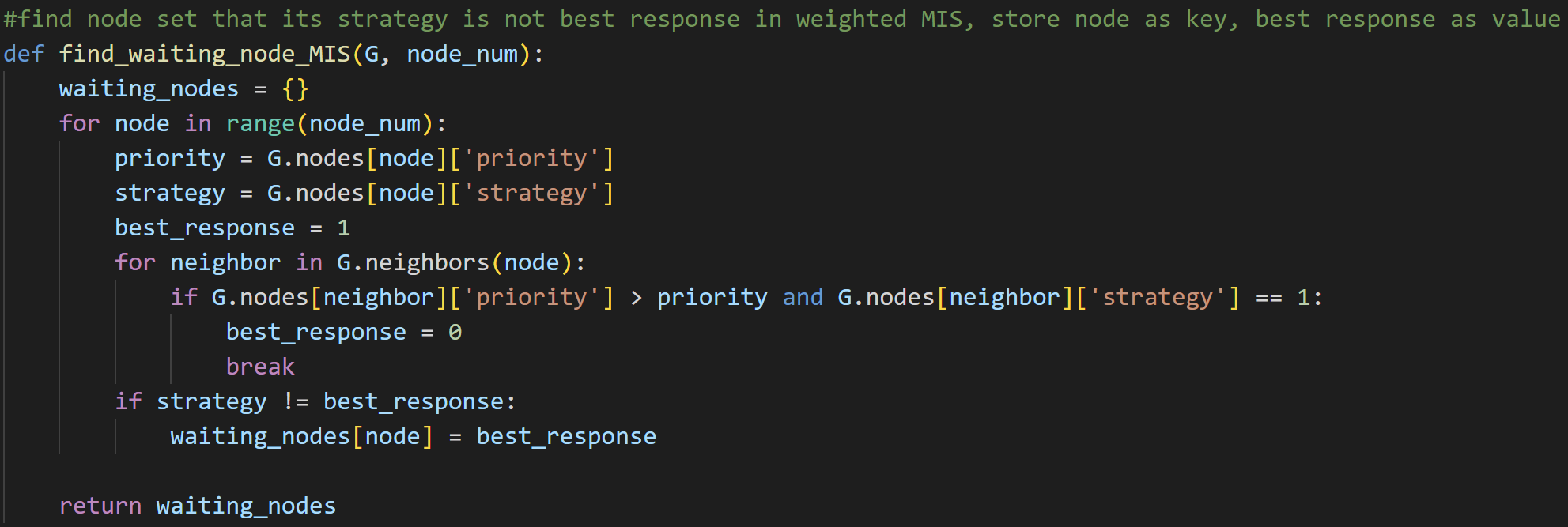


圖7. “find\_waiting\_node\_MIS” function

在不同rewiring probability情形下，以所有picked node的總weight及平均每個node進行strategy改變的次數分析performance。每個rewiring probability做100次再取平均。當中，使用 “calculate\_total\_weight” function計算圖中picked node的總weight。

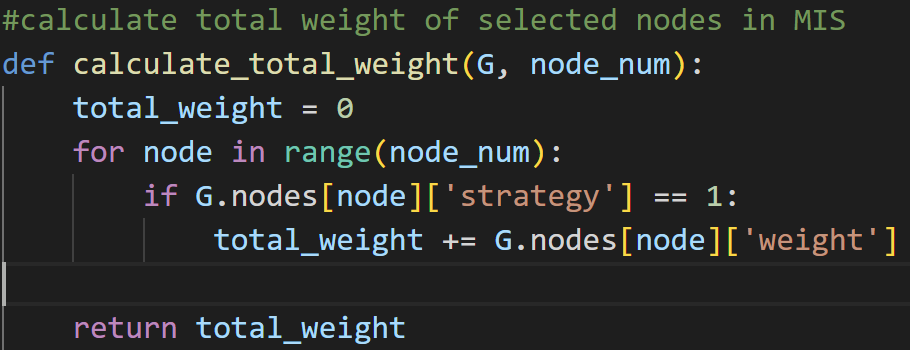


圖8. “calculate\_total\_weight” function

最後，先將node移動次數及總weight取平均，並把performance以折線圖呈現，並用 “print\_graph\_MIS” function畫一張Weighted MIS graph觀看結果的範例。

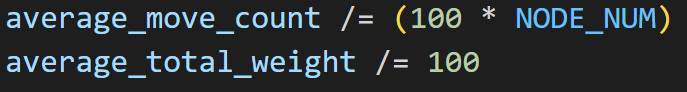


圖9. 將node移動次數及總weight取平均

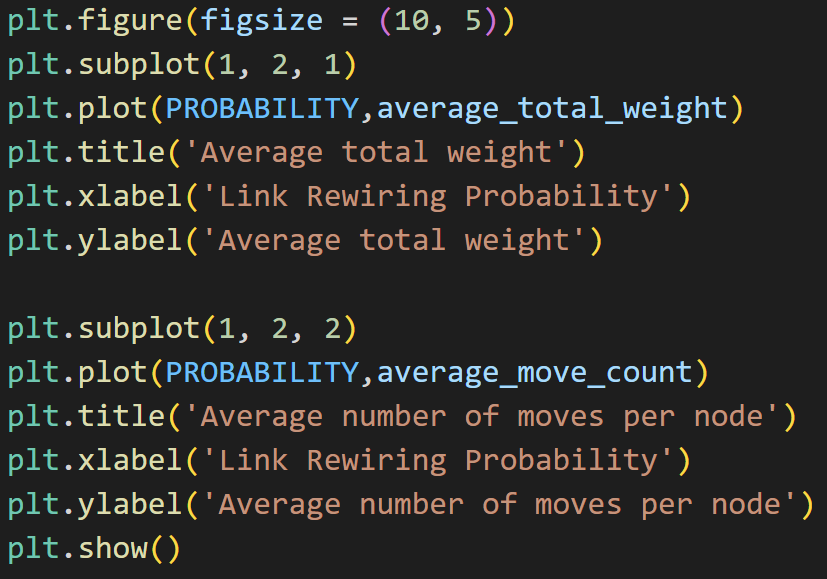


圖10. 畫Weighted MIS performance折線圖

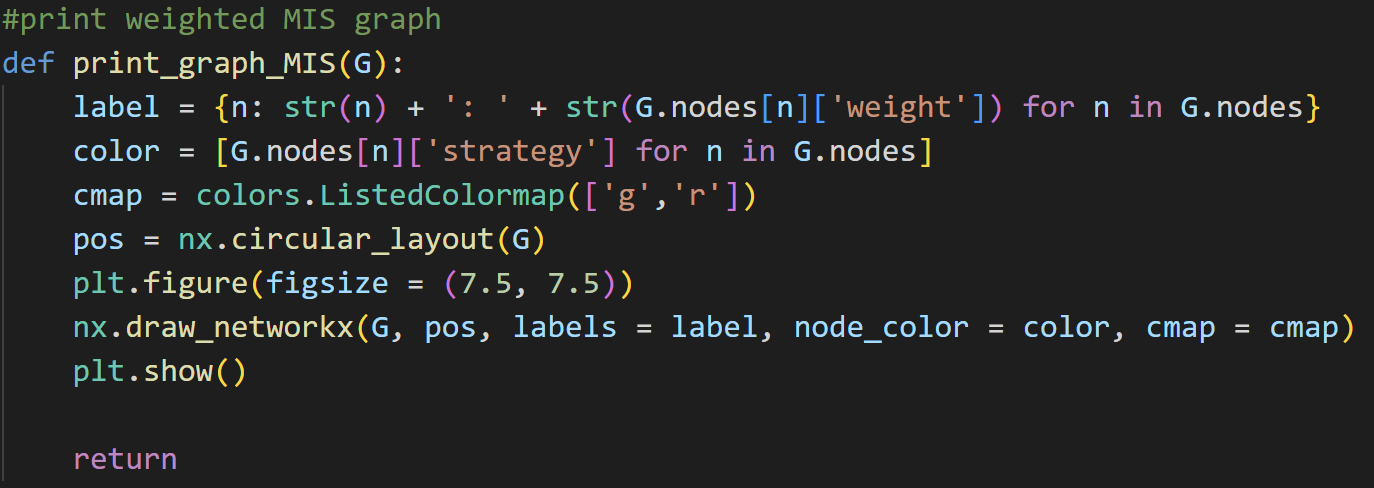


圖11. “print\_graph\_MIS” function

* 1. **Result**

由圖12.可以得知當rewiring probability越大時，總weight會越大，因為graph被mixed得更好，有助於找到更多node符合independence性質。當rewiring probability越大時，平均每個node進行strategy改變的次數大致有變多趨勢，但其實y軸數值差異太小，所以趨勢方向不是很明確。

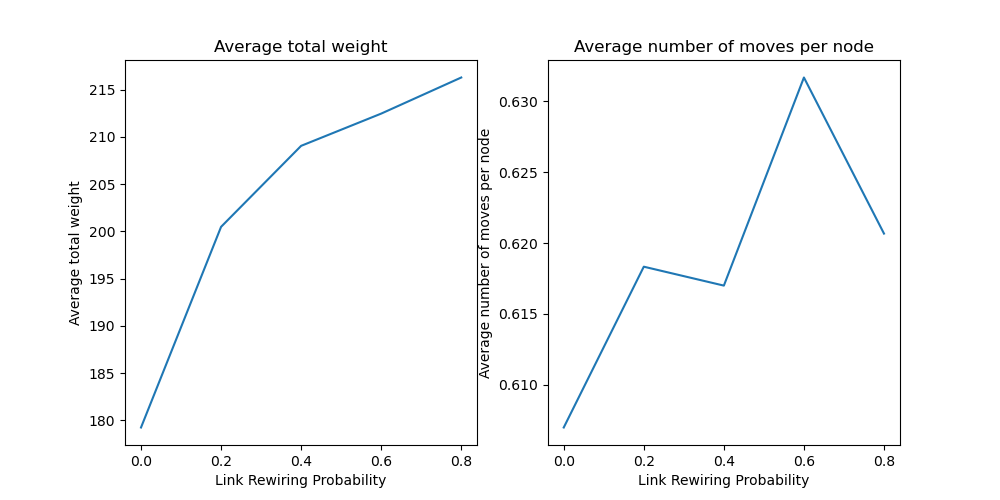


圖12. Weighted MIS performance

圖13.中每個node上的label x:y，x代表node編號；y代表node weight。紅點為strategy = 1的點；綠點為strategy = 0的點。

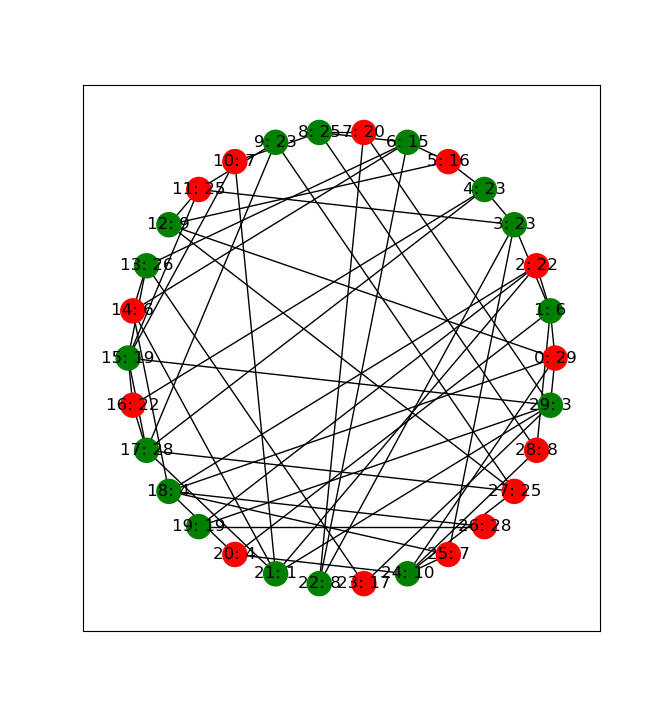
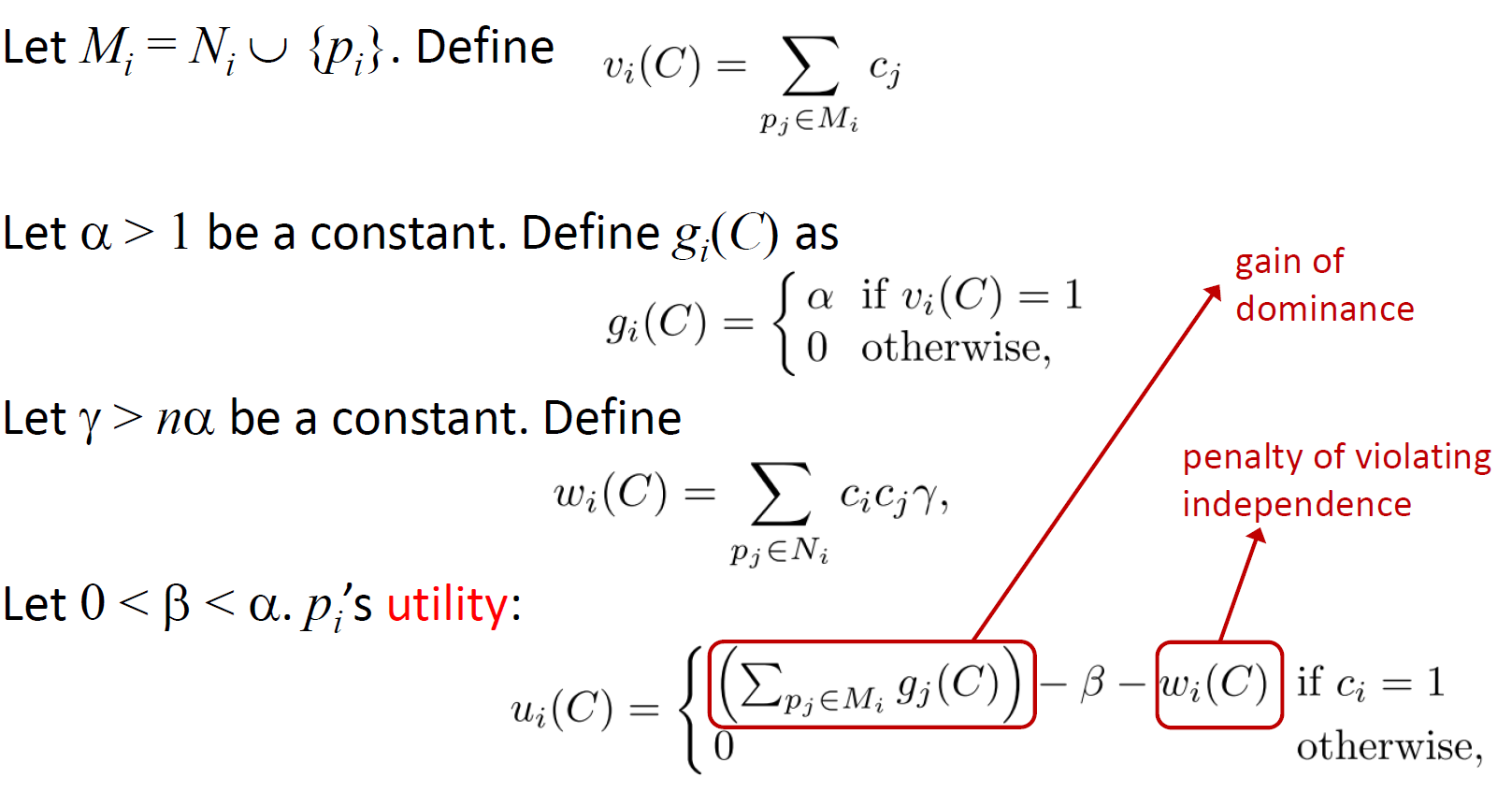


圖13. Weighted MIS graph example

1. **Requirement 1-2 (****Symmetric MDS-based IDS Game)**
   1. **Utility Function and Best Response**

Utility function：



由以上utility function，我們可以推論出best response：

* 1. **Code**

先以requirement1-1用過的“initialize\_strategy\_profile” function randomly給予每個node strategy{0, 1}。

主迴圈架構與requirement1-1相同，只差在尋找可以更改strategy增加其utility的node set是用Symmetric MDS-based IDS Game的best response公式，定義於 “find\_waiting\_node\_MDS\_based\_IDS” function。

並且因這裡的node沒有weight，performance改以cardinality測量，計算方式寫在 “calculate\_cardinality” function。

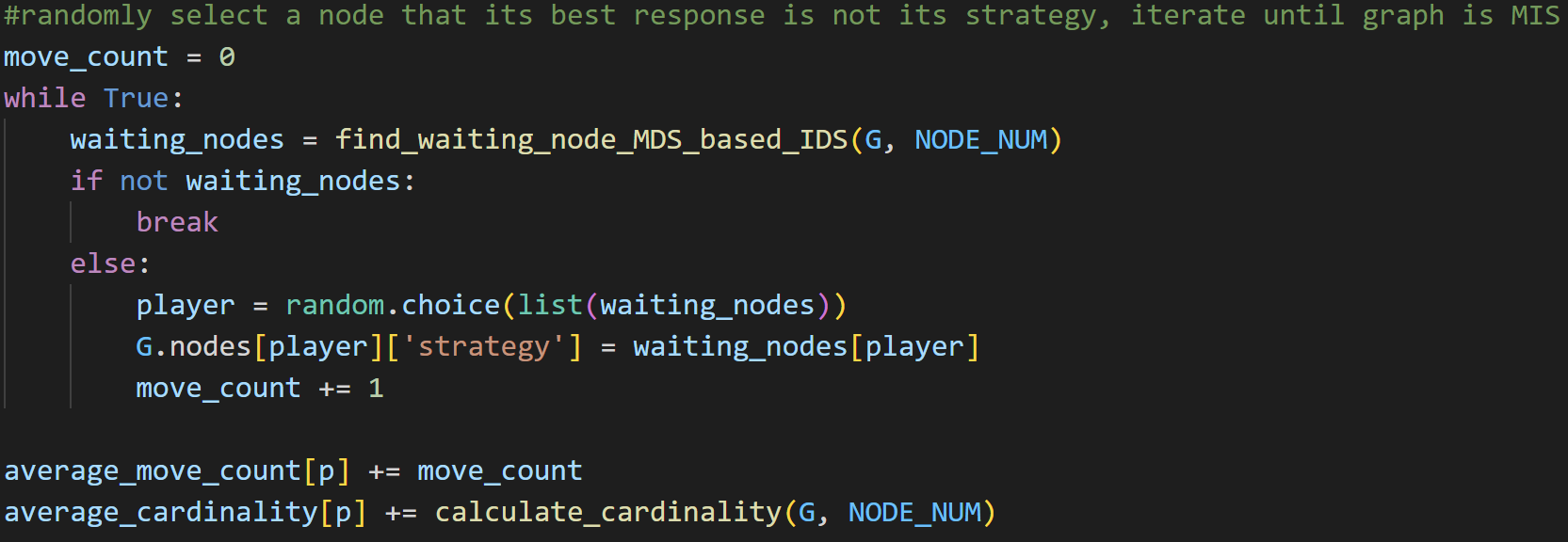


圖14. Symmetric MDS-based IDS Game主迴圈架構

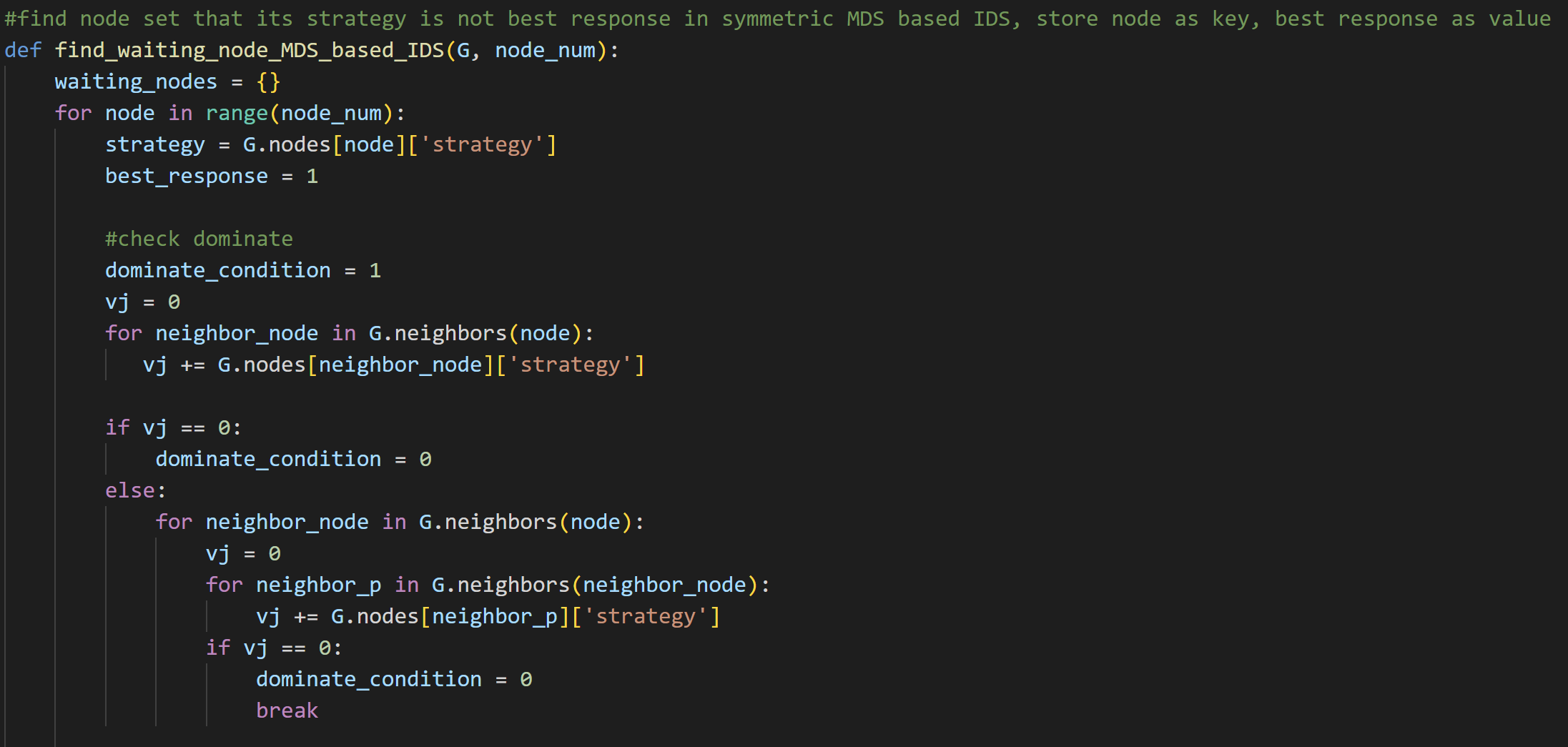


圖15. “find\_waiting\_node\_MDS\_based\_IDS” function – (1)

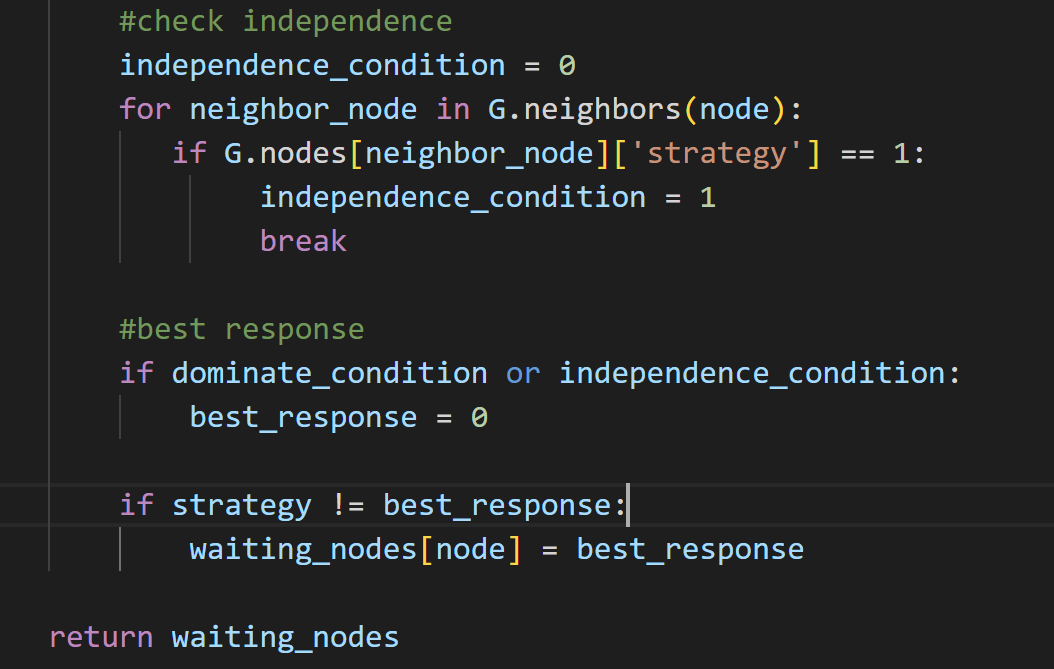
****

圖16. “find\_waiting\_node\_MDS\_based\_IDS” function – (2)

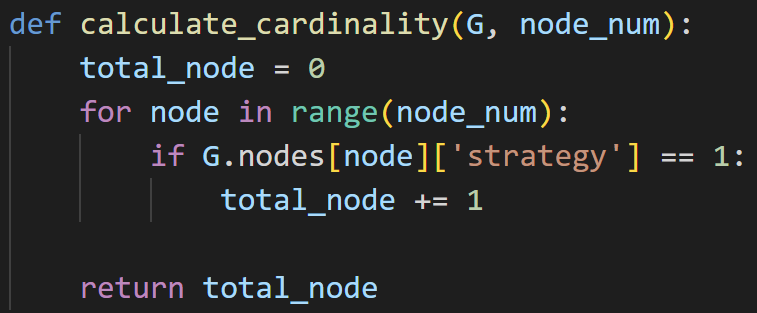


圖17. “calculate\_cardinality” function

最後，先將node移動次數及cardinality取平均，並把performance以折線圖呈現，並用 “print\_graph\_MDS\_based\_IDS” function畫一張Symmetric MDS-based IDS Game graph觀看結果的範例。

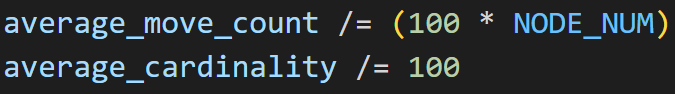


圖18. 將node移動次數及cardinality取平均

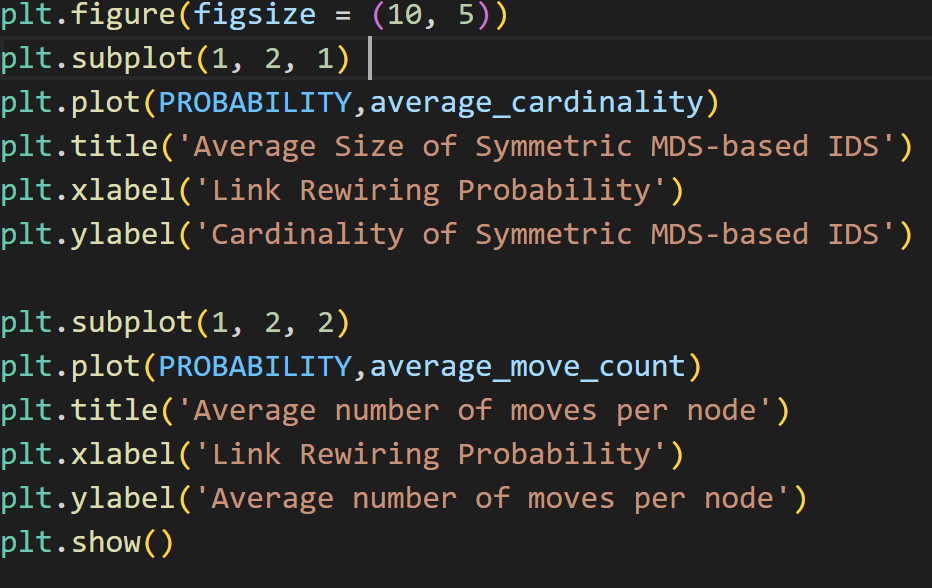


圖19. 畫Symmetric MDS-based IDS Game performance折線圖

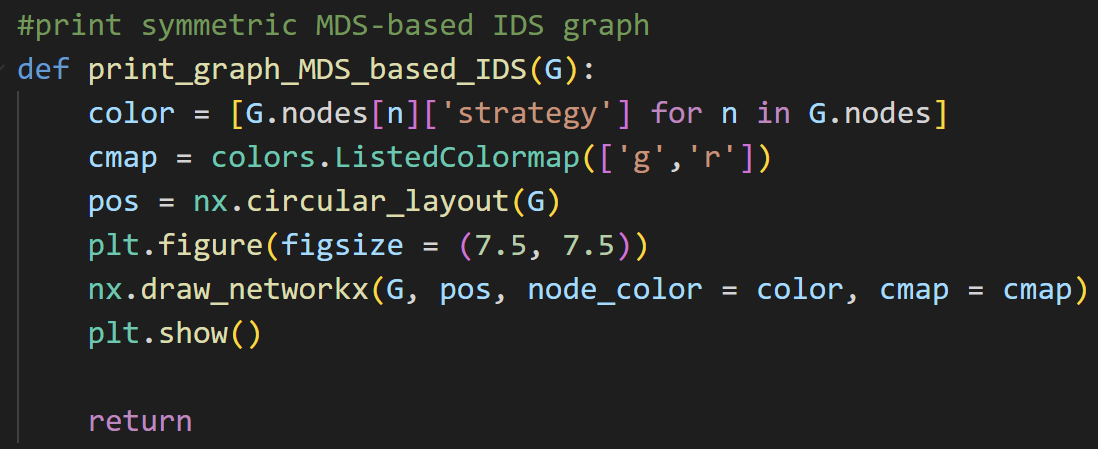


圖20. “print\_graph\_MDS\_based\_IDS” function

* 1. **Result**

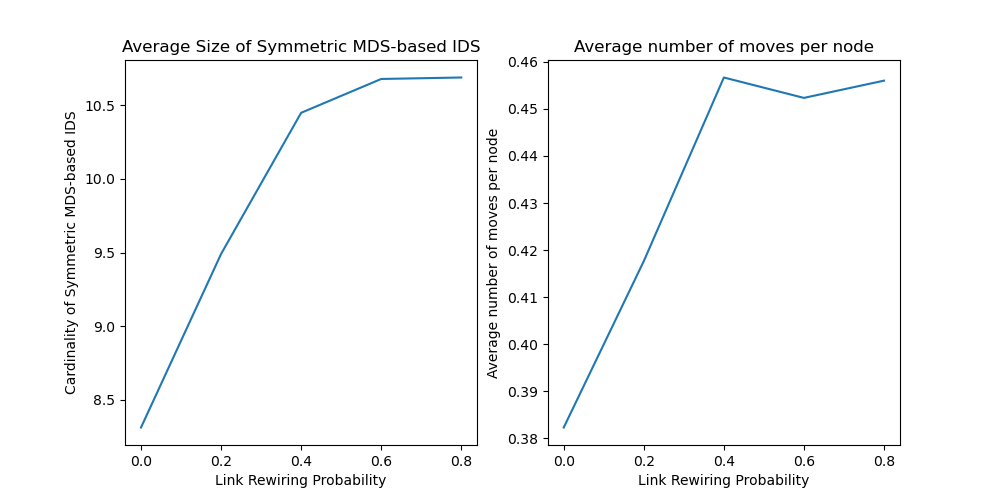
由圖21.可以得知當rewiring probability越大時，cardinality會越大，因為graph被mixed得更好，有助於找到更多node符合independence及dominance性質。當rewiring probability越大時，平均每個node進行strategy改變的次數越大，因為圖變得更沒規則性，要花更多步驟找到Nash Equilibrium。

圖21. Symmetric MDS-based IDS Game performance

圖22.中每個node上數字代表node編號。紅點為strategy=1的點；綠點為strategy=0的點。

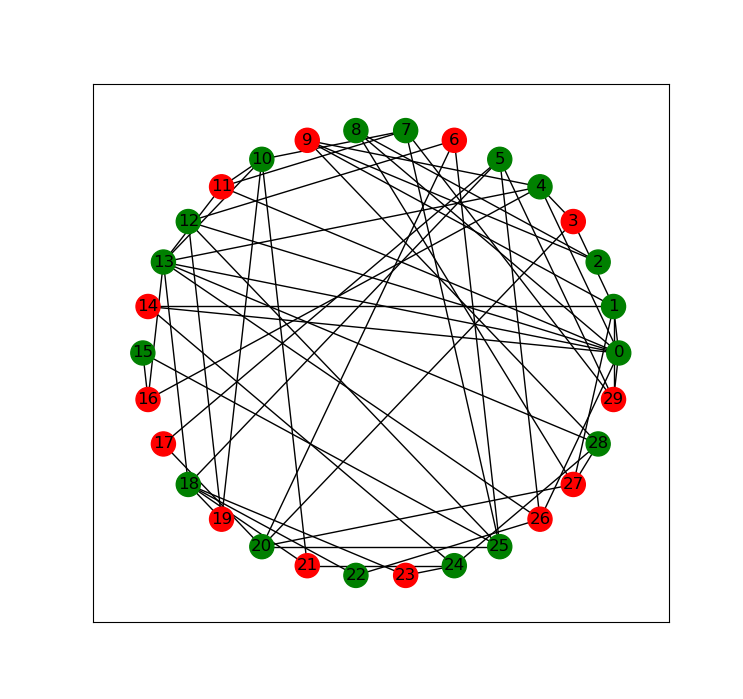


圖22. Symmetric MDS-based IDS Game example

1. **Requirement 2 (Matching Game)**
   1. **Utility Function and Best Response**

Utility function：

其中、。

當node i是matched時，給予最高utility 2；當node i選擇的對象並沒有選任何人時，給予一點獎賞utility為1：當node i沒選擇任何其他點時，給予utility 0；當node i選擇的對象選了其他的node而非i時，給它懲罰，所以utility設-1。

由以上utility function，我們可以推論出best response：

其中為隨機挑選中的任一node。

當有neighbor node選擇node i時，best response是選擇回去，形成matched pair；當沒有neighbor node選擇node i，但neighbor node全部都沒選擇任何node時，best response就隨機挑一個neighbor選，因為它的neighbor還有機會跟它形成matched pair；在其餘的狀況，best response則為不選擇任何點。

* 1. **Code**

為了撰寫程式方便，我定義null strategy為-1。先以 “initialize\_strategy\_profile” function隨機給予每個node strategy。

這邊的“initialize\_strategy\_profile” function跟requirement 1的不一樣。

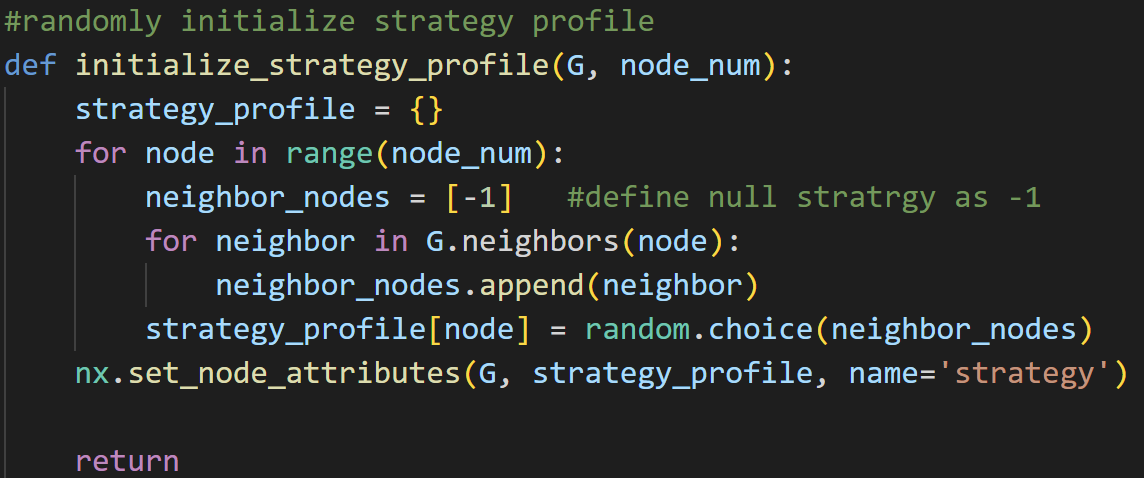


圖23. “initialize\_strategy\_profile” function

主迴圈架構與requirement 1-2相同，只差在尋找可以更改strategy增加其utility的node set是用Matching Game的best response公式，定義於 “find\_waiting\_nodes” function。

performance改以number of matched pairs測量，計算方式寫在 “calculate\_matched\_pair” function。並且在“calculate\_matched\_pair” function中，順便標記matched pairs的nodes跟weights，以利之後畫example圖。

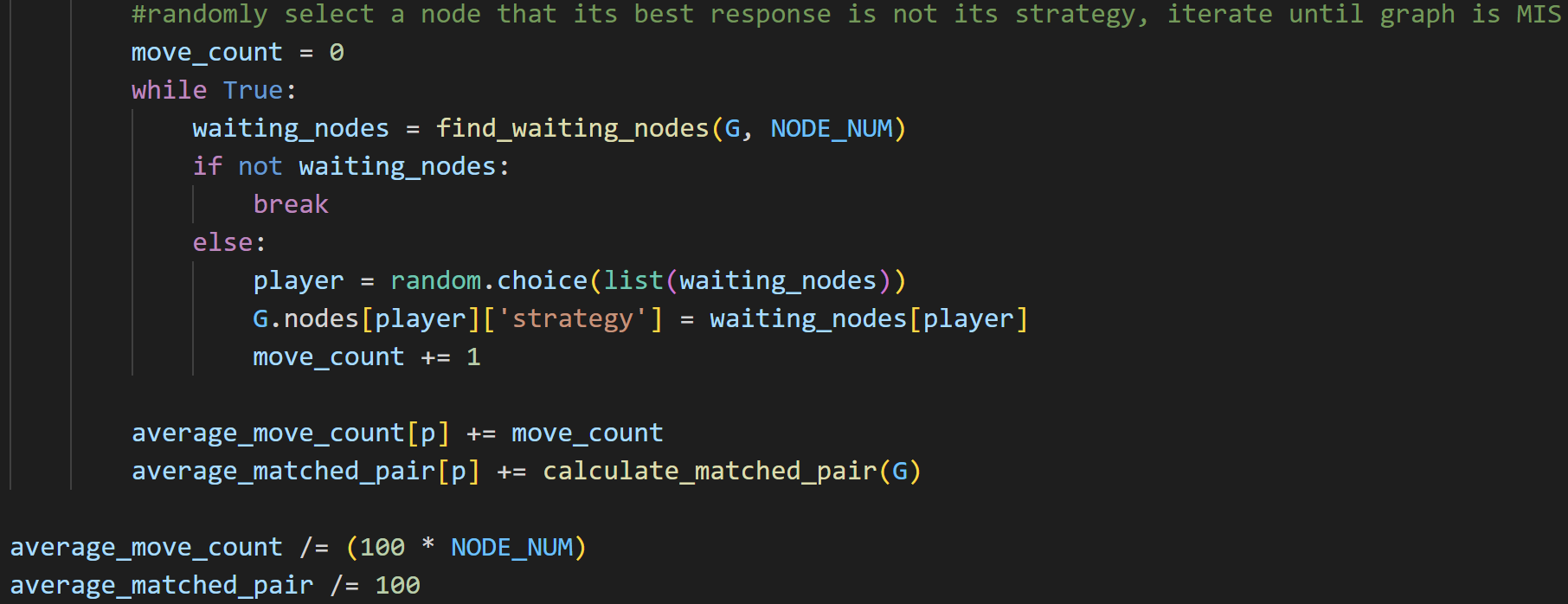
****

圖24. Matching Game主迴圈架構

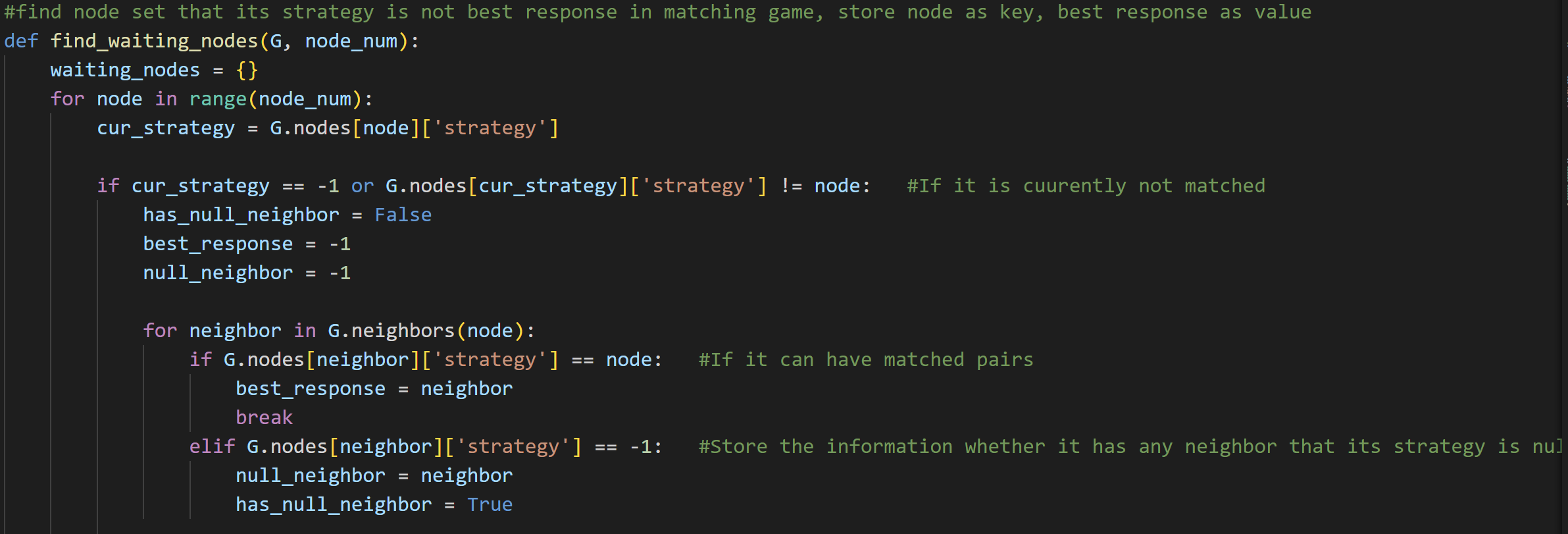


圖25. “find\_waiting\_nodes” function – (1)

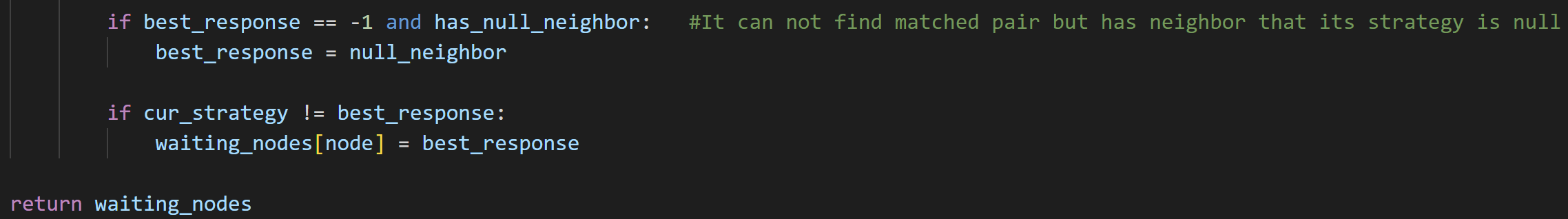


圖26. “find\_waiting\_nodes” function – (2)



圖27. “calculate\_matched\_pair” function

最後，先將node移動次數及matched pairs數目取平均，並把performance以折線圖呈現，並用 “print\_graph” function畫一張Matching Game graph觀看結果的範例。

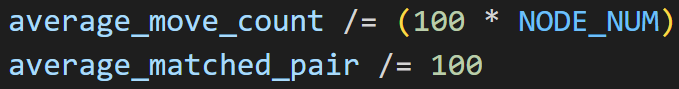


圖28. 將node移動次數及matched pairs數目取平均

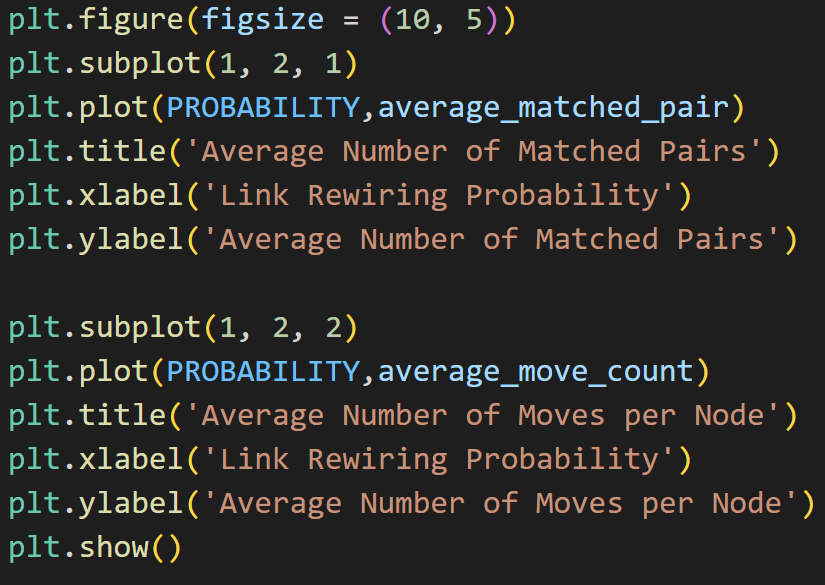


圖29. 畫Matching Game performance折線圖

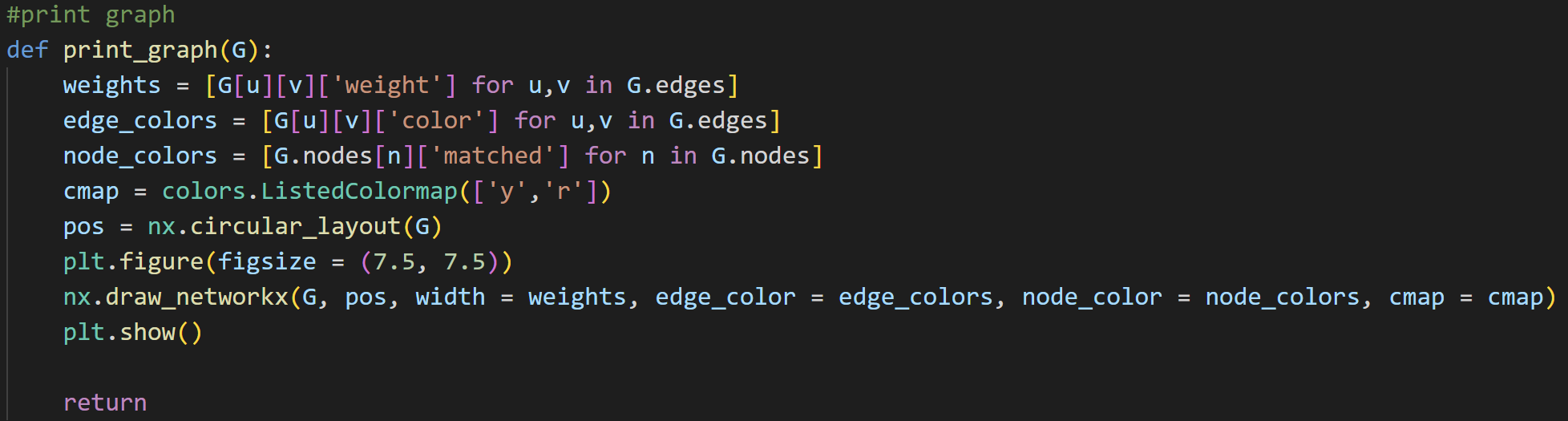


圖30. “print\_graph” function

* 1. **Result**

由圖31.可以得知當rewiring probability越大時，number of matched pairs會越小，因為graph被mixed得更好，edges雜亂的連來連過去，會降低獨立pairs的數目。當rewiring probability越大時，平均每個node進行strategy改變的次數越小，因為圖變得更沒規則性，沒有那麼多matched pairs，所以可以些微降低strategy改變的次數。

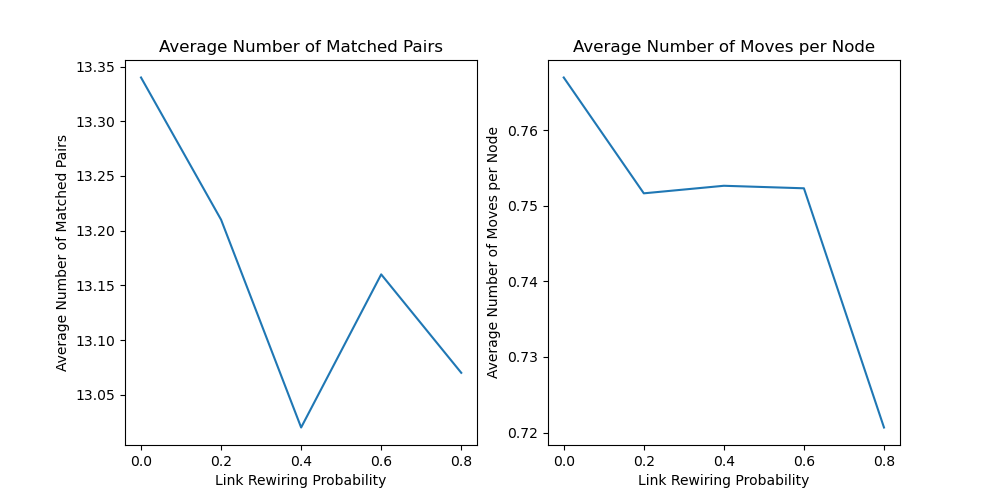


圖31. Matching Game performance

圖32.中每個node上數字代表node編號。紅點為有成功matched的點；黃點為unmatched的點。紅色的edge為matched pairs的edge；黑色的edge為unmatched pairs的edge

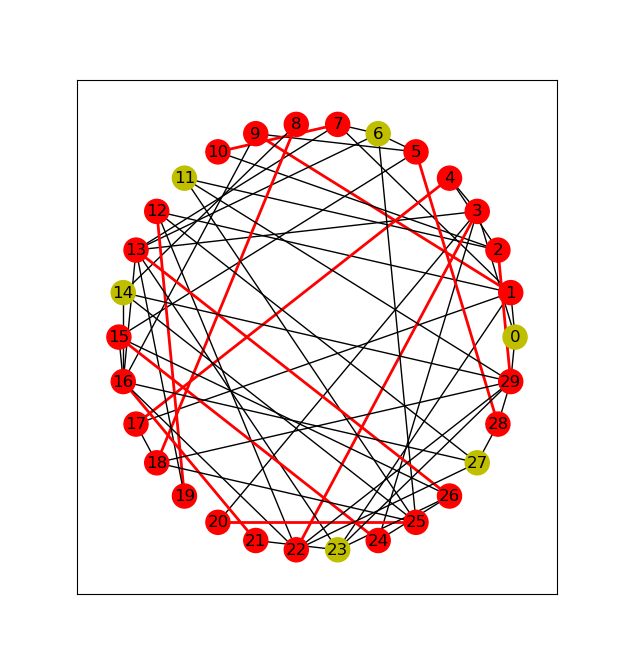


圖32. Matching Game example