Network Science HW4

電機所
R05921030 蔡仕竑

Network properties:

```
Network name: as-22july06
Type: Undirected
Nodes: 22963
Edges: 48436
```

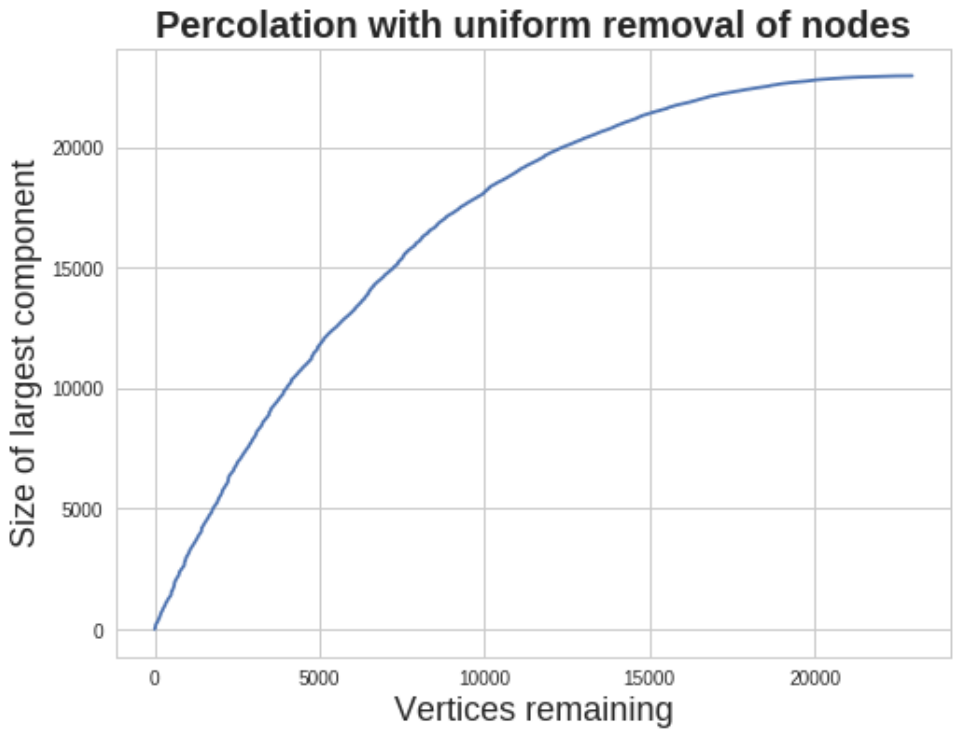
Network statistics:

```
Average degree:          4.21861255062
Density:                 0.000183721476815
Transitivity:            0.0111463838478
Clustering coefficient: 0.230447675236
```

Tools:

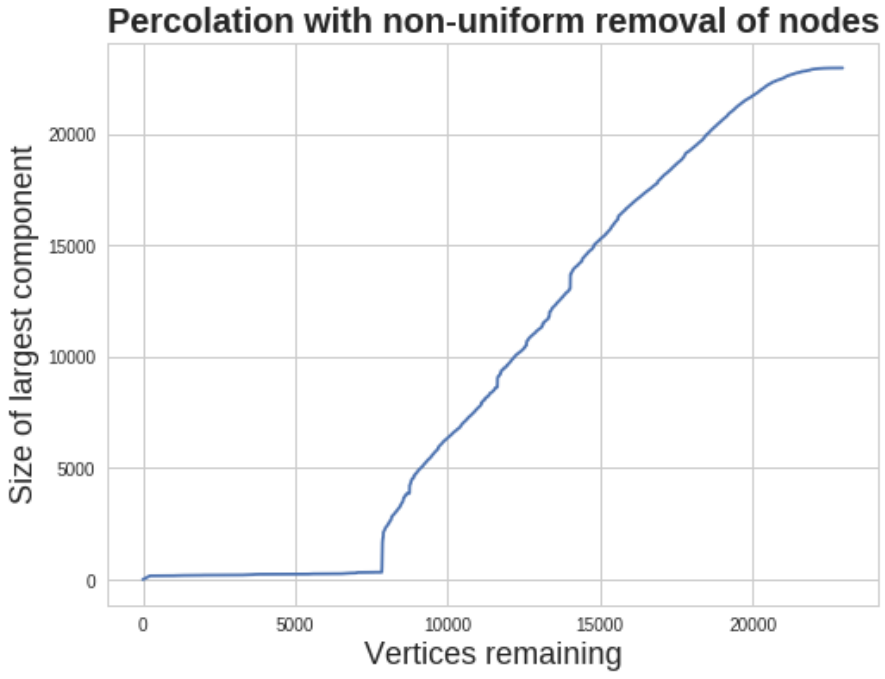
1. **Graph tool (for percolation)**
2. **Networkx (for epidemics)**
3. **numpy**

1. Percolation with uniform removal of nodes

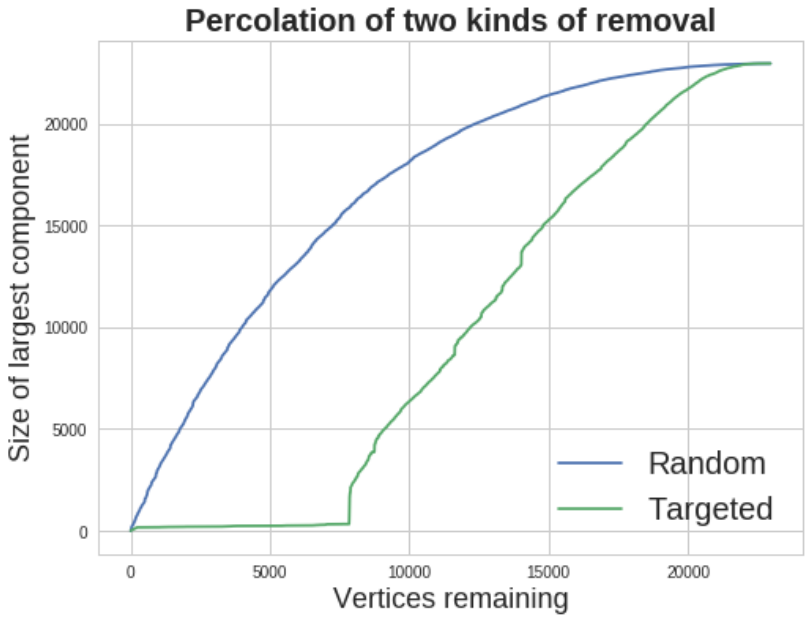


從圖形不難發現這個網路對隨機攻擊的 resilience 還滿高的，在一部分節點癱瘓掉之後還可以維持一定的 largest component size。

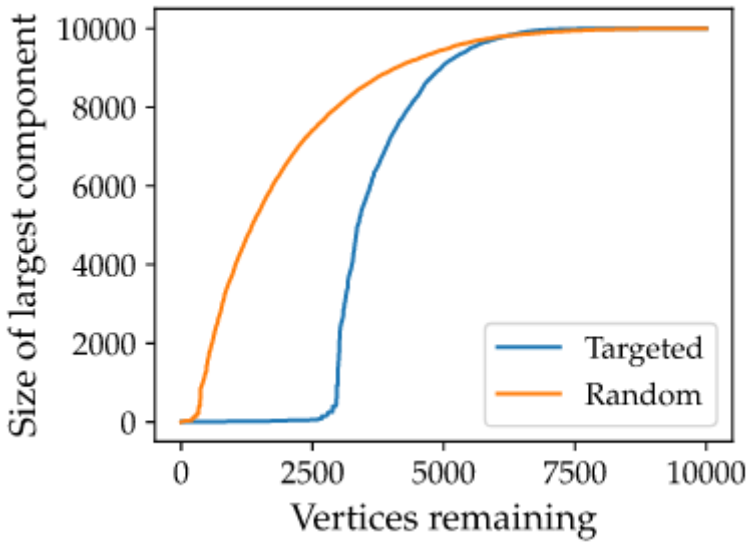
2. Percolation with non-uniform removal



從圖形可以發現有針對性的 removal(non-uniform)顯然會加速破壞這個網路，largest component size 的下降速度比起前一張圖要快很多(斜率更抖)。

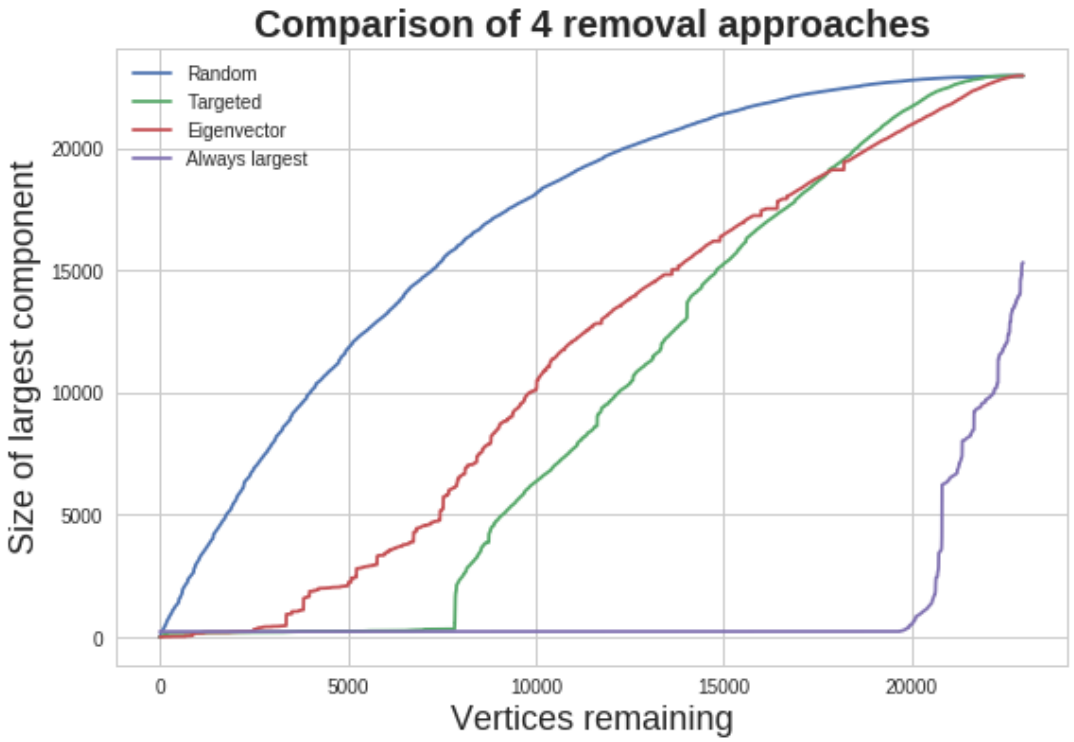


將兩張圖放在同一個畫面來比較的話，可以很明顯看出 network 崩壞速度的差別。值得一提的是，這張圖跟 graph_tool 所提供的 exponential network 範例在結構上滿接近的(如下圖)，可以推測此網路應該的 degree 分布應該類似於 exponential network。



3. Other removal approaches

我用的兩個方法分別是：
(1)根據 eigenvector centrality，從最大的開始移除
(2)動態找出當前 degree 最大的 node，予以移除



從 4 個方法的比較圖可以發現 random 是效果最差的，動態決定移除對象的效果是最好的，而依照 degree 移除則比依照 eigenvector centrality 好一點，這個結論其實不難接受，畢竟 largest component size 跟 degree 最直接相關，因此很 greedy 的每次都移除 degree 最大的節點絕對優於按照其他準則的方法。

另外值得一提的是，按照 eigenvector 來移除在過程剛開始的時候，效果比按照 degree 移除來的好，我猜這可能跟此網路本身的結構有關。

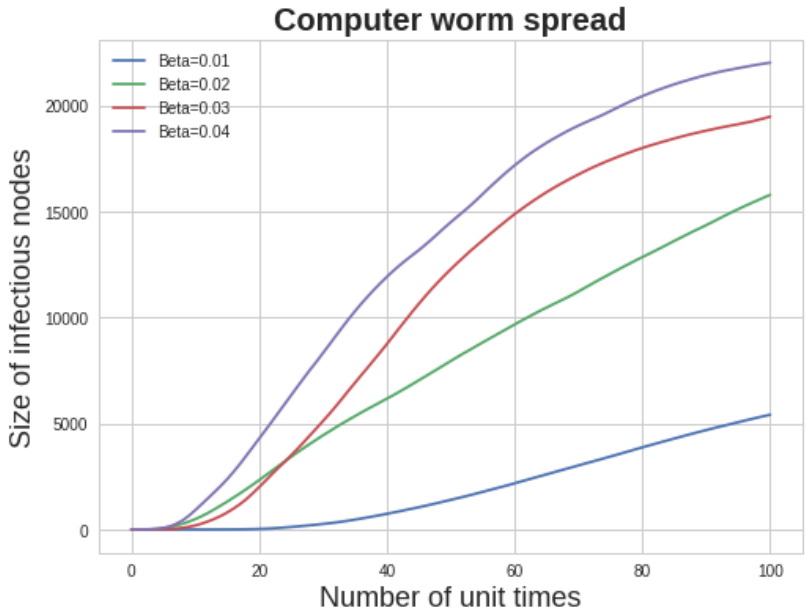
4. Epidemics 1

實作方法(有放到 github)： 因為病毒的傳播都是從鄰居開始，所以我主要是用一個 state hash table (python dictionary)來記錄每個節點是否遭受感染，並且用另一個 candidate list 來儲存那些被感染節點的鄰居(也就是下一次可能受感染的 candidate)。

首先將每個節點的狀態都初始化為 Susceptible，並且隨機挑選一點當作 0 號帶原者，將其狀態初始化為 Infectious，這個原始感染者的鄰居則是下一次可能會受感染的對象，將這些節點放入 candidate list。

接著則是開始模擬病毒散播的過程。在每個單位時間裡，所有的 candidate 都會有一定的機率被感染，他們會通過一個給定 beta 的 function，判斷是否在這個 stage 遭受感染。被感染的節點則被暫存下來，沒有被感染的則繼續維持為 candidate。在所有 candidate 都處理過之後，新被感染的節點的鄰居會被挑出來，加入下一輪的 candidate list。

此步驟會重複 100 次(總共設定 100 個單位時間)，並且對於每個 beta 值都實驗 20 個 epoch 來取平均，結果如下圖。



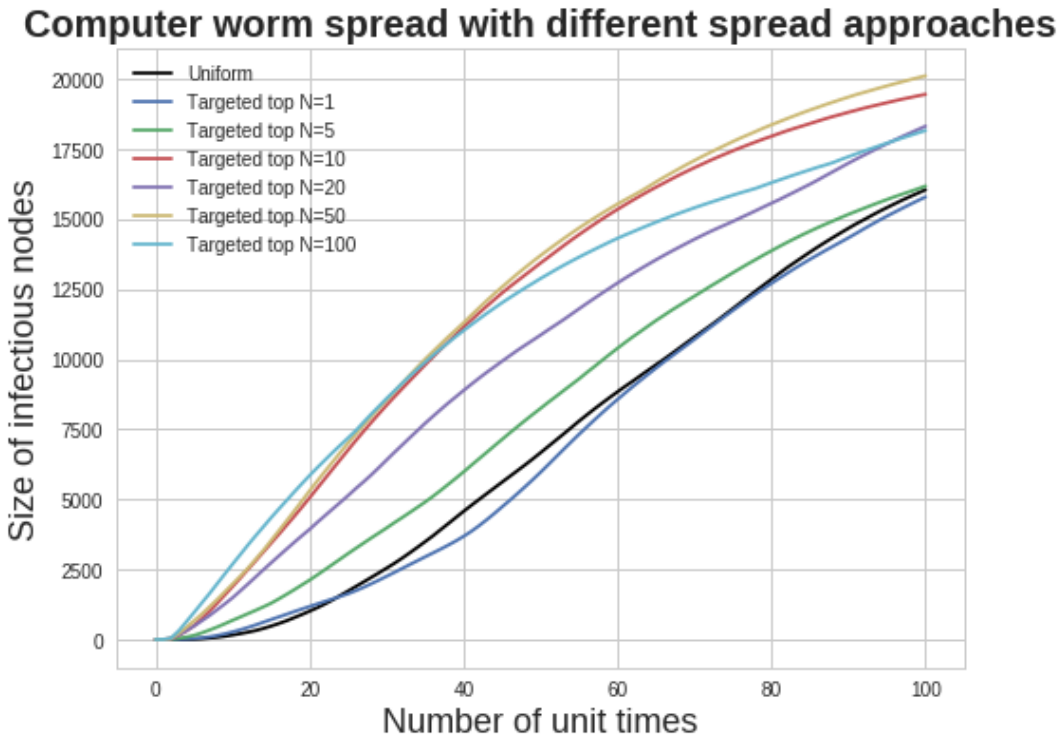
從圖上可以看出 beta 越高，感染擴散的速度越快，只是不知道為什麼 beta=0.03 在剛開始的幾個單位時間裡，感染速度會輸給 beta=0.02，我猜這可能是跟 random 挑選的起始點位置有關，如果挑到 degree 比較小的 0 號感染者，就可能會有 cold start 的問題。另外，我認為 bond percolation 基本上就是一種 epidemics 的特例，只是感染機率為 100%(beta=1)。

5. Epidemics 2

我將感染機制設定成如下：

這是個帶有記憶的 worm，這個 worm 有一個口袋名單，當它在網路中流轉，遇到口袋名單裡面的節點時，worm 就會啟動特殊機制，對這些節點有 100%的感染率，我將這些口袋人選設定為 degree 最高的 N 個節點，並對不同的 N 值去做比較。

為了平衡 beta，特殊名單裡的人選越多，剩下節點間的感染率 beta 就小，其加權平均值會跟 uniform 的狀況一樣。模擬的結果如下圖：



從圖上可以發現，當 N=1 或 5 時，其效果跟沒有特殊名單是差不多的。但當 N=50 時，這個攻擊策略的價值就會顯現出來(如圖中褐色線 v.s 黑色線)。

也就是說，犧牲對其他節點的感染力來換取對 degree 高的節點有百分百感染力顯然是靠譜的策略，這可以提升一定的感染速度。但這個交換並不適合永無止境下去，當 N=100 時，其攻擊速度就已不如 N=50，可見 N 應該是有一最適值。