

Algorithm Final Project Report



Thesis:

Zarezadeh, M., Nourani, E. & Bouyer, A. DPNLP: distance based peripheral nodes label propagation algorithm for community detection in social networks. *World Wide Web* **25**, 73–98 (2022). <https://doi.org/10.1007/s11280-021-00966-4>.

Department: Electronic and Computer Engineering

Student ID: M11102155

Name: JING-YE YANG

Outline

I. Introduction

1. Community Detection Algorithm (CDA)
2. Label Propagation Algorithm (LPA)
3. Why and How
4. Application

II. Definition

1. Core node
2. Peripheral node
3. Benefit score

III. Algorithm

1. Distance based Peripheral nodes Label Propagation (DPNLP)
2. Example

IV. Experiment and Simulation

1. Setup
2. Measurement
3. Compare Algorithm
4. Simulation Algorithm

5. Louvain Algorithm
6. Modularity (Q)
7. Normalized Mutual Information (NMI)
8. Results stability
9. Execution time

V. Conclusion

VI. Execution sample

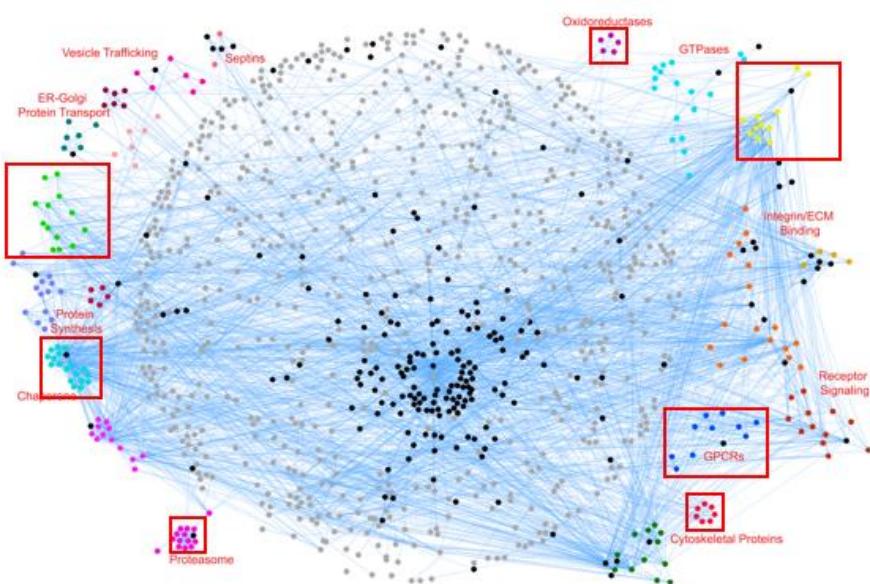
VII. Reference

I. Introduction

1. Community Detection Algorithm (CDA)

Community detection algorithm 是一個用於在複雜網路中找尋有緊密聯繫性或是有關係的一群節點。

以下圖 Fig.1 為 Protein-Protein Interaction Network [1]，其中紅色框框為經過 CDA 後找到之群。



(Fig.1)

2. Label Propagation Algorithm (LPA)

Label Propagation Algorithm [2] 為一個 2007 年提出的線性時間複雜度的 Community detection algorithm，雖然其速度快，目前被廣泛使用，但存在些許缺點，如結果不唯一且不穩定，即其執行後的結果有可能為大好或大壞，還有可能無法自行收斂。

下方 Algorithm1 為 LPA 之 pseudo code :

Algorithm 1 Label Propagation Algorithm

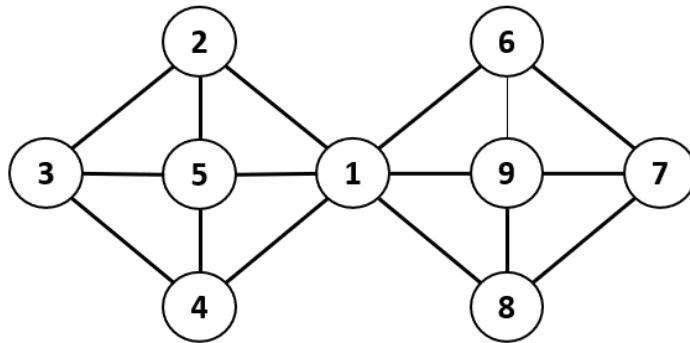
Input: $G(V, E)$: Network

Output: Label list L_u of each node

```
1: for each  $v \in V$  do
2:     initial each node an unique label;
3: end for
4: repeat
5:     randomly select a node  $v \in V$ ;
6:     get the label form  $v$ 's neighbors  $u \in \text{neigh}(v)$ ;
7:     get the most frequency label;
8:     if exist more than one most frequency label then
9:         randomly select one as node  $v$ 's dominate label;
10:    end if
11: until all label remain as the previous iteration
```

Example 1 (LPA 執行過程) :

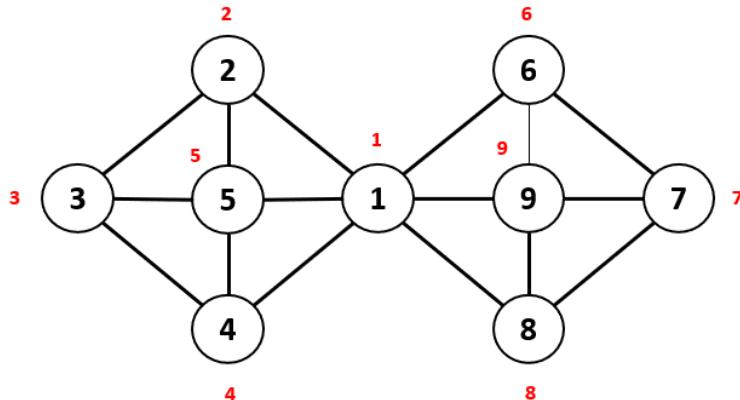
Assume G 為 9 個 node 之 network，對其做 LPA。



(Fig.2)

Step 1:

Assign 每一個節點一個獨立的 label。

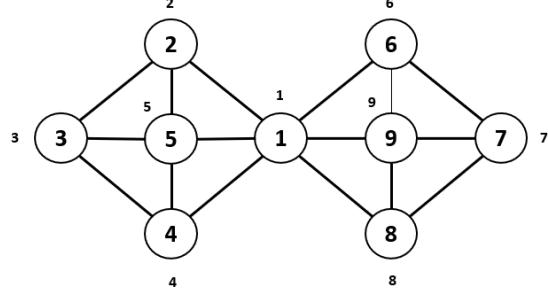


(Fig.3)

Step 2:

隨機選擇一個節點更新其 label，並收集周圍鄰居節點之 label，選擇出現次數最多之 label 當作其更新的 label，若有相同出現次數最多之 label 則隨機選一個。

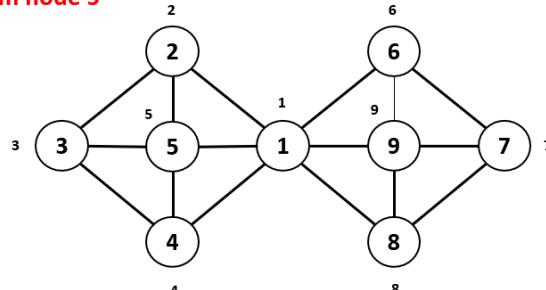
We assume start from node 5



Node 5 gets label {1, 2, 3, 4}

(Fig4)

We assume start from node 5

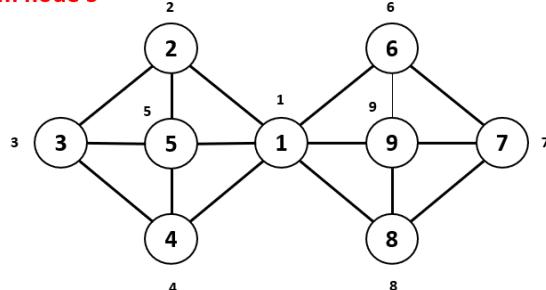


Node 5 gets label {1, 2, 3, 4}

Choose the label random if the maximum number of label are equal.

(Fig5)

We assume start from node 5



Node 5 gets label {1, 2, 3, 4}

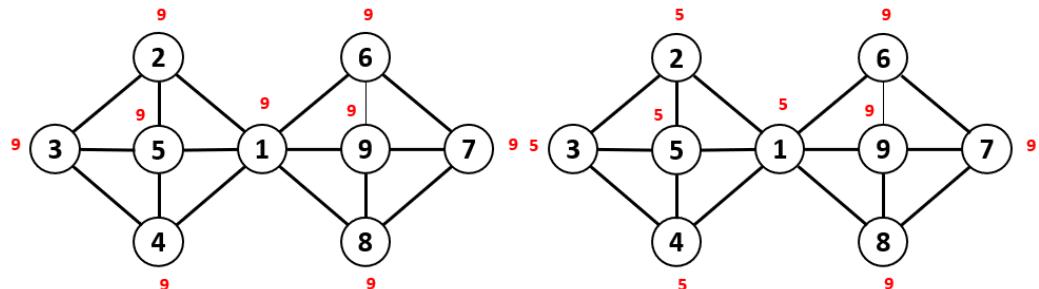
Assume we randomly choose 2 as node 5's dominate label.

(Fig6)

直到每一個節點的 label 與前一次更新相同就停止做 LPA。

Result:

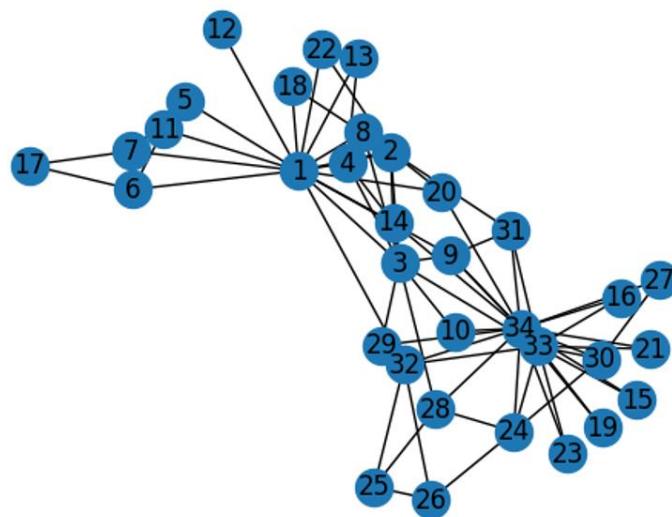
從 Fig.6 可看出，其執行結果並非唯一。



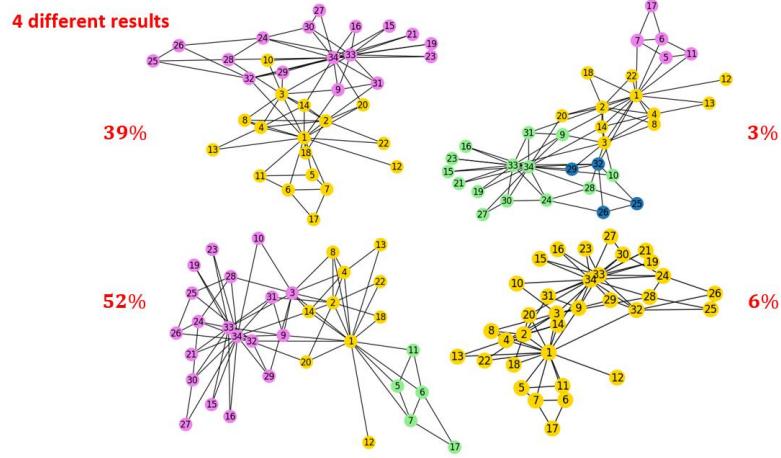
(Fig.6)

Example 2 (對 Zachary's Karate Dataset [3] 做 LPA 以及本篇改善之演算法 DPNLP)

Fig.7 為原始之 dataset，Fig8 為做完 LPA 之結果，Fig.9 為做完 DPNLP 之結果，可以看到 Fig.8 上方之百分比為我做 100 次產生之不同結果的比例。

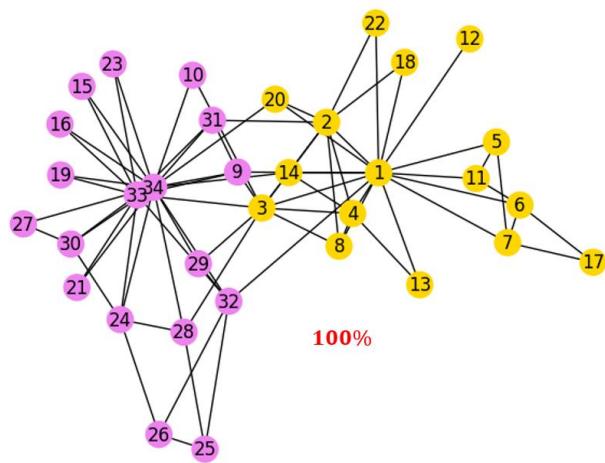


(Fig.7)



(Fig.8)

Only 1 result



(Fig.9)

3. Why and How

由於 LPA 之結果並非唯一且不穩定且低準確度，本篇 DPNLP 利用預先尋找 Initial Communities 方式增加其結果穩定性與準確度。

此外，本篇論文作者認為 LPA 的速度仍然不夠快，因此將節點分支度為 1 與 2 留至最後才進行處理，更新標籤的節點僅限於 Peripheral node，即扣除建立 Initial Communities 的 Core node 以即分支度為 1 和 2 的剩餘節點。

4. Application

本篇論文可應用於社交網絡，找尋未被發現的人與人間的相連，或適用於生物網絡，找尋相似的基因序列，如 Fig.1 所示。

II. Definition

1. Core node

Core node 即為一個網絡中較為 Dense 之節點或區域，此處的「Dense」指的是節點之分支度，本篇論文提供了 3 個條件來判斷一個節點是否足夠密集以成為 Core node。

First Condition:

$$degree(v_i) \geq \left(\frac{E}{N}\right)^2$$

其中 E 指 network 之總邊數， N 指 network 之總節點數。

判斷此節點是否相對於整張網絡來說較為 dense，為何不採用平均分之度，我認為是因為在大資料集的情況下，用平均分之度無法準確地辨別 Dense，如 Fig.10 所示，Orkut 資料集，其最大分支度為 1807，若是採用平均分之度，將會無法準確分辨出節點是否為 Dense，因此，此篇論文稍做了修正。

Network	Nodes	Edges	$2(\frac{E}{N})$	$(\frac{E}{N})^2$
Zakary's karate club [36]	34	78	4.58	5.26
Dolphins [17]	62	159	5.12	6.57
Football Collage [10]	115	613	10.66	28.41
Political Books [23]	105	441	8.4	17.64
Amazon [35]	334863	925872	5.52	7.644
DBLP [35]	317181	1,149,866	7.25	13.14
Youtube [35]	1134891	2987624	5.625	6.39
LiveJournal [35]	3997962	34,681,189	17.34	72.25
Orkut [35]	3072441	117185083	76.28	1454.71

(Fig.10)

Second Condition:

$$degree(v_i) \geq average_degree(\Gamma(v_i))$$

其中 $\Gamma(v_i)$ 指節點 v_i 之相鄰節點。

判斷此節點是否相對於周圍區域之節點更加 Dense，此判斷在於有若是有一群節點其分支度均很高，那麼不會重複選取過多之節點成為 Core node。

Third Condition:

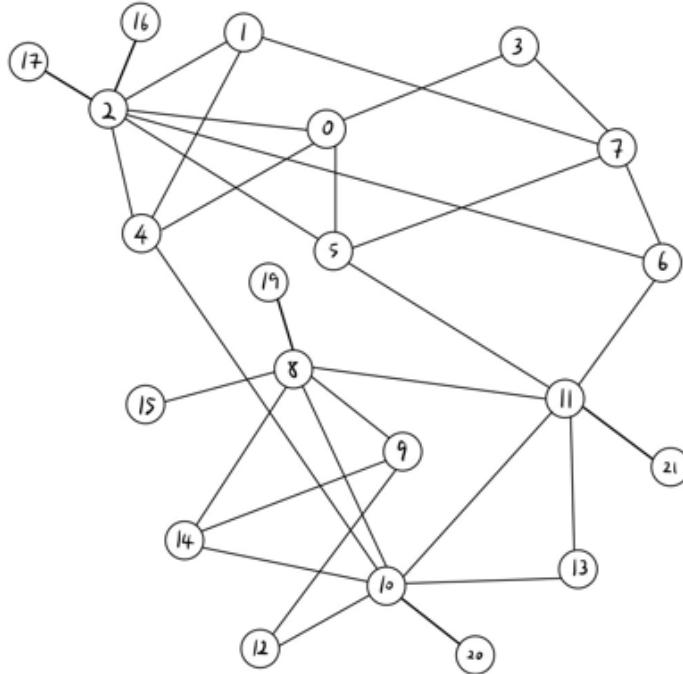
$$\frac{degree(v_i) + average_degree(\Gamma(v_i))}{2} \geq \frac{3}{2}k$$

其中 k 指此網絡之平均分之度。

此判斷主要是預防小資料集，其中整張網絡平均分支度與周圍鄰居平均分之度均不高但通過了第一與第二條件，因此，利用第三個條件判斷是否此區域為較 Dense 的部分。

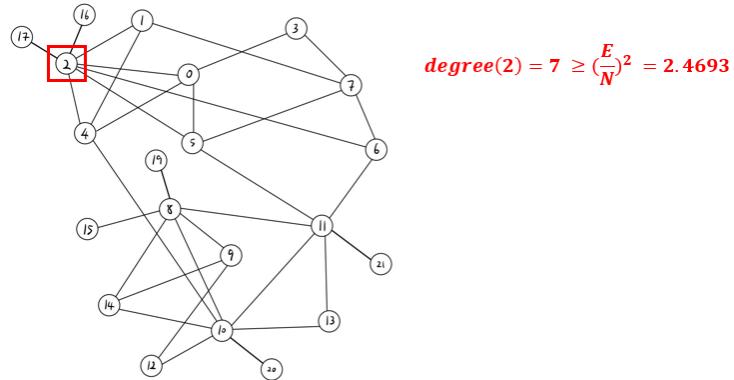
Example 3

下方 Fig.11，假設判斷此網絡節點 2 是否能成為 Core node。

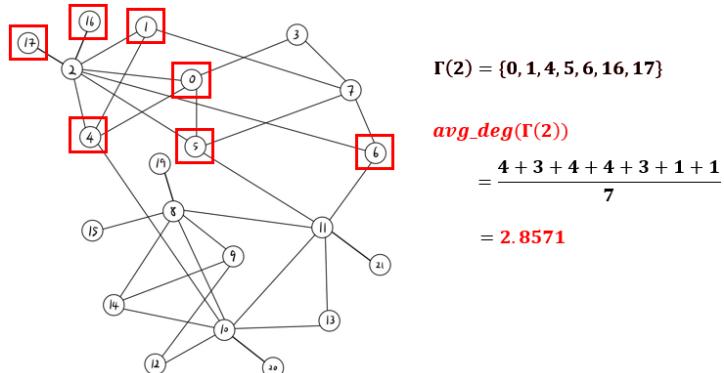


(Fig.11)

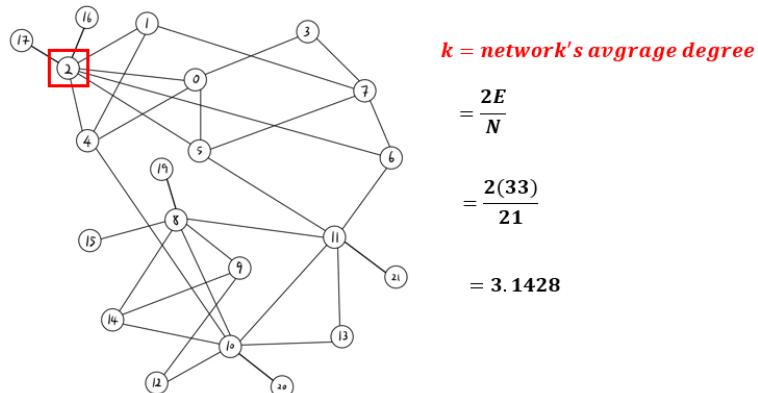
判斷條件一，節點 2 分之度大於等於總邊數除以總節點數之平方，如 Fig.12 所示，通過條件一，再判斷條件二，其鄰居平均總分之度為 2.8571，如 Fig.13 所示，節點 2 分之度也大於等於其鄰居平均總分之度，通過條件二，最後判斷條件三，網絡平均分之度為 3.1428，如 Fig.14 所示，節點 2 分之度與其周圍鄰居平均總分之度的平均也大於等於網絡平均分之度的 1.5 倍，通過條件三，綜合上述，節點二被判斷為 Core node，如 Fig.15 所示。



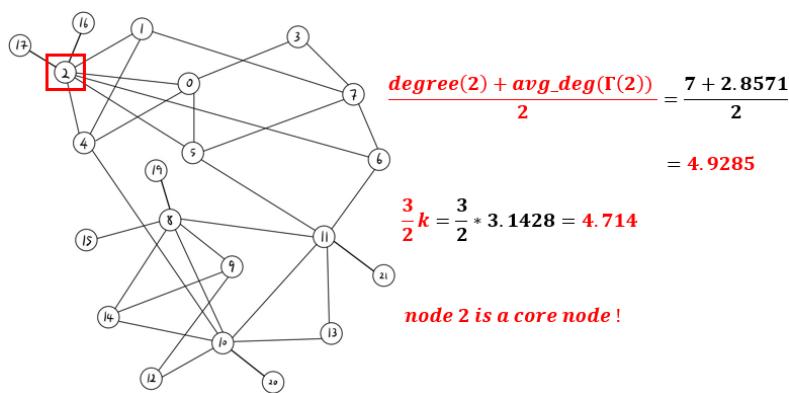
(Fig.12)



(Fig.13)



(Fig.14)



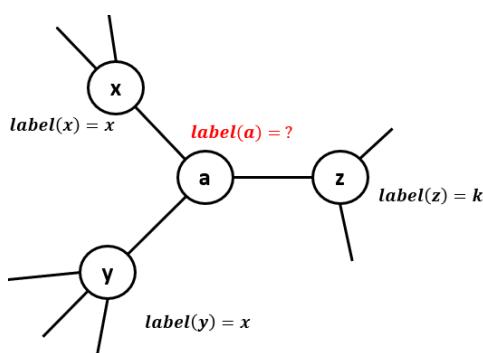
(Fig.15)

2. Peripheral node

Peripheral node 為一節點其所屬 label 與其周圍鄰居不盡相同，根據本篇論文告訴我們從以往的實驗可得出，若一個節點與其周圍鄰居之 label 有 70% 不相同，則為 Peripheral node。簡單來說 Peripheral node 就是指其 label 還會因為周圍鄰居之 label 不盡相同所以會再改變，即還會再持續更新其 label 之節點。

3. Benefit score

一個節點周圍會存在許多不同 label，那就會產生一個問題，如 Fig.16 所示，該如何選取更新之 label，本篇論文提出了一個公式，稱為 Benefit Score，解決了此問題。節點會選擇具有最高 Benefit Score 之 label 作為其節點之更新 label，若存在一個以上最高 Benefit Score 則會隨機選取一個，但經過實驗發現此情況幾乎不會發生。



(Fig.16)

Benefit Score:

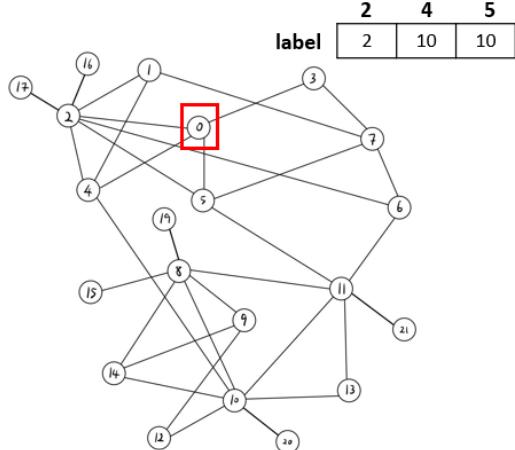
$$\begin{aligned} & \text{benefit score}(v, C_i) \\ &= \frac{t - d(v, C_i)}{t} * [(K * L(v, C_i)) + \text{sum_deg}(v, C_i)] \end{aligned}$$

上述公式中 v 為判斷之節點， C_i 為其周圍之不同 label，即不同所屬群， t 為一可調變數，此處由於社群網絡分析中一個定理，六步距離，即每一個人與一陌生人之間的最短距離平均為六步遠，因此此處將 t 設為 6， $d(v, C_i)$ 為節點 v 與其所屬群 C_i 的中心節點，即 Core node，之最短距離， K 為整個網絡之平均分之度， $L(v, C_i)$ 為節點 v 與其所屬群 C_i 之間連接邊數， $\text{sum_deg}(v, C_i)$ 為其周圍鄰居節點屬於群 C_i 之總分之度。

Benefit score 越高則表示此節點越屬於此群，下方為一計算 *Benefit score* 之範例。

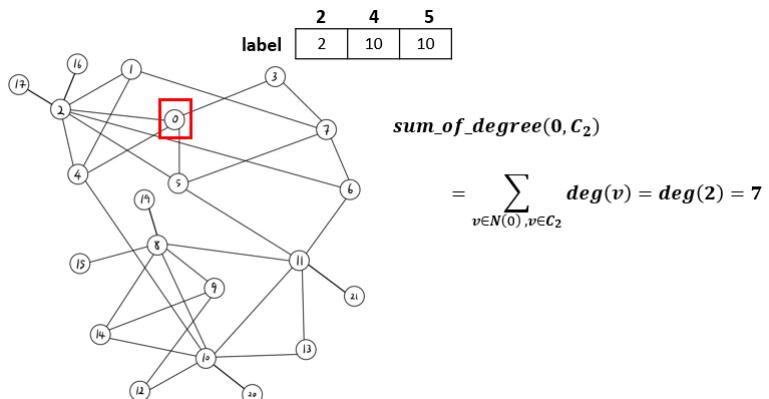
Example

判斷此網絡節點 0 之所數 label，即所屬群，計算 $\text{Benefit score}(0, C_2)$ ，2 為其周圍鄰居之 label 其中之一，假設其周圍鄰居之 label 已得到如下方 Fig.17 所示

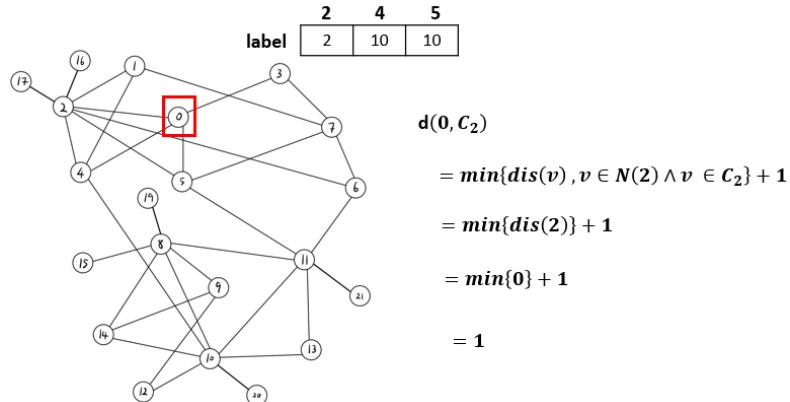


(Fig.17)

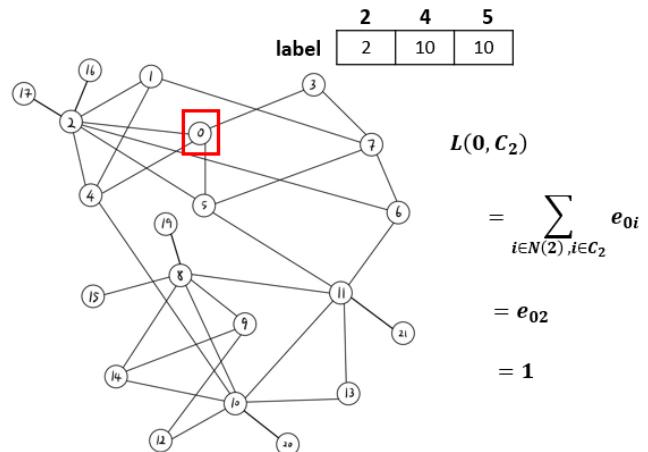
計算 $sum_deg(v, C_2)$ 如 Fig.18 所示，節點 2 周圍屬於群 2 之總結分之度為 7，計算 $d(v, C_2)$ 如 Fig.19 所示，其實作層面會利用一個陣列儲存所有節點到其所屬群之中心節點之距離，並且每次更新 label，此陣列中之值也會更新，更新方法為遍歷有者相同所屬群的周圍鄰居，從中選取最小值再加一即可，計算 $L(v, C_i)$ 如 Fig.20 所示，節點 0 與群 2 之間僅有一條邊相連接，其整個網絡平均分之度 K 可以回看 Fig14。最後其計算 Benefit score 之過程如 Fig.21 所示，得到值 8.4523。



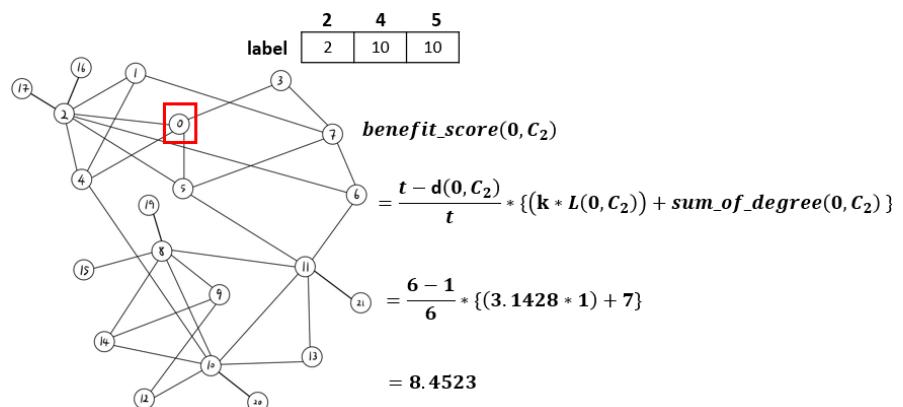
(Fig.18)



(Fig.19)



(Fig.20)



(Fig.21)

III. Algorithm

1. Distance based Peripheral nodes Label Propagation (DPNLP)

下方 Algorithm 2 為 DPNLP 之 pseudo code :

Algorithm 2 DPNLP

Input: $G(V, E)$: Network
Output: Label list L_u of each node

1: create set $deg_{1,2}$, $core$, $peripheral$;
2: **for** each $v \in V$ **do**
3: **if** $deg(v) \leq 2$ **then**
4: push node into $deg_{1,2}$
5: **end if**
6: **end for**
7: **for** each $v \in V - deg_{1,2}$ **do**
8: initial each node v an unique label;
9: **end for**
10: **for** each $v \in V - deg_{1,2}$ **do**
11: **if** $deg(v) \geq (\frac{E}{N})^2$ **then**
12: **if** $deg(v) \geq avg_deg(\Gamma(v))$ **then**
13: **if** $\frac{deg(v)+avg_deg(\Gamma(v))}{2} \geq \frac{3}{2}k$ **then**
14: push v into $core$;
15: $PropagateLabel(v, core, G);$ \triangleright Algorithm 3
16: **else**
17: push v into $peripheral$;
18: **end if**
19: **else**
20: push v into $peripheral$;
21: **end if**
22: **else**
23: push v into $peripheral$;
24: **end if**
25: **end for**
26: **repeat**
27: randomly select a node v from $peripheral$;
28: **for** each label $l \in label(\Gamma(v))$ **do**
29: compute $Benefit_score$ of each label l ; \triangleright Measurement at define
30: **end for**
31: choose label l with max $Benefit_score$ as node v 's update label;
32: **for** each node $u \in \Gamma(v)$ **do**
33: **if** $label(u)$ is 70% different to $label(\Gamma(u))$ **then**
34: push node u into $peripheral$;
35: **end if**
36: **end for**
37: **until** $peripheral \neq \phi$
38: **for** node $v \in deg_{1,2}$ **do**
39: **if** $deg(v) = 1$ **then**
40: $label(v) \leftarrow label(\Gamma(v));$
41: **else**
42: **for** each label $l \in label(\Gamma(v))$ **do**
43: compute $Benefit_score$ of each label l ;
44: **end for**
45: choose label l with max $Benefit_score$ as node v 's update label;
46: **end if**
47: **end for**

如 Algorithm 2 所示，DPNLP 為基於 LPA 做改良，用於無方向無權位之網絡，首先會先移除分支度為 1 或 2 之節點 (第 2-6 行)然後給予剩下節點獨立標籤 (第 7-9 行)，之後會處理所有除了分支度為 1 與 2 之節點，利用前面所介紹之三個判斷條件判斷這些節點屬於 Core node 亦或是 Peripheral node，若發現為 Core node，則呼叫 Algorithm 3 傳遞 Core node 上之標籤，以建立 Community 之雛形 (第 10-25 行)。再對 Peripheral node 利用前面所講之判斷更新標籤公式，Benefit score，進行 LPA (第 26-37 行)，最後處理分支度為 1 或 2 之節點 (第 38-47 行)。

下方 Fig.22 為 Algorithm 2 之檢疫流程圖：

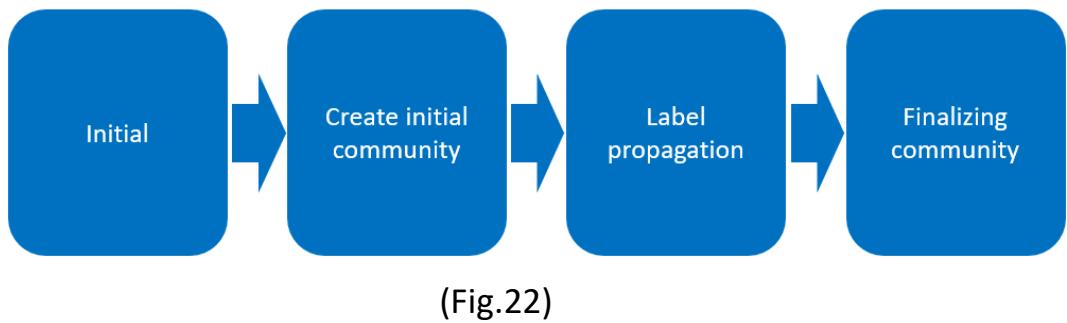


Fig.22 中 Initial 部分代表 Algorithm 2 中第 1 到 9 行，而 Create initial community 則是第 10 到 25 行，之後進行之 Label propagation 為第 26 到 37 行，最後 Finalizing community 為第 38 到 47 行。

下方 Algorithm 3 為 DPNLP 呼叫功能之 pseudo code :

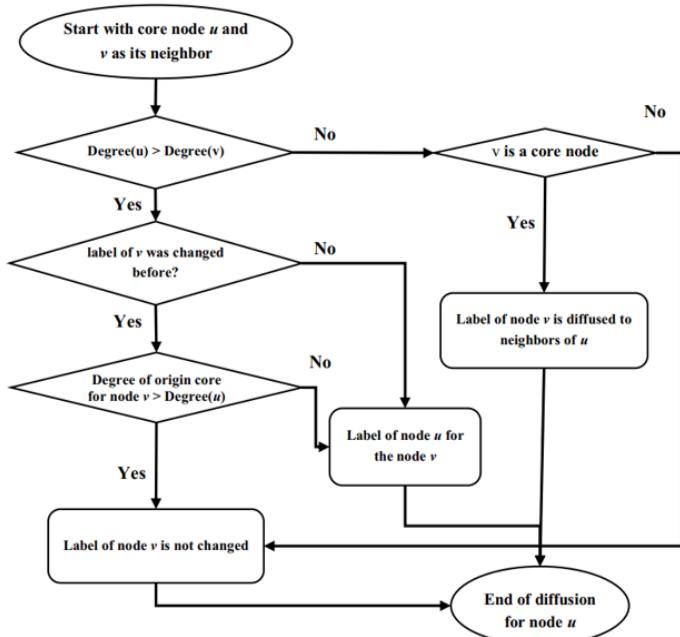
Algorithm 3 PropagateLabel

Input: node v from core , core , $G(V, E)$: Network
Output: initial community

```

1: for each node  $u \in \Gamma(v)$  do
2:   if  $\deg(v) > \deg(u)$  then
3:     if  $\text{label}(u) \neq u$  then
4:       if  $\deg(\text{label}(u)) > \deg(v)$  then
5:          $\text{label}(u)$  remain;
6:       else
7:          $\text{label}(u) \leftarrow \text{label}(v)$ ;
8:       end if
9:     else
10:     $\text{label}(u) \leftarrow \text{label}(v)$ ;
11:   end if
12: else
13:   if  $u \in \text{core}$  then
14:     for each  $w \in \Gamma(v)$  do
15:        $\text{label}(w) \leftarrow \text{label}(u)$ ;
16:     end for
17:   else
18:      $\text{label}(u)$  remain;
19:   end if
20: end if
21: end for

```



(Fig.23)

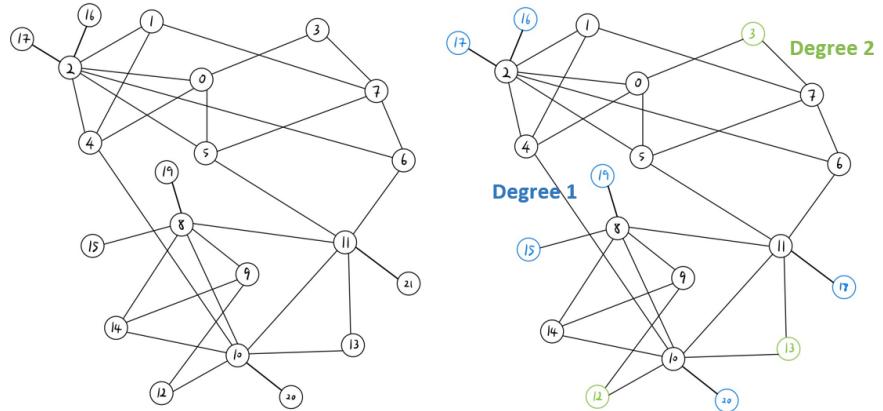
上方 Algorithm 3 本篇論文有提供其演算法之流程圖 Fig.23 ,

以幫助讀者容易理解。

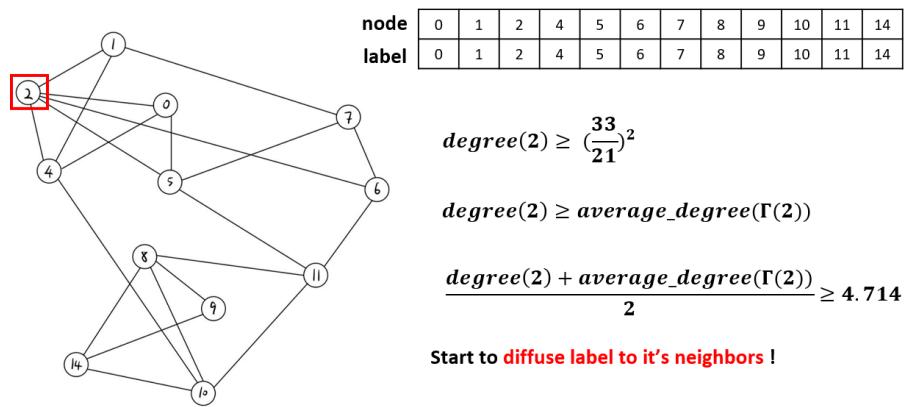
Algorithm 3 為傳遞 Core node 上之標籤，並建立 Community 之雛形，主要用分支度來判斷 Core node 是否能夠支配其周圍鄰居之節點，其中會考慮欲支配節點是否有被其它 Core node 支配過，如第 4 行所示，較為特別的部分為第 14 到 16 行，為將兩 core node 建立之 communities 雜型進行合併。

2. Example

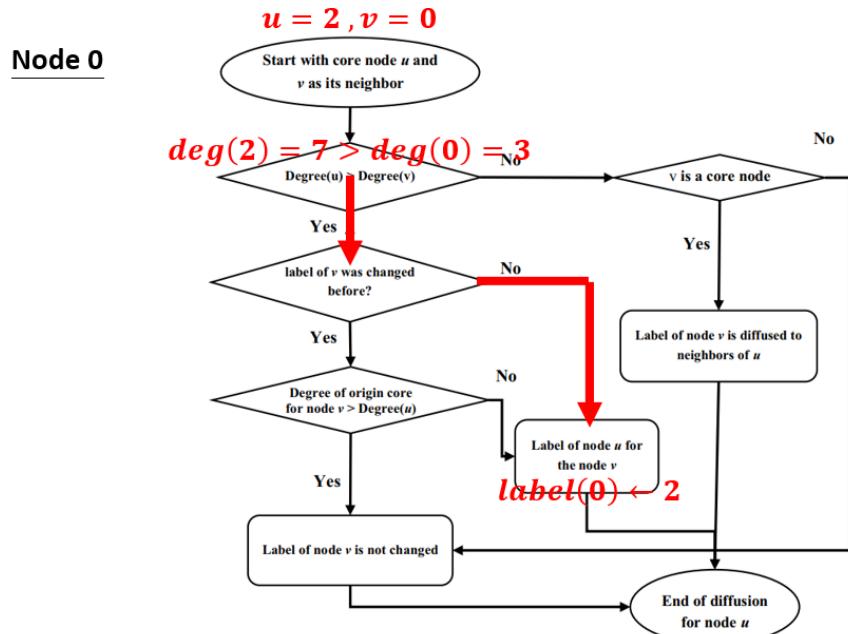
以 21 個節點與 33 條邊之無方向無權位之網絡為例，首先對其移除分支度為 1 或 2 之節點，如 Fig.24 所示，之後給定剩餘節點獨立標籤並且開始判斷哪些節點為 Core node，找到節點 2 為 Core node，如 Fig.25 所示，於是乎開始傳遞其標籤給其周圍節點產生雛型社區，首先選到節點 2 之鄰居節點 0，由於其分支度小於節點 2 且尚未被支配過，所以可以直接被節點 2 支配，如 Fig.26 所示，其它節點同理，其結果如 Fig.27 所示，接下來找到節點 8 為 Core node，如 Fig.28 所示，開始傳遞標籤，選到節點 8 之鄰居節點 9，與前面相同，再選到節點 10，其分支度大於節點 8 且還未知是否之後會判斷為 Core node，於是保留，如 Fig.29 所示。



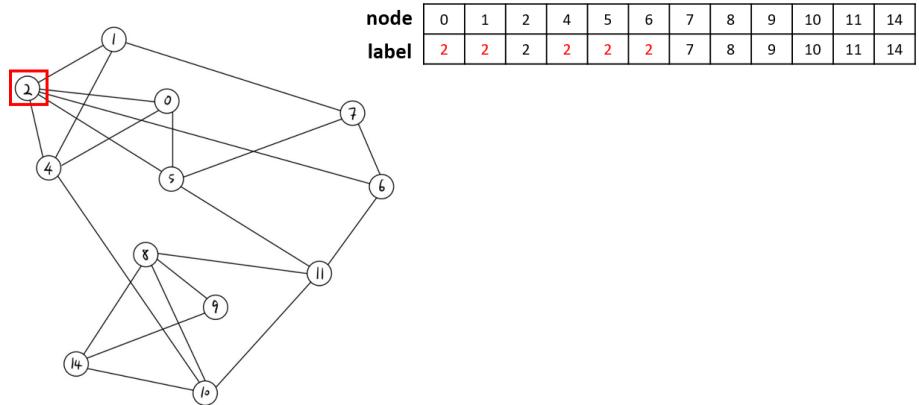
(Fig.24)



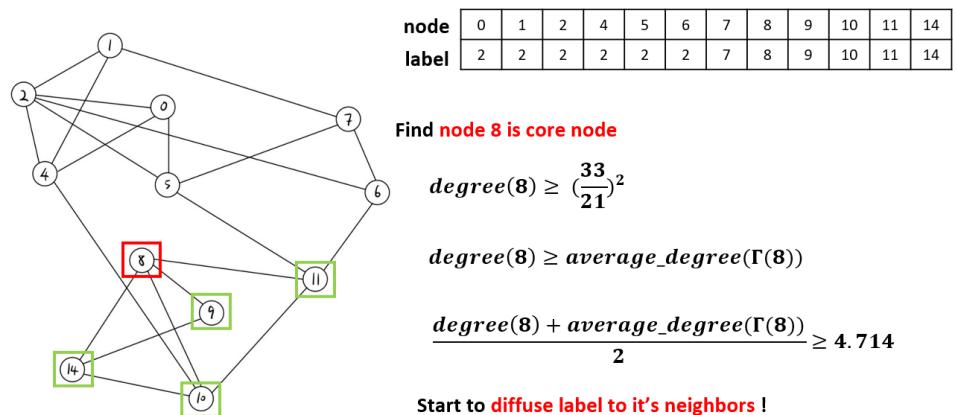
(Fig.25)



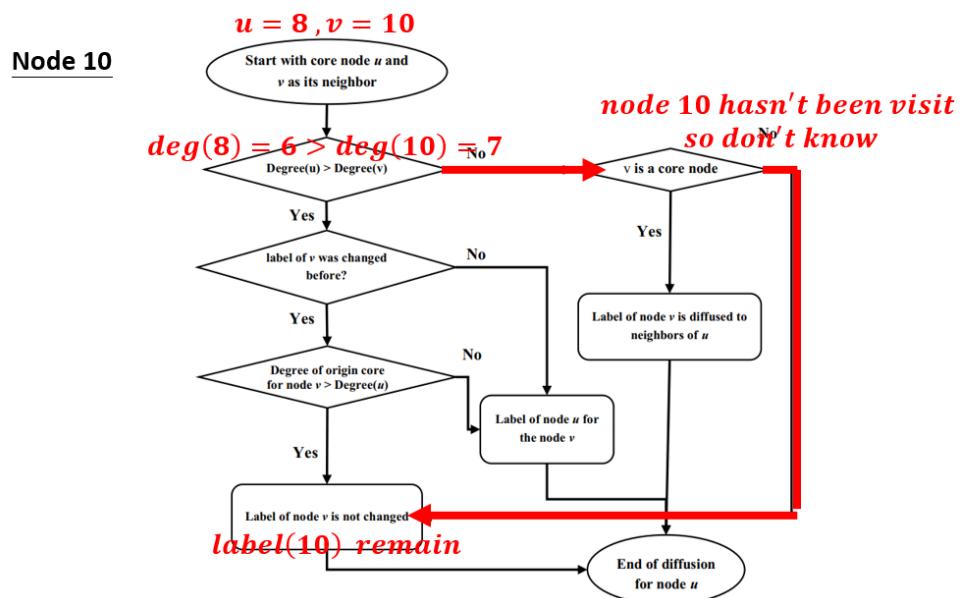
(Fig.26)



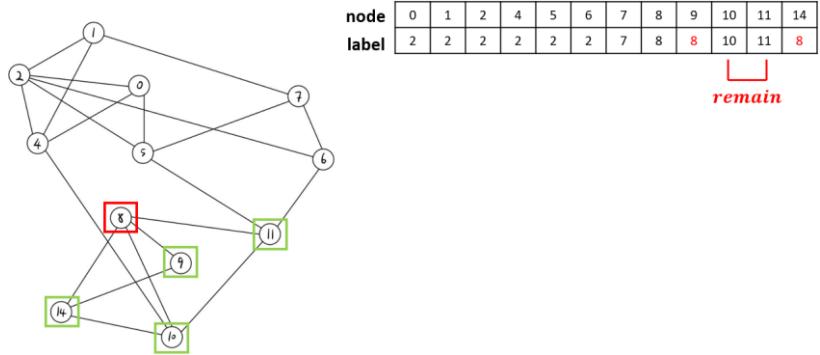
(Fig.27)



(Fig.28)

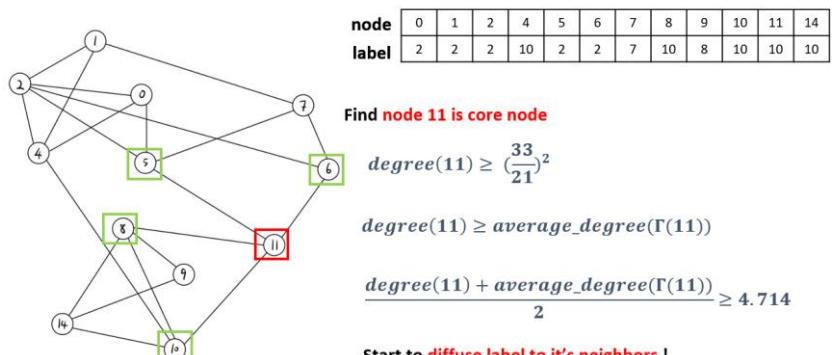


(Fig.29)

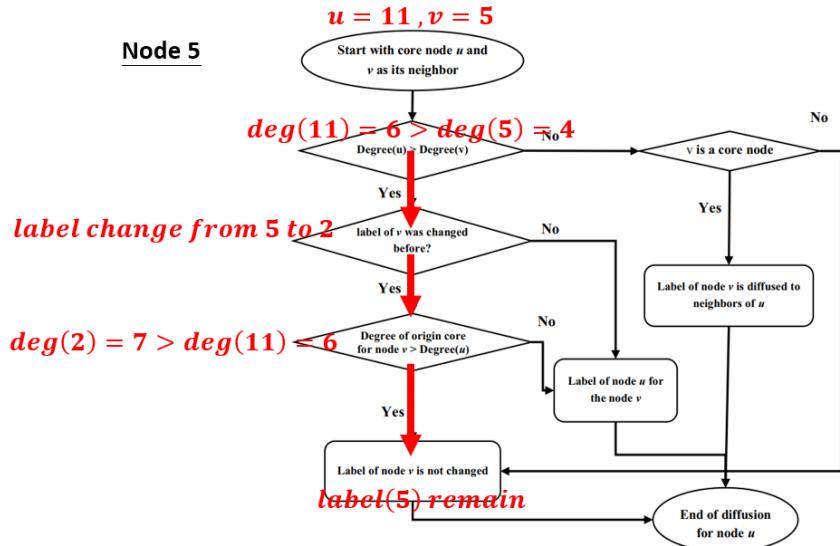


(Fig.30)

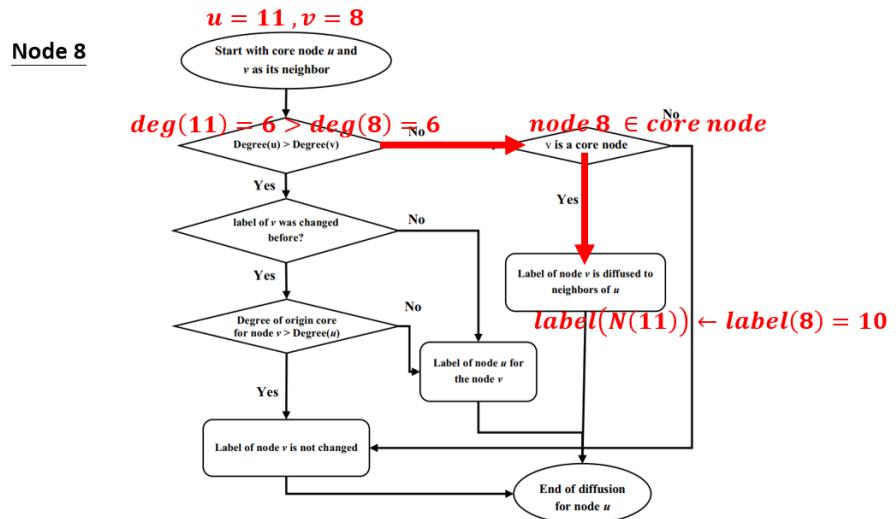
節點 8 傳遞標籤之最後結果如 Fig.30 所示，之後再找到節點 10 與節點 11 為 Core node，節點 10 傳遞標籤同前面敘述之方法，節點 11 有些與前面不同處將特別說明，可以看到 Fig.31，節點 11 被找到為 Core node 並且開始傳遞其標籤給鄰居，首先傳遞給節點 11 之鄰居節點 5，節點 5 之標籤更新過且節點 11 之分支度未大於目前支配節點 5 之節點 2，於是節點 5 之標籤保持不變，如 Fig.32 所示，再找到節點 8 為節點 11 之鄰居，由於節點 8 為 Core node，所以將兩社區雛型合併，如 Fig.33 所示，最終結果如 Fig.34 所示，除了判斷為 Core node 之節點以外，其餘節點均為 Peripheral node。



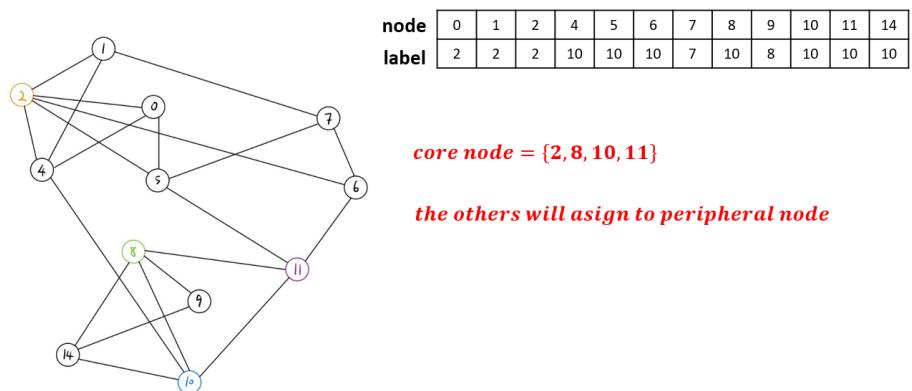
(Fig.31)



(Fig.32)

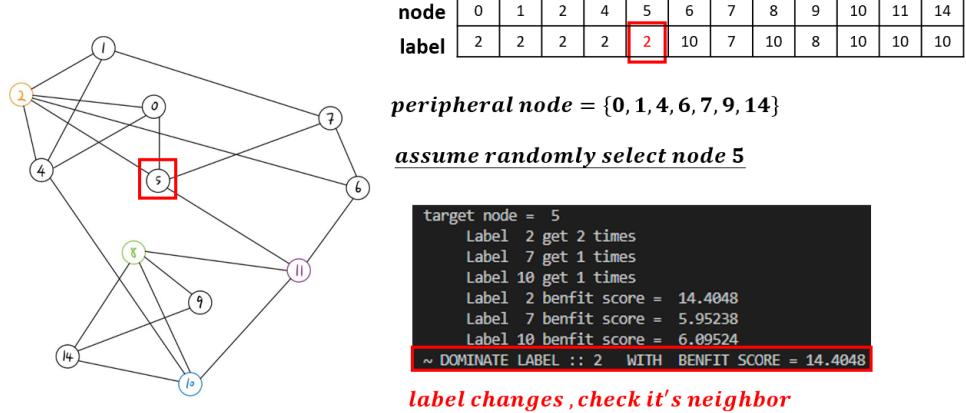


(Fig.33)

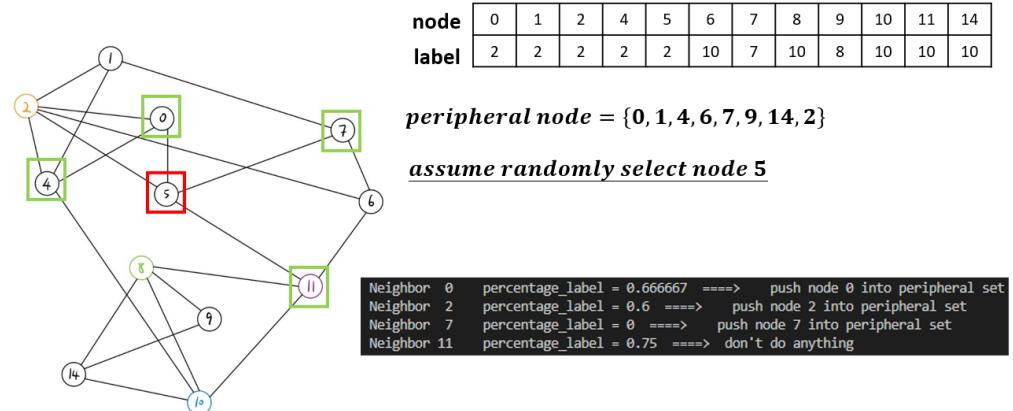


(Fig.34)

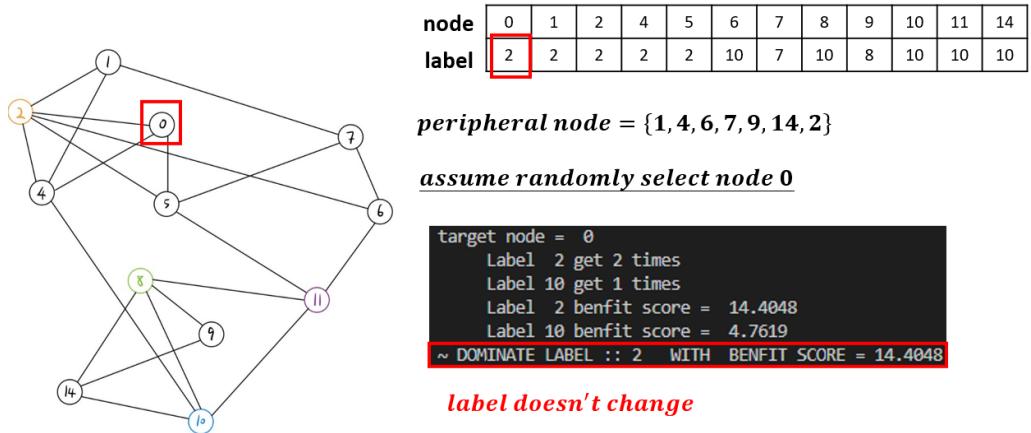
接下來開始對 Peripheral node 做 LPA，假設隨機選到節點 5 當作欲更新標籤之節點並開始取得節點 5 周圍之標籤，再計算節點 5 對周圍標籤之 Benefit score，即節點 5 有多屬於哪一個標籤，最後取具有最大 Benefit score 之標籤當作其更新標籤，如 Fig.35 所示，更新完節點 5 之標籤後，考慮節點 5 周圍鄰居是否會屬於 Peripheral node，即考慮因為節點 5 標籤更新造成其周圍鄰居之後是否有可能也會因此再度更新，將屬於 Peripheral node 之節點再度送回存放 Peripheral node 之集合，如 Fig.36 所示。之後再假設隨機選到節點 2 當作欲更新標籤之節點，發現其標籤不需要更新，於是也不需要考慮其周圍鄰居節點是否需要重新做標籤更新，如 Fig.37，直到存放 Peripheral node 之集合為空，則會停止做 LPA。最後會處理分支度為 1 或 2 之節點，分支度為 1 之節點標籤直接接受其鄰居之標籤，而分支度為 2 之節點則會考慮其周圍兩個鄰居之標籤，再透過計算 Benefit score 之方式決定其更新標籤。



(Fig.35)



(Fig.36)



(Fig.37)

IV. Experiment and Simulation

1. Setup

下方 Fig38 中，左邊表格為論文之實驗平台與實作語言，右邊為我做模擬的實驗平台與實作語言。

Thesis platform :		My simulation platform :	
CPU	Intel i5	CPU	Intel i7-12700
RAM	12 GB	RAM	16 GB
OS	-	OS	Ubuntu 20.04.5 LTS
Implement	Python	Implement	C++
		Compiler	gcc 9.4.0

(Fig.38)

下方 Fig.39 為本篇論文所使用之真實資料集，其中紅色框框為具有真實結果的資料集 [3-7]，也是我這次模擬所跑的資料集，所有資料集之取得之網站會附於報告最後。

Network	Nodes	Edges	Number of communities
Zakary's karate club [36]	34	78	2
Dolphins [17]	62	159	2
Football Collage [10]	115	613	12
Political Books [23]	105	441	3
Amazon [35]	334863	925872	75149
DBLP [35]	317181	1,149,866	13447
Youtube [35]	1134891	2987624	8385
LiveJournal [35]	3997962	34,681,189	287512
Orkut [35]	3072441	117185083	6288363
Email [12]	1133	5451	Unknown
Email-Enron [14]	36692	183831	Unknown
Condmat2003 [22]	31163	121129	Unknown
Condmat2005 [21]	41421	175691	Unknown
Power grid [33]	4941	6594	Unknown
NetScience [24]	1589	2742	Unknown

(Fig.39)

2. Measurement

接下來將介紹兩種用來辨別分群效果的指標，
 $\text{Modularity}(Q)$ [8] 與 $\text{Normalized Mutual Information}$ (NMI)，
接下來將會大致介紹這兩個指標。

Modularity (Q)

其指標是用於判斷此次分群之社區結果之連接性強度，將連接性區分為社區內部與社區間(外部)，指標數值越高則代表每一個社區的內部連接性十分密集，而社區間外部連接性十分稀疏。

以上論文中數學公式解析此指標可以看作分群完的結果與其期望產生之隨機社區群之內部連接性差異，而這個期望可以透過各種型產生，最常用的就是 Configuration model。指標值介於 $[-1,1]$ ，若值為 0 則代表其結果與隨機產生之社區相同，此次分群結果無意義，若是為正數則代表其結果比隨機產生之社區還要緊密，此次分群發現網絡存在特殊連結性，最後若是值為負數則代表其結果比隨機產生之社區還要稀疏，換句話來說，即本應該有關係的人卻沒有關係，有些人也會特別研究此領域。

Normalized Mutual Information (NMI)

此指標是用於將分群結果跟具有 Ground-Truth 之資料集之分群結果做比較，其值介於 $[0,1]$ ，值越大代表分群結果與 Ground-Truth 資料集的分群結果越相似，值為 0 則代表完全不一樣，反之值為 1 則代表完全相同。

3. Compare Algorithm

本篇論文與下列幾種演算法做比較，如 Fig.40，其中 Louvain(2008) [8] 我將會在後面再介紹，下方演算法 CNM(2006) [9] 為一個利用 eigenvector 方法對找出分群，Infomap [10] 則是利用 random walk 之方法，再來 NIBLPA(2014) [11]、G-CN(2019) [12] 、SD-GCN(2020) [13] 、LSMD(2020) [14] 、LCDR(2021) [15] 、PLPAC(2018) [16] 均為基於 LPA(2007)做出之優化，NIBLPA 是利用節點之 Influence 進行 LPA，G-CN 則是只對 boundary node 進行 LPA，LSMD 是一個從節點分支度小開始做 Label Diffusion 的辦法其過程鮮少有隨機之情況，而 LCDR 提出了一個新的測量節點重要程度之參數，最後 PLPAC 則是裡面唯一一個平行版本，透過開 threads 的方式，計算節點與欲更新標籤之間的 Confidence，Confidence 越高代表節點越屬於此標籤。

Comparing Algorithms :

- Louvain (2008)
- CNM (2006)
- LPA (2007)
- NIBLPA (2014)
- Infomap (2008)
- G-CN (2019)
- SD-GCN (2020)
- LSMD (2020)
- LCDR (2021)
- PLPAC (2018)

(Fig.40)

4. Simulation Algorithm

下方 Fig.41 為我將模擬之三個演算法，其中將 Louvain 分為兩個版本，其中的差異只再 random 之方式，指標結果上幾乎毫無差異，但發現執行時間差了非常多，其中版本 1 之執行時間更相似於本篇論文，即執行時間太慢了。而 LPA 我將最大的疊代次數設為 1000 次與 10000 次。

DPNLP (2022)	
Louvain (2008)	Random version 1
	Random version 2
LPA (2007)	Max iteration = 1000
	Max iteration = 10000

(Fig.41)

5. Louvain Algorithm

下方 Algorithm 4 為 Louvain 之 pseudo code :

Algorithm 4 *Louvain*

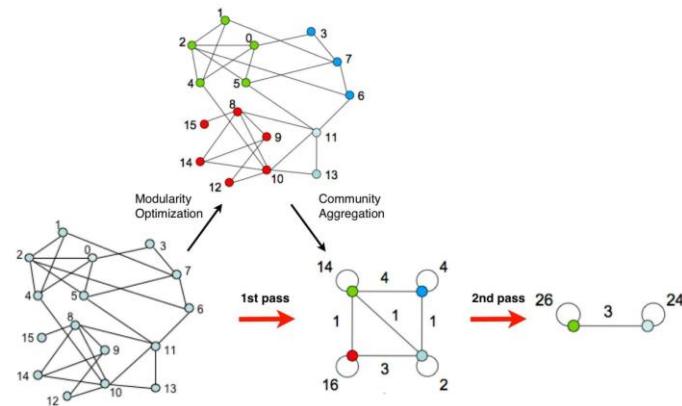
Input: $G(V, E)$: Network
Output: Label list L_u of each node

```
1: max_modularity  $\leftarrow 0$ ;
2: for each  $v \in V$  do
3:   initial each node  $v$  an unique label;
4: end for
5: max_modularity  $\leftarrow$  compute modularity;
6: repeat
7:   modularity  $\leftarrow 0$ ;
8:   repeat
9:     Flag  $\leftarrow 1$ 
10:    for each  $v \in V$  randomly selected do
11:       $\Delta_{max\_modularity} \leftarrow 0$ ;
12:      dominate_label  $\leftarrow label(v)$ ;
13:      for each  $u \in \Gamma(v)$  do
14:        compute  $\Delta_{modularity}$  as moving  $v$  into  $u$ 's community;
15:        if  $\Delta_{modularity} > \Delta_{max\_modularity}$  then
16:           $\Delta_{max\_modularity} \leftarrow \Delta_{modularity}$ ;
17:          dominate_label  $\leftarrow label(u)$ ;
18:        end if
19:      end for
20:      if dominate_label  $\neq label(v)$  then
21:        label(v)  $\leftarrow dominate\_label$ ;
22:        Flag  $\leftarrow 0$ ;
23:      end if
24:    end for
25:  until Flag = 1
26: rebuild the network, merge the nodes with same label into new node;
27: modularity  $\leftarrow$  compute modularity;
28: if modularity  $> max\_modularity$  then
29:   max_modularity  $\leftarrow modularity$ ;
30: end if
31: until max_modularity doesn't change compare with last iteration
```

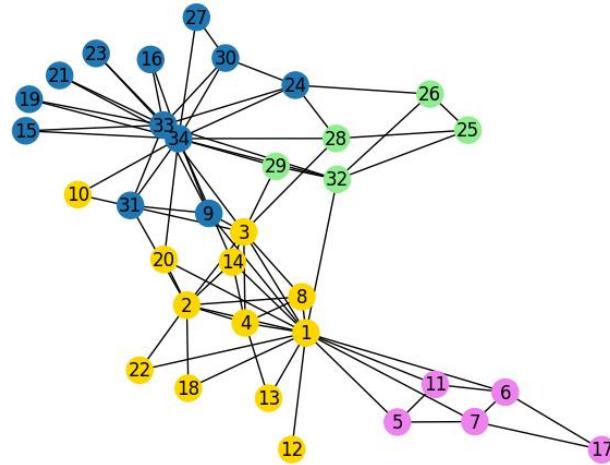
這是一個 2008 年提出的 Community Detection Algorithm，

其為一個最大化 Modularity 這一個指標的演算法，利用貪婪法則在每一步決策中選取使 Modularity 變化量上升最大且大於零的步驟，整個演算法分為兩個階段，第一階段(第 8-25 行)

將每一個節點考慮移至鄰居的群中，並且考慮 Modularity 變化量，選取有著最大且大於零的 Modularity 變化量的步驟移動節點，直到整張網絡中的節點均不會再移動到他人的群就結束第一階段，第二階段將同一群的節點壓縮成一個新的節點，形成新的網絡，之後再重新執行第一階段與第二階段直到 Modularity 達到最大。



(Fig.42)



(Fig.43)

上方 Fig.42 為 Louvain 之步驟圖，解釋階段 1 與階段 2 之工作，而 Fig.43 為前面提到之資料集，Zachary's Karate Dataset，做 Louvain 之結果。

6. Modularity (Q)

下方 Fig.44 為本篇論文對所有資料集跑所有比較演算法的 Modularity 結果，其中 CN 代表分群之個數， Q 代表 Modularity。可以看到以下方論文實驗結果可知，其 Modularity 最好的驗算法為 Louvain，這也是因為 Louvain 本身即是優化此指標的演算法。

Table 4 Modularity values of running algorithms on various networks. Q is the modularity value and CN is the number of discovered communities

Network	LVN		CNM		LPA		NBLPA		Infomap		SD-GCN		LSMD		LCDR		DPNLP	
	CN	Q	CN	Q	CN	Q	CN	Q	CN	Q	CN	Q	CN	Q	CN	Q	CN	Q
Karate	4	0.41	3	0.38	3	0.38	5	0.40	3	0.40	2	0.371	2	0.371	2	0.371	2	0.371
Dolphins	6	0.51	4	0.49	5	0.46	5	0.43	5	0.53	4	0.526	2	0.378	2	0.378	2	0.378
Football	10	0.60	6	0.54	9	0.54	8	0.50	12	0.58	12	0.60	12	0.58	14	0.602	11	0.584
Polbooks	5	0.52	4	0.50	4	0.50	4	0.55	5	0.52	3	0.483	3	0.44	5	0.502	3	0.441
Amazon	286	0.90	6086	0.87	27959	0.64	44036	0.67	17319	0.75	17966	0.67	34304	0.68	21749	0.778	19101	0.69
DBLP	225	0.81	3301	0.72	30384	0.60	30986	0.61	16921	0.71	6599	0.66	17280	0.65	19405	0.693	13219	0.63
Email_Enron	1463	0.55	1605	0.51	2112	0.38	2131	0.12	2605	0.52	1014	0.47	1221	0.39	1612	0.56	1651	0.36
NetScience	276	0.92	276	0.90	336	0.91	366	0.87	294	0.90	185	0.56	275	0.95	332	0.92	23	0.92
Power_Grid	57	0.84	41	0.89	774	0.74	1356	0.61	496	0.79	397	0.58	446	0.79	473	0.79	715	0.36
Condmat_2003	98	0.72	1916	0.66	3814	0.53	3725	0.50	2540	0.61	1985	0.68	3502	0.57	3122	0.681	2180	0.66
Condmat_2005	1077	0.71	2253	0.63	4503	0.50	3523	0.23	2966	0.63	2290	0.59	3952	0.44	3691	0.645	2405	0.62
Email	15	0.54	17	0.48	77	0.34	149	0.40	60	0.52	104	0.35	142	0.42	16	0.543	4	0.39

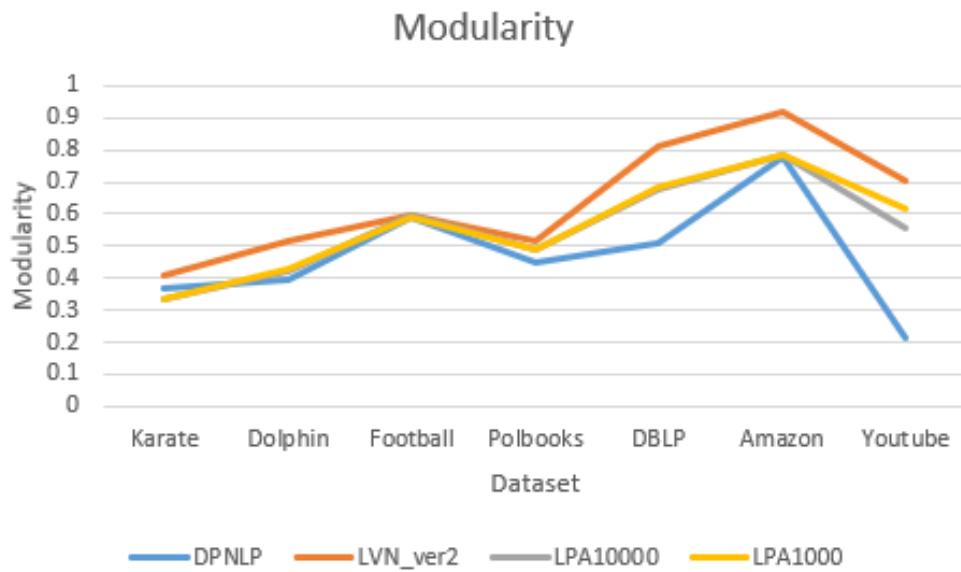
(Fig.44)

下方 Fig.45 為我模擬之結果，其中 Karate、Dolphin、Football、Polbooks 為我測試 50 次的結果，而 DBPL、Amazon、Youtube 為我測試 20 次的結果，之後的所有模擬也均相通。

Dataset \ Algo	DPNLP		Lvn_ran_ver2		LPA (10000)		LPA (1000)	
	CN	Q	CN	Q	CN	Q	CN	Q
Karate	2	0.3665	4	0.4111	2	0.3335	2	0.3334
Dolphin	2	0.3949	5	0.5189	6	0.4228	6	0.4296
Football	9	0.5892	9	0.5980	11	0.5948	11	0.5929
Polbooks	3	0.4515	4	0.5164	3	0.4918	3	0.489
DBLP	6766	0.3814	117	0.8092	21412	0.6762	22379	0.6882
Amazon	15306	0.7766	131	0.9175	22310	0.7859	22271	0.7864
Youtube	6335	0.2144	4109	0.7087	36673	0.5177	38351	0.6333

(Fig.45)

可以看到其中跑得最好的也是 Louvain 演算法，其它資料集除了 DBLP 與 YouTube 以外，均跑得跟 LPA 差不多，並且我發現尤其是 YouTube 這個資料集，本篇論文提出的演算法跑出來的實驗參數與其它比較演算法相比極差，這也可能是這篇論文作者沒放上去的原因，下方 Fig.46 為 Fig.45 中 Modularity 化成折線圖，可以看到 DPNLP 在 DBLP 與 YouTube 兩個資料集中與其它演算法相比有較大的落差，尤其是 YouTube。



(Fig.46)

7. Normalized Mutual Information (*NMI*)

下方 Fig.47 為本篇論文對所有資料集，跑所有比較演算法的 *NMI* 結果。可以看到，DBNLP 相比其它比較演算法有著更高的 *NMI*。

Network	LVN	CNM	LPA	NIBLPA	Infomap	G-CN	SD-GCN	LSMD	LCDR	DPNLP	PLPAC
Karate	0.59	0.69	0.68	0.58	0.70	0.83	1	1	1	1	0.98
Dolphins	0.52	0.55	0.53	0.50	0.54	0.54	0.70	1	1	1	0.92
Football	0.89	0.75	0.8	0.72	0.92	0.87	0.93	0.93	0.90	0.89	0.90
Polbooks	0.45	0.53	0.52	0.53	0.50	0.53	0.58	0.59	0.58	0.70	N/A
Amazon	0.11	0.11	0.53	0.60	0.72	0.59	0.70	0.72	0.69	0.693	N/A
DBLP	0.13	0.16	0.49	0.46	0.41	0.51	0.54	0.50	0.70	0.56	N/A
Youtube	0.06	N/A	0.07	N/A	0.02	0.07	0.10	0.19	0.31	0.17	N/A
LiveJournal	0.03	N/A	N/A	N/A	0.03	N/A	N/A	0.80	N/A	0.38	N/A
Orkut	0.06	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0.17	N/A

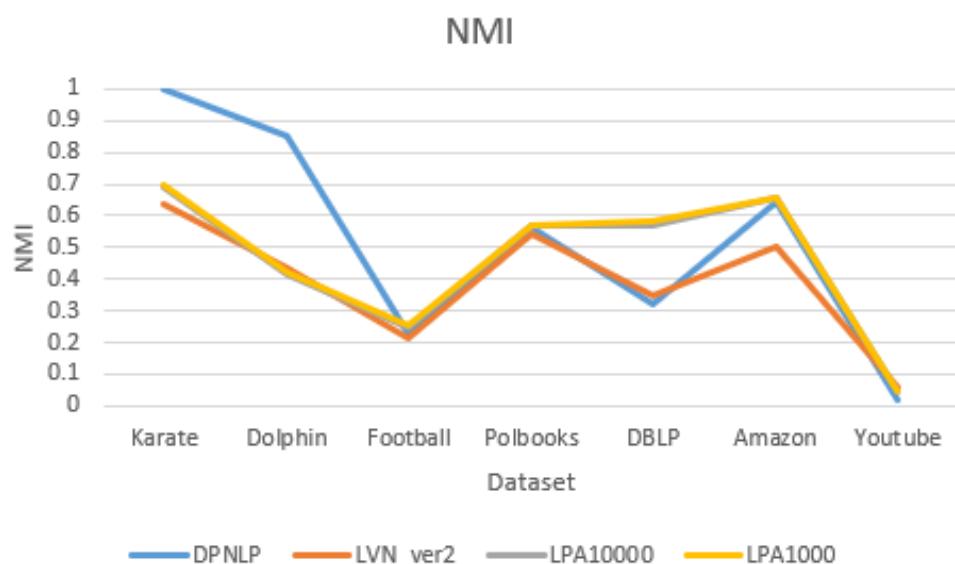
(Fig.47)

下方 Fig.48 為我模擬之結果，而 Fig.49 為以 Fig.48 畫出的折線圖

線圖

Dataset \ Algo	DPNLP	Lvn_ran_ver2	LPA (10000)	LPA (1000)
Karate	1	0.6374	0.6892	0.6994
Dolphin	0.8539	0.4353	0.4155	0.4238
Football	0.2282	0.2145	0.2502	0.2508
Polbooks	0.5659	0.5404	0.5670	0.5723
DBLP	0.323	0.3469	0.5715	0.5829
Amazon	0.6453	0.5084	0.6547	0.6547
Youtube	0.0207	0.0575	0.0456	0.0476

(Fig.48)



(Fig.49)

從 Fig.49 中可以發現，DPNLP 在小資料集的狀況下均跑得比其它比較演算法更貼近 Ground Truth，但一到大資料集跑多次取平均後就沒有此優勢了，雖然在跑的過程中有出現指標十分貼近論文實驗結過的時候，但大部分均沒有，所以平均下來反而還會小輸其它比較演算。

8. Results stability

以下為本篇論文對所有資料集，跑所有比較演算法的 Result Stability 結果，其中 min 與 max 為測試多次結果中最小與最大值，而 Δ 則是由 max 與 min 相減而得，其值越小代表結果越相近，反之越大則代表結果越不一樣。

以下 Fig.50 為本篇論文之結果

Table 7 Comparison of the stability of different algorithms with ten times execution on networks

Dataset	LVN			LPA			NIBLPA			Infomap			G-CN			LSMD			DPNLP			
	min	max	Δ	min	max	Δ	min	max	Δ	min	max	Δ	min	max	Δ	min	max	Δ	min	max	Δ	
karate	NMI	0.58	0.61	0.03	0.22	0.79	0.57	0.27	0.58	0.31	0.69	0.7	0.01	0.83	0.83	0	1	1	0	1	1	0
	Q	0.41	0.43	0.02	0.13	0.4	0.27	0.11	0.4	0.29	0.4	0.41	0.01	0.37	0.37	0	0.37	0.37	0	0.37	0.37	0
	CN	4	5	1	1	3	2	3	5	2	3	4	1	2	2	0	2	2	0	2	2	0
Dolphins	NMI	0.48	0.53	0.05	0.48	0.55	0.07	0.48	0.50	0.02	0.53	0.54	0.01	0.51	0.54	0.03	1	1	0	1	1	0
	Q	0.45	0.5	0.05	0.49	0.51	0.02	0.43	0.44	0.01	0.52	0.527	0.007	0.51	0.52	0.01	0.37	0.37	0	0.37	0.37	0
	CN	5	6	1	2	4	2	5	5	0	5	6	1	4	5	1	2	2	0	2	2	0
Football	NMI	0.86	0.89	0.03	0.79	0.81	0.02	0.70	0.72	0.02	0.9	0.91	0.01	0.85	0.87	0.02	0.93	0.93	0	0.85	0.91	0.06
	Q	0.55	0.59	0.04	0.53	0.55	0.02	0.50	0.51	0.01	0.58	0.6	0.02	0.58	0.58	0	0.58	0.58	0	0.55	0.59	0.04
	CN	9	10	1	6	13	7	8	8	0	12	12	0	12	13	1	12	12	0	10	12	2
Polbooks	NMI	0.45	0.47	0.02	0.5	0.53	0.03	0.53	0.53	0	0.5	0.51	0.01	0.51	0.53	0.02	0.59	0.59	0	0.69	0.69	0
	Q	0.49	0.51	0.02	0.48	0.49	0.01	0.55	0.55	0	0.522	0.525	0.003	0.51	0.52	0.01	0.44	0.44	0	0.44	0.44	0
	CN	5	5	0	1	3	2	4	4	0	5	6	1	4	6	2	3	3	0	3	3	0
Email	Q	0.52	0.55	0.03	0.33	0.38	0.05	0.10	0.12	0.02	0.52	0.53	0.01	0.09	0.11	0.02	0.39	0.39	0	0.33	0.33	0
	CN	1451	1483	32	2025	2161	136	2174	2459	285	2608	2617	9	1753	1892	139	1221	1221	0	1625	1633	8
	NetScience	Q	0.91	0.92	0.01	0.89	0.91	0.02	0.87	0.88	0.01	0.9	0.91	0.01	0.87	0.89	0.02	0.94	0.94	0	0.864	0.873
Condmat2003	CN	271	278	7	313	323	10	8	8	0	277	294	17	312	328	16	275	275	0	340	344	4
	Q	0.71	0.73	0.02	0.51	0.54	0.03	0.40	0.51	0.02	0.61	0.63	0.02	0.59	0.61	0.02	0.57	0.57	0	0.67	0.67	0
	CN	91	116	25	2362	2445	83	3586	3675	89	2540	3222	682	3399	3413	14	3502	3502	0	1806	1815	9
Condmat2005	Q	0.71	0.73	0.02	0.5	0.54	0.04	0.2	0.23	0.03	0.6	0.63	0.03	0.6	0.63	0.03	0.44	0.44	0	0.63	0.63	0
	CN	1056	1108	52	2448	2721	273	2169	2368	199	2957	3266	309	3341	3827	486	3952	3952	0	2049	2055	6

(Fig.50)

以下為我模擬之結果

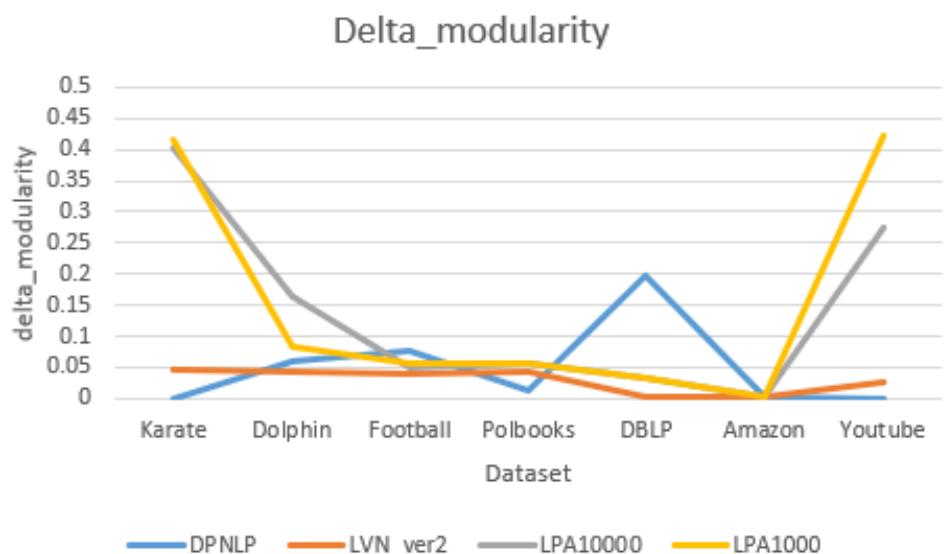
$\Delta Modularity$:

下方 Fig.51 與 Fig.52 為我跑演算法之 *Modularity* 的變化量，

可以看到其中 DPNLP 有較 LPA 更加穩定，其中 DPNLP 在 DBLP 資料集中會有大於 LPA 之變化量原因可能為我測試次數不夠多，尚未測到其最差情況，Amazon 與 YouTube 均相同。

Algo Dataset	DPNLP	Lvn_ran_ver2	LPA (10000)	LPA (1000)
Karate	0	0.0464	0.402	0.4151
Dolphin	0.0588	0.0446	0.1647	0.0818
Football	0.0751	0.03911	0.0499	0.0572
Polbooks	0.0111	0.0423	0.0546	0.055
DBLP	0.1982	0.0022	0.0324	0.0316
Amazon	0.003	0.0015	0.0017	0.0041
Youtube	0.0007	0.0255	0.2751	0.4215

(Fig.51)



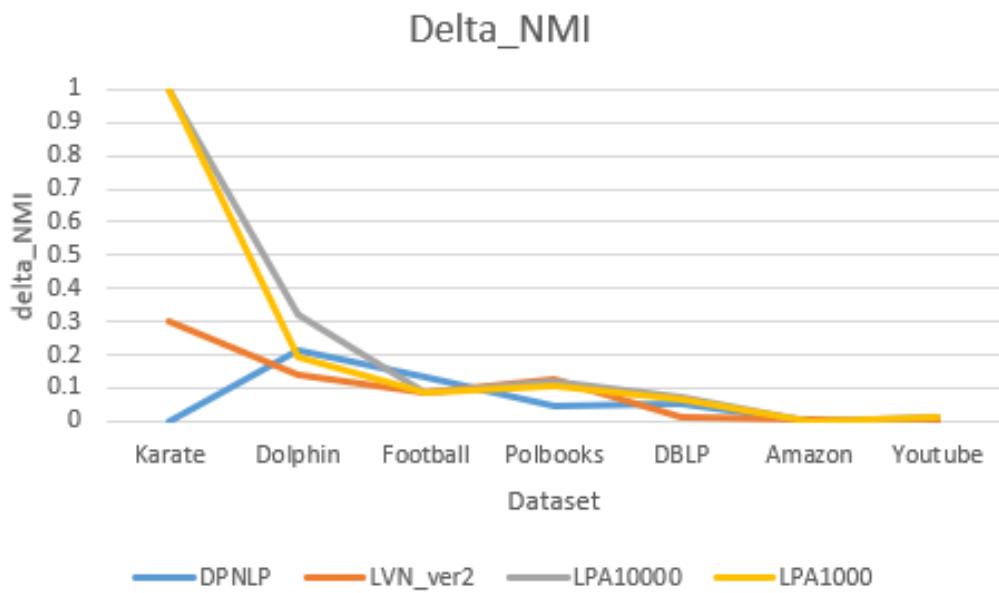
(Fig.52)

ΔNMI :

下方 Fig.51 與 Fig.52 為我跑演算法之 NMI 的變化量，大部分情況 DPNLP 相較於 LPA 較更穩定，除了 Football 資料集，於是我有去多測試 100 次這個資料集跑，我發現 LPA 的 ΔNMI 上升到 0.1334，而 DPNLP 維持不變，所我認為是我跑的實驗次數不夠多的原因。

Algo Dataset \	DPNLP	Lvn_ran_ver2	LPA (10000)	LPA (1000)
Karate	0	0.3029	1	1
Dolphin	0.2126	0.1366	0.324	0.1956
Football	0.135	0.0865	0.0869	0.0849
Polbooks	0.0452	0.1288	0.1165	0.1091
DBLP	0.0557	0.01	0.0718	0.0632
Amazon	0.0014	0.0084	0.0005	0.0005
Youtube	0.0018	0.0014	0.0098	0.013

(Fig.53)



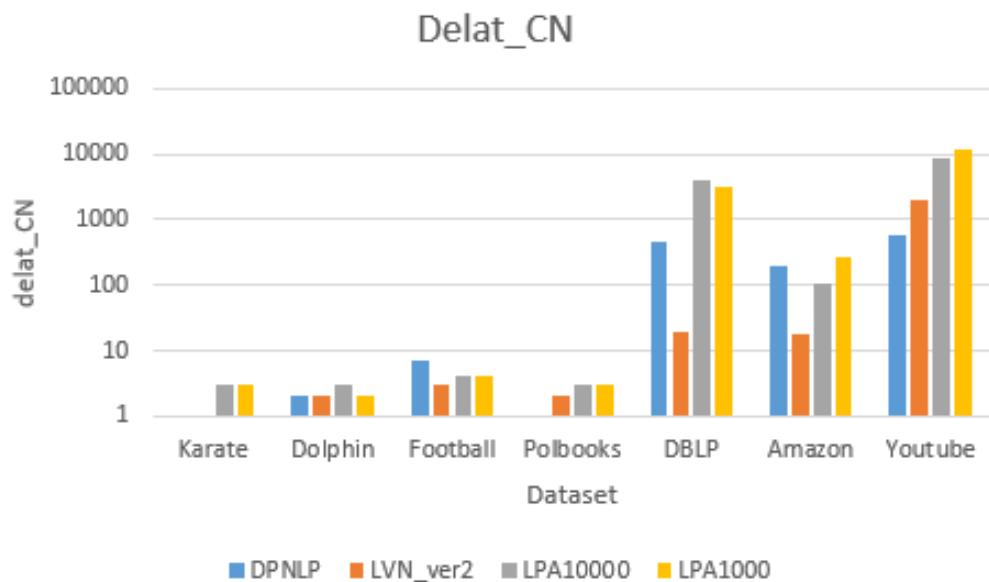
(Fig.54)

ΔCN : (分群結果個數之變化量)

下方 Fig.51 與 Fig.52 為我跑演算法之結果分群個數的變化量，其中 LPA 差異最大，其中大資料集僅僅做了 20 次就測到分群個數差了 11898 群，而 DPNLP 在 Football 跑的結果分群變化量極大，我認為有可能此資料集較難判斷 Core node。

Dataset \ Algo	DPNLP	Lvn_ran_ver2	LPA (10000)	LPA (1000)
Karate	0	1	3	3
Dolphin	2	2	3	2
Football	7	3	4	4
Polbooks	0	2	3	3
DBLP	474	19	3942	3273
Amazon	198	18	103	262
Youtube	564	2030	8576	11898

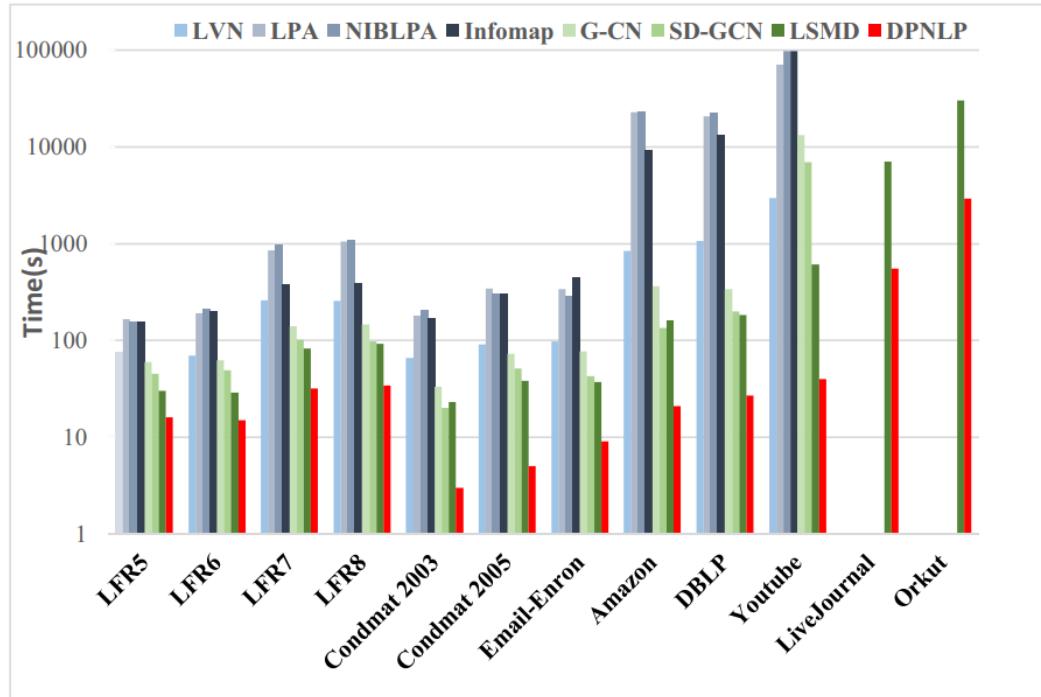
(Fig.55)



(Fig.56)

9. Execution time

以下 Fig.57 為本篇論文之結果，可以看到其論文方法跑得相當快，而其它演算法均跑得十分慢。

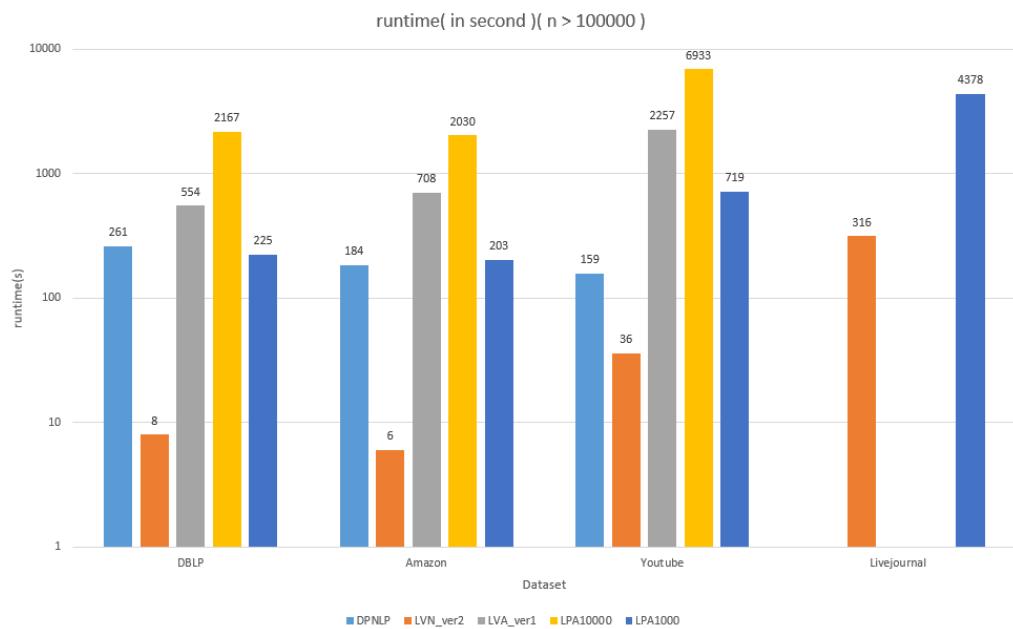


(Fig.57)

以下 Fig.58 與 Fig.59 為我模擬之結果

Algo Dataset \	DPNLP	Lvn_ran_ver2	Lvn_ran_ver1	LPA (10000)	LPA (1000)
Karate	0.000704	0.001684	0.001855	0.001026	0.0010
Dolphin	0.001749	0.002364	0.002806	0.002684	0.002935
Football	0.007118	0.002925	0.003756	0.002515	0.002439
Polbooks	0.002292	0.003143	0.003236	0.002669	0.002703
DBLP	261.3211	8.0564	554.4795	2167.053	225.7736
Amazon	184.3456	6.3486	708.148	2030.938	203.8571
YouTube	159.7537	36.852	2257.96	6933.823	719.041
LiveJournal	33929.436	316.8617	-	-	4378.32

(Fig.58)



(Fig.59)

可以看到最快的為 Louvain ver2，其次是本篇論文提出之方法，DPNLP，但到了 LiveJournal 資料集其 DPNLP 就無法在一萬秒內結束，我猜是因為當網絡一大起來 DPNLP 需要花很多時間收斂，不像 LPA 可以設定最大疊代次數強迫它結束，如果要 DPNLP 想非常快收斂的話，可以將判斷 Peripheral node 的條件放寬，這樣會變得非常快。

V. Conclusion

這篇論文提出之演算法中在小資料集表現十分好，但在大資料集的結果就沒有如論文說的好，其中的優點為利用 **Peripheral node** 之概念減少進行 LPA 的節點數量，以及先移除分支度為 1 或 2 之節點，將其留在最後處理，缺點為模擬出來得時間沒有比 Louvain 快，Louvain 真的非常快，並且我在測試 Louvain 時發現，Louvain 並沒有像這篇論文中的實驗數據慘，其原因就有待探討。

VI. Execution sample

執行 DPNLP 跑 YouTube 資料集，如下 Fig.60 所示

```
m11102155@RTK3060-2:~/ALGO_final_project$ ./DPNLP_version5 dataset/youtube_dataset.txt DPNLP/dataset_log/youtube_dataset_log.txt dataset_gt/youtube_GT.txt DPNLP/exp_record_ver5/youtube_exp_record.txt
DATASET FILE OPEN SUCCESS !!!
=====
DATASET INFORMATIONS
-----
NAME      = dataset/youtube_dataset.txt
BASIC INFO =  sym unweighted
#EDGE     = 2987624
#NODE     = 1134899
AvgDegree = 5.26585
=====
/////////.... START DOING DPNLP ALGO ...../////////
//////.... EXECUTION TIME .... /////
READ BUILD GRAPH TIME TAKES :: 0.451827 sec
ALGO DPNLP TAKES :: 160.568300 sec
WRITTING THE RESULT INTO LOG FILE .
LOG FILE OPEN SUCCESS !!!
LOG FILE OPEN SUCCESS !!!
FINISH WRITTING !!!
/////////.... EXPERIMENT MEASUREMENTS ..... ///////////
TOTAL #COMMUNITY = 5966
MODULARITY = 0.214142
METUAL INFORMATION :: 0.012125
NMI = 0.021023
/////////.... WRITTING EXPERIMENT RESULTS ..... ///////////
WRITTING THE EXP RESULT INTO EXP_RECORD FILE .
WRITTING :
READ BUILD GRAPH TIME = 0.451827
ALGO TIME             = 160.568300
NUM COMMUNITY         = 5966
MODULARITY            = 0.214142
NMI                  = 0.021023
=====
```

(Fig.60)

執行 Louvain version2 跑 Amazon 資料集，如下 Fig.61 所示

```
m11102155@RTK3060-2:~/ALGO_final_project$ ./random_louvain_version2 dataset/amazon_dataset LOUVAIN/dataset_log/amazon_dataset_log.txt dataset_gt/amazon_GT.txt LOUVAIN/exp_record_rnd_version2/amazon_dataset_exp_record.txt
FILE OPEN SUCCESS !!!
=====
DATASET INFORMATIONS
-----
NAME      = dataset/amazon_dataset
BASIC INFO =  sym unweighted
#EDGE     = 925872
#NODE     = 334863
AvgDegree = 5.52986
=====
/////////.... START DOING LOUVAIN ALGO ...../////////
//////.... EXECUTION TIME .... /////
READ BUILD GRAPH TIME TAKES :: 0.000000 sec
ALGO LOUVAIN TAKES :: 6.468777 sec
WRITTING THE RESULT INTO LOG FILE .
LOG FILE OPEN SUCCESS !!!
LOG FILE OPEN SUCCESS !!!
FINISH WRITTING !!!
/////////.... EXPERIMENT MEASUREMENTS ..... ///////////
TOTAL #COMMUNITY = 122
Reading GT dataset ...
Partitioning dataset ...
Building algo_partition and gt_partition ...
Computing NMI ...
METUAL INFORMATION :: 2.664488
/////////.... WRITTING EXPERIMENT RESULTS ..... ///////////
WRITTING THE EXP RESULT INTO EXP_RECORD FILE .
WRITTING :
READ BUILD GRAPH TIME = 0.000000sec
ALGO TIME             = 6.468777sec
NUM COMMUNITY         = 122
MODULARITY            = 0.917885
NMI                  = 0.505871
```

(Fig.61)

執行 Louvain version1 跑 Dolphin 資料集，如下 Fig.62 所示

```
m11102155@RTX3060-2:~/ALGO_final_project$ ./random_louvain_ver1 dataset/dolphins_dataset.txt LOUVAIN/dataset_log/dolphins_dataset_log.txt dataset_gt/dolphins_GT.txt LOUVAIN
N/exp_record_random_version1/dolphins_dataset_exp_record.txt
FILE OPEN SUCCESS !!!
=====
DATASET INFORMATIONS
-----
NAME      = dataset/dolphins_dataset.txt
BASIC INFO = sym unweighted
#EDGE     = 159
#NODE     = 62
AvgDegree = 5.12903
=====
///////// ..... START DOING LOUVAIN ALGO ..... //////
// ..... EXECUTION TIME ..... //
READ BUILD GRAPH TIME TAKES :: 0.000000 sec
ALGO LOUVAIN TAKES :: 0.002286 sec
WRITTING THE RESULT INTO LOG FILE .
LOG FILE OPEN SUCCESS !!!
FINISH WRITTING !!!
///////// ..... EXPERIMENT MEASUREMENTS ..... //////
TOTAL #COMMUNITY = 5
Reading GT dataset ...
Partitioning dataset ...
Building algo_partition and gt_partition ...
Computing NMI ...
METUAL INFORMATION :: 0.420240
///////// ..... WRITTING EXPERIMENT RESULTS ..... //////
WRITTING THE EXP RESULT INTO EXP_RECORD FILE .
WRITTING ::

READ BUILD GRAPH TIME = 0.000000sec
ALGO TIME           = 0.002280sec
NUM COMMUNITY       = 5
MODULARITY          = 0.508643
NMI                 = 0.380212
```

(Fig.62)

執行 LPA 10000 跑 Karate 資料集，如下 Fig.63 所示

```
m11102155@RTX3060-2:~/ALGO_final_project$ ./lpa_10000 dataset/karate_dataset.txt LPA/dataset_log/karate_dataset_log.txt dataset_gt/karate_GT.txt LPA/exp_record_random_iter
_10000/karate_dataset_exp_record.txt
FILE OPEN SUCCESS !!!
=====
DATASET INFORMATIONS
-----
NAME      = dataset/karate_dataset.txt
BASIC INFO = sym unweighted
#EDGE     = 78
#NODE     = 34
AvgDegree = 4.58824
=====
///////// ..... START DOING LPA ALGO ..... //////
// ..... EXECUTION TIME ..... //
READ BUILD GRAPH TIME TAKES :: 0.000312 sec
ALGO DPNL takes :: 0.000832 sec
WRITTING THE RESULT INTO LOG FILE .
FILE OPEN SUCCESS !!!
FINISH WRITTING !!!
///////// ..... EXPERIMENT MEASUREMENTS ..... //////
TOTAL #COMMUNITY = 3
MODULARITY = 0.399888
METUAL INFORMATION :: 0.691416
///////// ..... WRITTING EXPERIMENT RESULTS ..... //////
WRITTING THE EXP RESULT INTO EXP_RECORD FILE .
WRITTING ::

READ BUILD GRAPH TIME = 0.000312sec
ALGO TIME           = 0.000832sec
NUM COMMUNITY       = 3
MODULARITY          = 0.399888
NMI                 = 0.825518
```

(Fig.63)

執行 LPA 1000 跑 Polbooks 資料集，如下 Fig.64 所示

```
millie2155@RTX3060-2:~/ALGO_final_project$ ./lpa_random1000 dataset/polbooks_dataset.txt LPA/dataset_log/polbooks_dataset_kog.txt dataset_gt/polbooks_GT.txt LPA/exp_record_
random_iter_1000/polbooks_dataset_exp_record.txt
FILE OPEN SUCCESS !!
=====
DATASET INFORMATIONS
-----
NAME      = dataset/polbooks_dataset.txt
BASIC INFO = sym unweighted
#EDGE     = 441
#NODE     = 105
AvgDegree = 8.4
=====
/////////..... START DOING LPA ALGO .....,///
//// ... EXECUTION TIME .... /////
READ BUILD GRAPH TIME TAKES :: 0.000688 sec
ALGO DPNLp TAKES :: 0.001896 sec
WRITTING THE RESULT INTO LOG FILE .
FILE OPEN SUCCESS !!!
FINISH WRITTING !!!
/////////.... EXPERIMENT MEASUREMENTS ..... /////
TOTAL #COMMUNITY = 3
MODULARITY = 0.451923
METUAL INFORMATION :: 0.463941
/////////.... WRITTING EXPERIMENT RESULTS ..... /////
WRITTING THE EXP RESULT INTO EXP_RECORD FILE .
WRITTING ::

  READ BUILD GRAPH TIME = 0.000688sec
  ALGO TIME           = 0.001896sec
  NUM COMMUNITY       = 3
  MODULARITY          = 0.451923
  NMI                 = 0.525951
```

(Fig.64)

VII. Reference

- [1] H. Qureshi, Amir; Chaoji, Vineet; Maiguel, Dony; Hafeez Faridi, Mohd; J. Barth, Constantinos; M. Salem, Saeed; et al. (2015): Platelet Protein-Protein Interaction (PPI) network.. PLOS ONE. Figure. <https://doi.org/10.1371/journal.pone.0007627.g010>
- [2] PhysRevE.76.036106,Near linear time algorithm to detect community structures in large-scale networks Raghavan, Usha Nandini and Albert, R\'eka and Kumara, Soundar Phys. Rev. E, 2007,Sep,AmericanPhysicalSociety,10.1103/PhysRevE.76.036106,https://link.aps.org/doi/10.1103/PhysRevE.76.036106\
- [3] W. W. Zachary, Aninformation flow model for conflict and fission in small groups, Journal ofAnthropological Research 33, 452-473 (1977).
- [4] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson, *Behavioral Ecology and Sociobiology* **54**, 396-405 (2003).
- [5] M. Girvan and M. E. J. Newman, *Proc. Natl. Acad. Sci. USA* **99**, 7821-7826 (2002).
- [6] L. A. Adamic and N. Glance, "The political blogosphere and the 2004 US Election", in Proceedings of the WWW-2005 Workshop on the Weblogging Ecosystem (2005).
- [7] SNAP: A General-Purpose Network Analysis and Graph-Mining Library , leskovec2016snap , Marinka Zitnik, Rok Sosi, Sagar Maheshwari, and Jure Leskovec . ACM Transactions on Intelligent Systems and Technology (TIST), 2016 , ACM

- [8] Blondel ,Vincent D , Guillaume ,Jean-Loup . Fast unfolding of communities in large networks 2008 . Journal of Statistical Mechanics : Theory and Experiment P10008.
- [9] Newman, M.E.: Finding community structure in networks using the eigenvectors of matrices. Phys. Rev. E 74(3), 036104 (2006)
- [10] Rosvall, M., Bergstrom, C.: Maps of random walks on complex networks reveal community structure. Proc. Natl. Acad. Sci. U. S. A. 105, 1118–23 (2008)
- [11] Xing, Y., Meng, F., Zhou, Y., Zhu, M., Shi, M., Sun, G.: A node influence based label propagation algorithm for community detection in networks. Sci. World J. 2014, 627581 (2014)
- [12] Tasgin, M., Bingol, H.O.: Community detection using boundary nodes in complex networks. Physica A 513, 315–324 (2019)
- [13] Zarezadeh, M., Nourani, E., Bouyer, A.: Community detection using a new node scoring and synchro nous label updating of boundary nodes in social networks. JAIDM. 8(2), 201–212 (2020)
- [14] Bouyer, A., Roghani, H.: LSMD: A fast and robust local community detection starting from low degree nodes in social networks. Futur. Gener. Comput. Syst. 113, 41–57 (2020)
- [15] Aghaalizadeh, S., Afshord, S.T., Bouyer, A., Anari, B.: A three-stage algorithm for local community detection based on the high node importance ranking in social networks. Physica A: Stat Mech. Appl. 563, 125420 (2021)
- [16] Chen, N., Liu, Y., Cheng, J., Liu, Q.: A novel parallel community detection scheme based on label propagation. World Wide Web 21(5), 1377–1398 (2018)

- [17] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson, *Behavioral Ecology and Sociobiology* **54**, 396-405 (2003).
- [18] L. A. Adamic and N. Glance, "The political blogosphere and the 2004 US Election", in Proceedings of the WWW-2005 Workshop on the Weblogging Ecosystem (2005).
- [19] SNAP: A General-Purpose Network Analysis and Graph-Mining Library , leskovec2016snap , Marinka Zitnik, Rok Sosi, Sagar Maheshwari, and Jure Leskovec . ACM Transactions on Intelligent Systems and Technology (TIST), 2016 , ACM