An improvement on the Louvain algorithm using random walks

DO Duy Hieu and PHAN Thi Ha Duong

Institute of Mathematics
Vietnam Academy of Science and Technology

Email: ddhieu@math.ac.vn (Do Duy Hieu), phanhaduong@math.ac.vn (Phan Thi Ha Duong).

Abstract

We will present improvements to famous algorithms for community detection, namely Newman's spectral method algorithm and the Louvain algorithm. The Newman algorithm begins by treating the original graph as a single cluster, then repeats the process to split each cluster into two, based on the signs of the eigenvector corresponding to the second-largest eigenvalue. Our improvement involves replacing the time-consuming computation of eigenvalues with a random walk during the splitting process.

The Louvain algorithm iteratively performs the following steps until no increase in modularity can be achieved anymore: each step consists of two phases—phase 1 for partitioning the graph into clusters, and phase 2 for constructing a new graph where each vertex represents one cluster obtained from phase 1. We propose an improvement to this algorithm by adding our random walk algorithm as an additional phase for refining clusters obtained from phase 1. It maintains a complexity comparable to the Louvain algorithm while exhibiting superior efficiency. To validate the robustness and effectiveness of our proposed algorithms, we conducted experiments using randomly generated graphs and real-world data.

1 Introduction

Research on community detection in networks is an essential field within network science, with a wide array of applications in computer science and various other scientific disciplines [7, 11, 22]. Consequently, numerous research efforts from scientists have employed various methodological approaches. Among these, two algorithms garnering significant attention are Newman's spectral method [16] and the Louvain algorithm [2]. Therefore, numerous extensions and improvements have been made to these algorithms [4, 25, 27, 28].

Let $G = (V, E)$ be an undirected and connected graph defined on the vertex set $V = \{1, 2, \ldots, n\}$, and define A the adjacency matrix of G, which means $A_{ij} = 1$ if vertices i and j are connected (linked by an edge), and $A_{ij} = 0$ otherwise. Let m be the number of edges of G. The degree $d_i = \sum_j A_{ij}$ of a vertex i represents the number of its neighbors, including itself. Finally, we denote D as the diagonal matrix of degrees ($D_{ii} = d_i$ and $D_{ij} = 0$ for $i \neq j$).

Random walk is a focal point of interest in network community research, as it helps elucidate the characteristics of vertices belonging to the same or different communities [9, 21]. This paper will employ random walks to improve the Newman's spectral method and the Louvain algorithm.

1.1 Modularity

Modularity is commonly used to evaluate the quality of clustering. Modularity has various definitions, but the most widely recognized definition is presented in [15, 20]. Specifically, with $\mathcal{P} = \{C_1, C_2, ..., C_k\}$ being a clustering of graph G, denote the cluster label to which vertex i belongs as $C(i)$. Then, the modularity Q is defined as follows:

$$Q(\mathcal{P}, G) = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{d_i d_j}{2m} \right] \delta_{C(i)C(j)}, \tag{1.1}$$

where $\delta_{C(i)C(j)}$ is the Kronecker delta and $A_{ij} - \frac{d_i d_j}{2m}$ is the difference between the actual number of edges between vertices i and j, and the expected number of edges. Modularity Q can be rewritten as:

$$Q(\mathcal{P}, G) = \frac{1}{2m} \sum_{C \in \mathcal{P}} \sum_{i,j \in C} \left[A_{ij} - \frac{d_i d_j}{2m} \right]. \tag{1.2}$$

From this, we can interpret the modularity corresponding to each clustering as the total modularity of each community. For a community C, its corresponding modularity is:

$$Q_C(G) = \sum_{i,j \in C} \left[A_{ij} - \frac{d_i d_j}{2m} \right]. \tag{1.3}$$

1.2 Random walk on graphs

A random walk on a graph is a process that starts at a given vertex and moves to another vertex at each step. The next vertex in the walk is chosen uniformly and randomly from among the neighbors of the current vertex. Consequently, at each step, the transition probability from vertex i to vertex j is given by $P_{ij} = \frac{A_{ij}}{d_i}$. This definition establishes the transition matrix P for the random walk process. It's evident that $P = D^{-1}A$. Moreover, P^t_{ij} denotes the probability of transitioning from vertex i to vertex j after t steps. The transition matrix P satisfies $\lim_{t \to \infty} P^t = P_\infty$, where $(P_\infty)_{ij} = \phi_j$, the j-th component of the unique stationary distribution $\phi = (\phi_1, \phi_2, ..., \phi_n)$, note that $\phi_i = d_i / \sum d_j$ (see in [21]). Finally, the information about vertex i encoded in P^t resides in the n probabilities $\left(P^t_{ik}\right) 1 \leq k \leq n$, equivalent to the i-th row of matrix P^t denoted by $P^t_{i\bullet}$.

1.3 Newman's Spectral Method

Firstly, we recall the definition of the normalized Laplacian matrix $\mathbf{L} = D^{-1/2} A D^{-1/2}$, which has been widely used. This normalized Laplacian matrix $\mathbf{L}$ has one eigenvalue equal to 1, and the remaining eigenvalues have absolute values less than 1.

Newman's spectral method [16] utilizes the second eigenvector of the normalized Laplacian matrix $\mathbf{L}$ to partition the vertices of graph G into two communities C_1 and C_2. Subsequently, the algorithm iterates with two subgraphs induced by C_1 and C_2. This process repeats until the modularity no longer increases.

This approach is appropriate because it leverages the extensively studied Laplacian matrix. However, calculating the second eigenvector entails considerable computational complexity. Hence, instead of directly analyzing the Laplacian matrix, we employ random walks to mitigate the computational complexity of our algorithm. We refer to our algorithm as the Random Walk

Graph Partition Algorithm. Additionally, we establish a connection with the eigenvalues and eigenvectors of the normalized Laplacian matrix. This connection elucidates that our algorithm produces clustering outcomes akin to Newman's algorithm when the number of random walk steps is adequately large.

1.4 Louvain Algorithm

The Louvain algorithm, as introduced in [2], is renowned for its simplicity and effectiveness. It operates in two main phases:
Phase 1: Each vertex will belong to its own community. Then, each vertex will decide whether to go to its neighboring community based on whether it increases modularity.
Phase 2: Create a new graph where each vertex corresponds to a community. Each vertex will have a loop with a weight equal to the number of edges between vertices within the corresponding community, and there will be an edge between two vertices with a weight equal to the number of edges between the two corresponding communities.

The algorithm iterates these phases until Modularity ceases to increase. The Louvain algorithm is famous not only for its fast calculation speed but also for its high algorithm accuracy. However, this algorithm could be less effective when the network has an unclear community structure. In this paper, we will add a phase of fine-tuning local communities after the first phase of the Louvain algorithm using the Random Walk Graph Partition Algorithm. Therefore, we obtained a new algorithm that is more efficient than the Louvain algorithm and has equivalent computational complexity.

1.5 Our contribution

In this paper, we present our algorithms as follows:

In Section 2, we will specifically introduce Newman's spectral method, followed by our improved algorithm, the Random Walk Graph Partition Algorithm, and analyze the computational complexity.

In Section 3, we will detail the Louvain algorithm, followed by our enhanced algorithm, the RGP-Louvain Algorithm.

In Section 4, we conduct experiments to evaluate and compare our algorithms with several others. Evaluating the quality of community detection algorithms is highly important and has garnered considerable attention. As a result, metrics such as Modularity [18, 19], NMI [13], F1-score [26], GDM [1], have been developed to assess clustering quality. Thus, we will use some of these metrics in our experiment to demonstrate the validity and compare the effectiveness of our proposed algorithms on both randomly generated and real data.

2 Random Walk Graph Partition Algorithm

First of all, we recall the definition of the normalized Laplacian matrix as $\mathbf{L} = D^{-1/2}AD^{-1/2}$. Newman's spectral approach is based on spectral analysis of the normalized Laplacian matrix $\mathbf{L}$. Here's a summary of this algorithm:

1. Compute the normalized Laplacian matrix $\mathbf{L} = D^{-1/2}AD^{-1/2}$.

2. Compute the eigenvector corresponding to the eigenvalue with the 2nd most immense absolute value, denote it is $v_\beta = (v_\beta^1, v_\beta^2, \ldots, v_\beta^n)$.

3. Assign vertices to communities based on the $v_\beta = (v_\beta^1, v_\beta^2, \ldots, v_\beta^n)$. If $v_\beta^i \geq 0$, vertex i belongs to community C_1; otherwise, it belongs to community C_2.

4. Generate two induced subgraphs G_1 and G_2 corresponding to communities C_1 and C_2.

5. Repeat steps 1-4 for the subgraphs G_1 and G_2 until further partitioning no longer increases the modularity value.

We note that, $P = D^{-1}A = D^{1/2}AD^{-1/2}$. Random walk and the eigenvalues of the normalized Laplacian matrix are closely related, and many studies have been on this relationship, such as [21]. Therefore, this section will use this relationship to propose an algorithm similar to Newman's spectral method by approaching it using random walks.

Now, we will analyze the relationship between the spectral approach and random walk. First, we recall the following lemma.

Lemma 2.1 *([21, Lemma 1]) The eigenvalues of the matrix P are real and satisfy:*

$$1 = \lambda_1 > \lambda_2 \geq \ldots \geq \lambda_n > -1. \tag{2.1}$$

Moreover, there exists an orthonormal family of vectors $(s_\alpha)_{1\leq\alpha\leq n}$ such that each vector $v_\alpha = D^{-1/2}s_\alpha$ and $u_\alpha = D^{1/2}s_\alpha$ are respectively a right and a left eigenvector associated to the eigenvalue λ_α:

$$\forall\alpha,\ Pv_\alpha = \lambda_\alpha v_\alpha \quad \textit{and } P^T u_\alpha = \lambda_\alpha u_\alpha$$
$$\forall\alpha, \forall\beta,\ v_\alpha^T u_\beta = \delta_{\alpha\beta}$$

This paper only considers the case $\lambda_2 \neq \lambda_3$. We have the following theorem from Lemma 2.1.

Theorem 2.2 *Let i be any vertex of the graph G. Then we have the $j-th$ component of vector $P_{i\bullet}^t - \phi$ and the $j-th$ component of vector s_2 have the same sign for all $j = 1, 2, ..., n$ or opposite signs for all $j = 1, 2, ..., n$, where s_2 is the eigenvector corresponding to the second eigenvalue of the normalized Laplacian matrix $\mathbf{L}$.*

Proof Lemma 2.1 makes it possible to write a spectral decomposition of the matrix P :

$$P = \sum_{\alpha=1}^{n} \lambda_\alpha v_\alpha u_\alpha^T \text{ and } P^t = \sum_{\alpha=1}^{n} \lambda_\alpha^t v_\alpha u_\alpha^T. \tag{2.2}$$

It follows that

$$P_{ij}^t = \sum_{\alpha=1}^{n} \lambda_\alpha^t v_\alpha(i) u_\alpha^T(j) \text{ and } P_{i\bullet}^t = \sum_{\alpha=1}^{n} \lambda_\alpha^t v_\alpha(i) u_\alpha. \tag{2.3}$$

When t tends towards infinity, all the terms $\alpha \geq 2$ vanish. It is easy to show that the first right eigenvector v_1 is constant. By normalizing we have $\forall i$, $v_1(i) = \frac{1}{\sum_k d_k}$ and $\forall j$, $u_1(j) = \frac{d_j}{\sum_k d_k}$. Therefore, we have

$$\lim_{t\to\infty} P_{ij}^t = \lim_{t\to\infty} \lambda_\alpha^t v_\alpha(i) u_\alpha^T(j) = v_1(i) u_1^T(j) = \frac{d_j}{\sum_k d_k} = \phi_j. \tag{2.4}$$

From (2.3), (2.4) and $\lambda_1 = 1$, we have

$$P^t_{i\bullet} - \phi = \sum_{\alpha=2}^{n} \lambda_\alpha^t v_\alpha(i) u_\alpha = \lambda_2^t v_2(i) u_2 + \lambda_3^t v_3(i) u_3 + \ldots + \lambda_n^t v_n(i) u_n, \tag{2.5}$$

this is equivalent to

$$P^t_{i\bullet} - \phi = \lambda_2^t \left(v_2(i) u_2 + \frac{\lambda_3^t}{\lambda_2^t} v_3(i) u_3 + \ldots + \frac{\lambda_n^t}{\lambda_2^t} v_n(i) u_n \right). \tag{2.6}$$

On the other hand, from Theorem 2.2, we have $u_\alpha = D^{-1/2} s_\alpha$. From there, it follows.

$$P^t_{i\bullet} - \phi = \lambda_2^t D^{-1/2} \left(v_2(i) s_2 + \frac{\lambda_3^t}{\lambda_2^t} v_3(i) s_3 + \ldots + \frac{\lambda_n^t}{\lambda_2^t} v_n(i) s_n \right). \tag{2.7}$$

From Theorem 2.2 and $\lambda_2 \neq \lambda_3$, we have $|\lambda_\alpha / \lambda_2| < 1$ with $3 \leq \alpha \leq n$. Therefore, when t tends towards infinity, all the terms in 2.6 with $\alpha \geq 3$ vanish. From there we have the $j-$th component of vector $P^t_{i\bullet} - \phi$ and the $j-$th component of vector $\lambda_2^t v_2(i) s_2$ having the same sign for t large enough and for all $j = 1, 2, ..., n$. Hence the conclusion of the theorem. Furthermore, from (2.7), we deduce this conclusion holds for all $i = 1, 2, ..., n$. □

From Theorem 2.2, we observe that clustering based on random walk and spectral analysis is the same. Therefore, based on a random walk, we can propose a Random Walk Graph Partition Algorithm 1 as follows.

Algorithm 1: Random Walk Graph Partition Algorithm 1

```
Input: Graph $G$, $C \subset V(G)$, $Q = \emptyset$, $t$
Output: Final list of clusters $Q$
Phase 1:
$C_1 = \emptyset$, $C_2 = \emptyset$;
Create induced graph $G'$ from $C$, and select any vertex $i_0$ in cluster $V(G')$;
Calculate $P^t_{i_0\bullet} - \phi = (P^t_{i_0 1} - \phi_1, P^t_{i_0 2} - \phi_2, ..., P^t_{i_0 n} - \phi_n)$ in $G'$;
for each vertex $j$ do
    if $P^t_{i_0 j} - \phi_j \geq 0$ then
        Add $j$ to cluster $C_1$;
    end
    else
        Add $j$ to cluster $C_2$;
    end
end
Phase 2:
if $C_1$, $C_2$ are non-empty and $Q_{C_1}(G) + Q_{C_2}(G) > Q_C(G)$ then
    Apply Phase 1 with $C = C_1$ and apply Phase 1 with $C = C_2$;
end
else
    $Q = Q \cup \{C\}$;
end
```

In Algorithm 1, in phase 1, we need to compute $P^t_{i_0\bullet}$. The computational complexity for calculating $P^t_{i_0\bullet}$ is $O(tm)$, where m is the number of edges in the graph G (see [21]). In phase

2, we need to compute $Q(C, G)$. To calculate $Q(C, G)$, we need to count the number of edges in C (e_C) and the number of edges connected to C (a_C), so the computational complexity of phase 2 does not exceed $O(m)$. Therefore, the computational complexity of each iteration is $O(tm) + O(m) = O(tm)$. Assuming the number of communities in the graph G is k, the computational complexity of Algorithm 1 is $O(tkm)$.

Although the case $\lambda_2 = \lambda_3$ rarely occurs; if it does happen or λ_2 is very close to λ_3, our Random Walk Graph Partition Algorithm 1 will no longer be accurate. Therefore, to improve the effectiveness of our algorithm in these cases, after dividing cluster C into two clusters C_1, C_2, we will add an adjustment step. Specifically, we will review all vertices to see if they deserve to be in the current cluster or if they should be moved to another cluster based on maximizing modularity. From there, we propose the following algorithm. In this part, we shall use the notation C_i to represent the community that includes vertex i and $C_{\bar{i}}$ for the community that does not include vertex i.

Algorithm 2: Random Walk Graph Partition Algorithm 2

```
Input: Graph $G$, $C \subset V(G)$, $Q = \emptyset$, $t$
Output: Final list of clusters $Q$
Phase 1:
$C_1 = \emptyset$, $C_2 = \emptyset$;
Create induced graph $G'$ from $C$, and select any vertex $i_0$ in cluster $V(G')$;
Calculate $P^t_{i_0\bullet} - \phi = (P^t_{i_01} - \phi_1, P^t_{i_02} - \phi_2, ..., P^t_{i_0n} - \phi_n)$ in $G'$;
for each vertex $j$ do
    if $P^t_{i_0j} - \phi_j \geq 0$ then
        Add $j$ to cluster $C_1$;
    end
    else
        Add $j$ to cluster $C_2$;
    end
end
Count.number.move=1
while Count.number.move $\neq$ 0 do
    Count.number.move=0;
    for $i$ in $V(G')$ do
        if $\Delta Q_{i,C_i\setminus\{i\}} < \Delta Q_{i,C_{\bar{i}}}$ then
            Add $i$ to $C_{\bar{i}}$;
            Count.number.move = Count.number.move + 1;
        end
    end
end
Phase 2:
if $C_1$, $C_2$ are non-empty and $Q_{C_1}(G) + Q_{C_2}(G) > Q_C(G)$ then
    Apply Phase 1 with $C = C_1$ and apply Phase 1 with $C = C_2$;
end
else
    $Q = Q \cup \{C\}$;
end
```

The algorithm referred to as Algorithm 2 distinguishes itself from Algorithm 1 solely in the adjustment step, which involves computing $\Delta Q_{i,C}$. Regarding the while loop, it is worth mentioning that this loop only iterates a few times, and the computational complexity of calculating $\Delta Q_{i,C}$ does not surpass $O(m)$. Consequently, the computational complexity of Algorithm 2 is equivalent to that of Algorithm 1.

3 Random Walk Graph Partition Louvain algorithm

The Louvain algorithm [2] stands out for its simplicity and elegance. It optimizes a Modularity through two primary phases:
Phase 1: vertices assess potential relocation to neighboring communities by maximizing Modularity increase using the formula:

$$\Delta Q_{i,C_j} = \left[\frac{\Sigma_{in} + 2k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right] \tag{3.1}$$

Here, $\Delta Q_{i,C_j}$ signifies Modularity change upon placing vertex i into community C_j, Σ_{in} is the sum of weights of links within the transitioning community, Σ_{tot} is the sum of weights of links to vertices in the transitioning community, k_i is the weighted degree of i, $k_{i,in}$ is the sum of weights of links between i and other vertices in the transitioning community, and m is the sum of weights of all links in the network.
Phase 2: Consolidates vertices within the same community to form a new network. Self-loops denote intra-community links, and weighted edges represent inter-community connections.

The algorithm iterates these phases until Modularity ceases to increase.

The Louvain algorithm is very famous not only because of its fast calculation speed but also because of its high algorithm accuracy. However, this algorithm could be less effective when the network has an unclear community structure. In the paper [25], Leiden proposed one of the most prominent improvements of the Leiden algorithm - the author added a phase of fine-tuning local communities after the first phase of the Louvain algorithm.

The Leiden algorithm for fine-tuning local communities needs to work more effectively in cases where the community could be more opaque. Therefore, this paper will propose an algorithm more effectively fine-tuning local communities. We will use the Random Walk Graph Partition Algorithm mentioned above to refine each cluster obtained from Phase 1. Consequently, we propose a new algorithm named the **Random Walk Graph Partition Louvain Algorithm** (or **RWGP-Louvain Algorithm** for short).

Algorithm 3: Random Walk Graph Partition Louvain Algorithm

```
Input: Network $G_{ori}$, $\mathcal{P} = \emptyset$, $t$
Output: FinalCommunities - Final list of communities
$G = G_{ori}$
Phase 1:
for $i$ in $V(G)$ do
    $C_i = \{i\},\ \mathcal{P} = \mathcal{P} \cup C_i$;
end
while some vertices are moved do
    for $i$ in $V(G)$ do
        for neighboring community $C_j$ of $i$ do
            Calculate $\Delta Q_{i,C_j}$ according to Formula 3.1;
            Add $i$ to the community $C_{i_0}$ with maximizing $\Delta Q_{i,C_{j_0}}$;
        end
    end
end
for $C_j$ in $\mathcal{P}$ do
    if $C_j = \emptyset$ then
        $\mathcal{P} = \mathcal{P} \setminus \{C_j\}$;
    end
end
Phase 2:
If the clusters in $\mathcal{P}$ contain supervertices , then return the clusters containing the
  vertices of the original graph $G_{ori}$.
for $C_j$ in $\mathcal{P}$ do
    Apply Random Walk Graph Partition Algorithm 1 or Random Walk Graph
      Partition Algorithm 2 with $G = G_{ori}$, $C = C_j$, $Q = \emptyset$;
    $\mathcal{P} = \mathcal{P} \setminus \{C_j\} \cup Q$
end
Phase 3:
Create a new network $G'$ by consolidating vertices within the same community to
  supervertices ;
for each pair of vertices $u$ and $v$ in the same community do
    Add a self-loop to the community vertex for $u$ and $v$;
end
for each edge between vertices in different communities do
    Add a weighted edge between the corresponding community vertices ;
end
We will repeat phases 1 to 3 until modularity no longer increases.;
```

In Algorithm 3, compared to the Louvain algorithm, each iteration of the algorithm involves applying the Random Walk Graph Partition Algorithm to $G = G_{ori}$ and $C = C_j$ for each $C_j \in \mathcal{P}$. For each C_j, the computational complexity is $O(tm_{C_j})$, where m_{C_j} is the number of edges in cluster C_j. Consequently, the overall computational complexity for this part is $O(tm)$, where m represents the number of edges in the original graph G_{ori}.

4 Experiments

Evaluating a community detection algorithm is an important issue; hence, it has attracted considerable research attention. Among them, the data used for experiments is the most crucial aspect. Furthermore, data with a known community structure is a type of data that can be readily utilized to evaluate the clustering quality through various metrics. A classical approach is to use randomly generated graphs with given communities. Here, we will use this approach and generate the graphs as follows.

Planted l-partition model: The first generator model is the planted l-partition model [6]. By determining the number of groups l, the number of vertices in each group g, and two probabilities of inter-cluster p_{in} and intra-cluster p_{out}, we obtain one random graph with some Property:

- The average degree of one vertex is $E\left[k\right] = p_{in}(g-1) + p_{out}g(l-1)$.
- All communities have the same size.
- All vertices have approximately the same degree because of each community. It can be seen as one random graph proposed by Erdős and Rényi. Each pair of vertices is connected in those random graphs with equal probability p_{in} independent of other pairs.

Gaussian random partition generator: The next generator graph model is Gaussian random partition generator [6], which overcomes the part disadvantage of the planted $l-$ partition model above the vertex degree distribution. Unlike the planted $l-$partition model, the community size in the Gaussian random partition generator is a random variable of Gaussian distribution. The parameters What needs to be determined for the Gaussian random partition generator are:

- Number of vertices in the graph: N.
- Mean of community's size: m and variance of community's size: σ.
- Edge probability of inter p_{in} and intra-cluster p_{out}.

After clustering a graph, we need to evaluate its quality. Therefore, we will present a metric to evaluate the clustering quality next.

4.1 Evaluating metrics

In our experiments, we will use two metrics to compare the algorithms. The first metric is to use modularity (formula 1.1). The second metric is that we use Normalized Mutual Information (NMI). Normalized Mutual Information [13] quantifies the similarity between true class labels Y and predicted cluster assignments C. It is computed as:

$$\mathrm{NMI}(Y, C) = \frac{2 \times \mathrm{MI}(Y, C)}{\mathrm{H}(Y) + \mathrm{H}(C)}, \tag{4.1}$$

where:

- $\mathrm{MI}(Y, C)$ is the Mutual Information between Y and C,
- $\mathrm{H}(Y)$ and $\mathrm{H}(C)$ are the entropies of Y and C respectively.

The NMI values range from 0 to 1, with higher values indicating better clustering alignment with true class labels. It is a normalized measure commonly used in cluster evaluation.

4.2 Experiments for Random Walk Graphs Partition Algorithm

In this section, we conduct experiments to compare the effectiveness of our two algorithms, Random Walk Graphs Partition Algorithm 1 and Random Walk Graphs Partition Algorithm 2, with the algorithms Louvain [2] and Newman's Spectral Method [16]. The experiments involve randomly generated graphs using the Gaussian random generator and Planted-l partition models. Given that this is not the primary focus of our paper, we will perform a limited set of experiments.

We conduct ten trials on randomly generated graphs for each experiment using either the Gaussian random generator or the Planted-l partition model. Subsequently, we calculate Modularity for the clustering results obtained from different algorithms.

Finally, we present graphical representations of the Modularity values corresponding to Random Walk Graphs Partition Algorithm 1 (RWGP1), Random Walk Graphs Partition Algorithm 2 (RWGP2), Louvain [2], and Newman's Spectral Method [16] (Newman).

4.2.1 Experiments for Random Walk Graphs Partition Algorithm on the random graph generated by the Gaussian random generator model

With the Gaussian random generator model, we will experiment on graphs with the number of vertices ranging from a few hundred to tens of thousands. For each type of graph, we will fix $p_{in} = 0.7$, and we will explore p_{out} values of 0.01, and 0.03, corresponding to graphs with clear community structure to graphs with unclear community structure. And we will set the variance of the community's size $\delta = 2.5$.

Experiment 1: Experiment on the random graph generated by the Gaussian random generator model.

Using the Gaussian random generator model with a number of vertices N, the mean of the community's size m is taken with a uniform distribution in the following corresponding intervals: $N \in [500; 1000]$, $m \in [50; 100]$. In implementing the Random Walk Graphs Partition Algorithm 1 and Random Walk Graphs Partition Algorithm 2, we set t to 15. We present these results in Figure 1.

Experiment 2: Experiment on the random graph generated by the Gaussian random generator model.

We use the Gaussian random generator model with a number of vertices N, the mean of the community's size m taken with a uniform distribution in the following corresponding intervals: $N \in [1000; 2000]$, $m \in [100; 200]$. In implementing the Random Walk Graphs Partition Algorithm 1 and Random Walk Graphs Partition Algorithm 2, we set t to 40. We present these results in Figure 2.

4.2.2 Experiments for Random Walk Graphs Partition Algorithm on the random graph generated by the Planted-l partition model

With the Planted-l partition model, we also fix $p_{in} = 0.7$ and explore p_{out} values of 0.01, and 0.03, corresponding to graphs with clear community structure to graphs with unclear community structure.

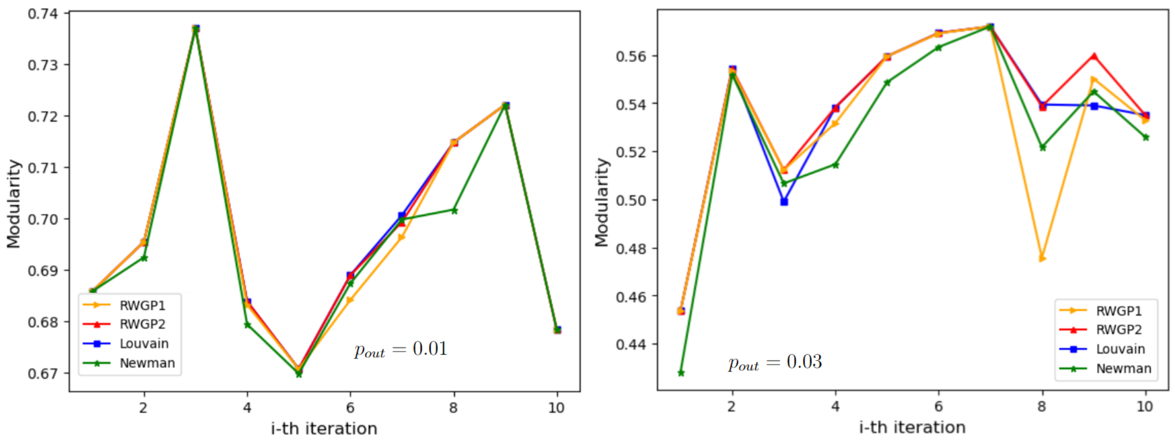

Figure 1: Modularity obtained in **Experiment 1** using Gaussian random generator model with $N \in [500; 1000]$, $m \in [50; 100]$.

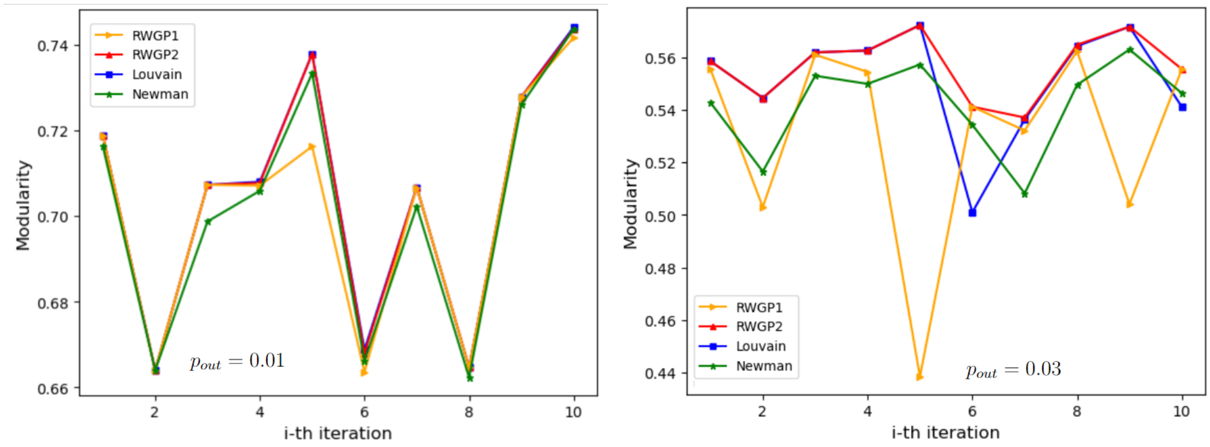

Figure 2: Modularity obtained in **Experiment 2** using Gaussian random generator model with $N \in [1000; 2000]$, $m \in [100; 200]$.

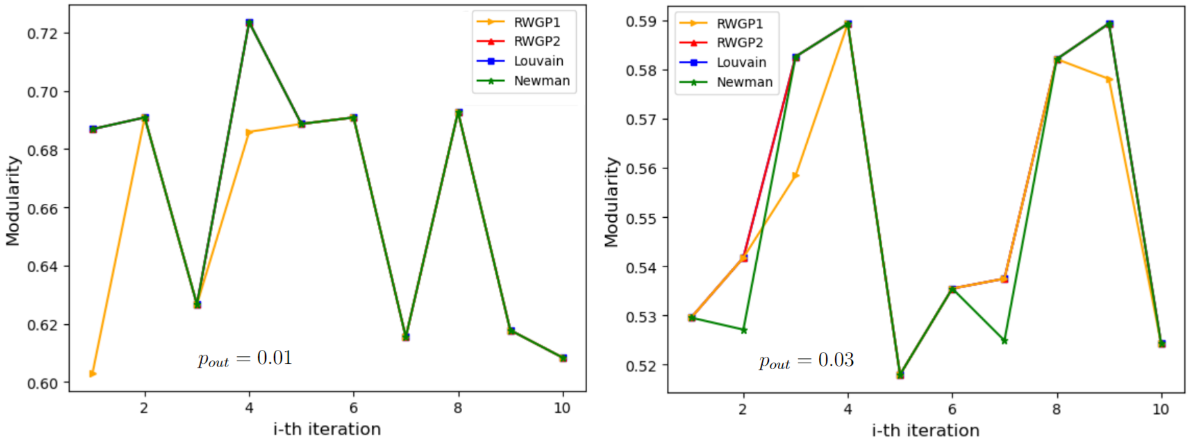

Figure 3: Modularity obtained in **Experiment 1** using the Planted-l partition model with $g \in [3; 5]$, $l \in [50; 70]$.

Experiment 1: Experiment on the random graph generated by the Planted-l partition model.

Using the Planted-l partition model with a number of communities l, the size of each community g is taken with a uniform distribution in the following corresponding intervals: $g \in [3; 5]$, $l \in [50; 70]$. In implementing the Random Walk Graphs Partition Algorithm 1 and Random Walk Graphs Partition Algorithm 2, we set t to 15. We present these results in Figure 3.

Experiment 2: Experiment on the random graph generated by the Planted-l partition model.

We use the Planted-l partition model with a number of communities l, the size of each community g taken with a uniform distribution in the following corresponding intervals: $g \in [5; 10]$, $l \in [100; 200]$. In implementing the Random Walk Graphs Partition Algorithm 1 and Random Walk Graphs Partition Algorithm 2, we set t to 40. We present these results in Figure 4.

4.3 Experiments for Random Walk Graphs Partition Louvain Algorithm on graphs are randomly generate

In this section, we conduct experiments to compare the effectiveness of our proposed Random Walk Graphs Partition Louvain Algorithm (RWGP-Louvain) with existing algorithms, namely the original Louvain algorithm [2], the Fast Louvain algorithm [28], and the Leiden algorithm [25]. The experiments involve randomly generated graphs using the Gaussian random generator and Planted-l partition models.

We perform ten trials on randomly generated graphs for each experiment utilizing either the Gaussian random generator or the Planted-l partition model. Subsequently, we calculate Modularity (use formula 1.1) for the clustering results obtained from different algorithms and calculate the NMI (use formula 4.1) between the clustering results obtained when applying the algorithms and the original clustering generated when creating the graph.

Finally, we present the Modularity and NMI values corresponding to RWGP-Louvain, Louvain [2], Fast Louvain [28], and Leiden [25] algorithms through graphical representations. We

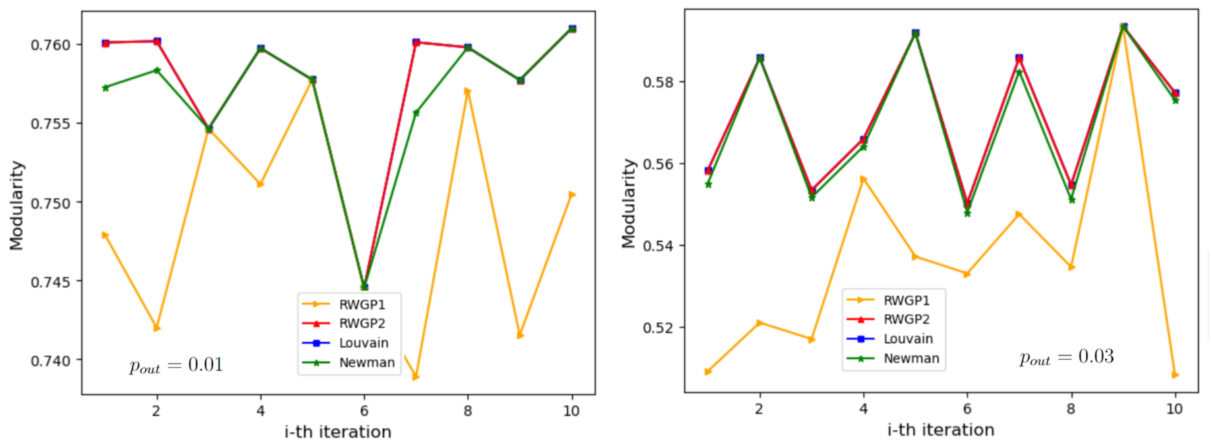

Figure 4: Modularity obtained in **Experiment 2** using the Planted-l partition model with g, l taken with $g \in [5; 10]$, $l \in [100; 200]$, and $p_{in} = 0.7$.

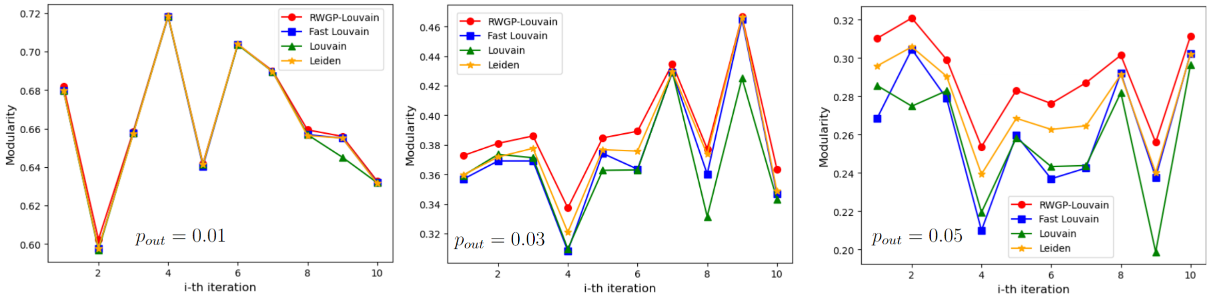

Figure 5: Modularity was observed in **Experiment 1** using the Gaussian random generator model with $N \in [500; 1000]$ and $m \in [20; 30]$.

note that we apply the Random Walk Graph Partition Algorithm 2 in Phase 2 of the Random Walk Graph Partition Louvain Algorithm.

4.3.1 Experiments for Random Walk Graphs Partition Louvain Algorithm on the random graph generated by the Gaussian random generator model

With the Gaussian random generator model, we will experiment on graphs with the number of vertices ranging from a few hundred to tens of thousands. For each type of graph, we will fix $p_{in} = 0.7$, and we will explore p_{out} values of 0.01, 0.03, and 0.05, corresponding to graphs with clear community structure to graphs with unclear community structure. And we will set the variance of the community's size $\delta = 2.5$.

Experiment 1: Random Graphs from the Gaussian Random Generator Model

Using the Gaussian Random Generator Model, we set the number of vertices (N) and the mean community size (m) with a uniform distribution within $N \in [500; 1000]$ and $m \in [20; 30]$. For RWGP-Louvain Algorithm, t is set to 15. The results are depicted in Figures 5 and 6.

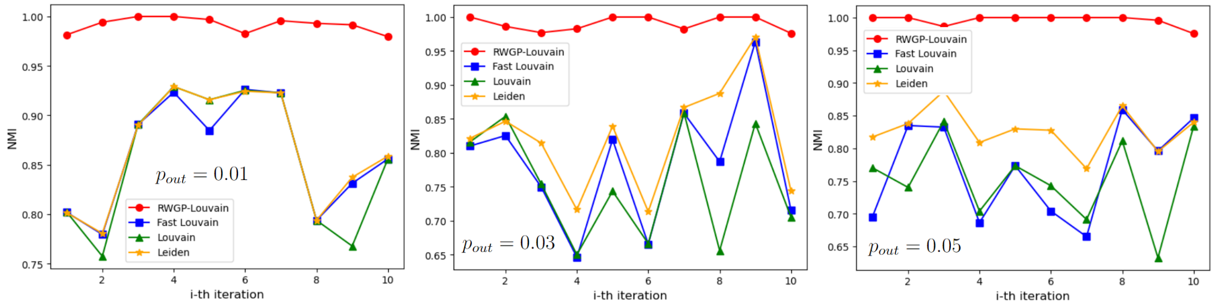

Figure 6: NMI was observed in **Experiment 1** using the Gaussian random generator model with $N \in [500; 1000]$ and $m \in [20; 30]$.

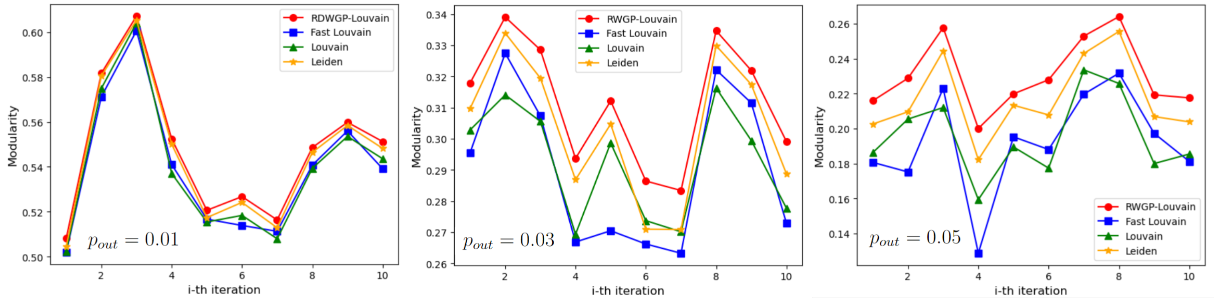

Figure 7: Modularity was observed in **Experiment 2** using the Gaussian random generator model with $N \in [2000; 4000]$ and $m \in [50; 70]$.

Experiment 2: Random Graphs from the Gaussian Random Generator Model

Using the Gaussian Random Generator Model, we set the number of vertices (N) and the mean community size (m) with a uniform distribution within $N \in [2000; 4000]$ and $m \in [50; 70]$. For RWGP-Louvain Algorithm, t is set to 25. The results are illustrated in Figures 7 and 8.

Experiment 3: Random Graphs from the Gaussian Random Generator Model

Using the Gaussian Random Generator Model, we set the number of vertices (N) and the mean community size (m) with a uniform distribution within $N \in [4000; 8000]$ and $m \in [100; 150]$. For RWGP-Louvain Algorithm, t is set to 40. The outcomes are presented in Figures 9 and 10.

4.3.2 Experiments for RWGP-Louvain Algorithm on the random graph generated by the Planted-l partition model

With the Planted-l partition model, we also fix $p_{in} = 0.7$ and explore p_{out} values of 0.01, 0.03, and 0.05, corresponding to graphs with clear community structure to graphs with unclear community structure.

Experiment 1: Random Graphs from the Planted-l Partition Model

Using the Planted-l Partition Model, we set the number of communities (l) and the size of each community (g) are selected with a uniform distribution within the intervals $g \in [30; 50]$ and

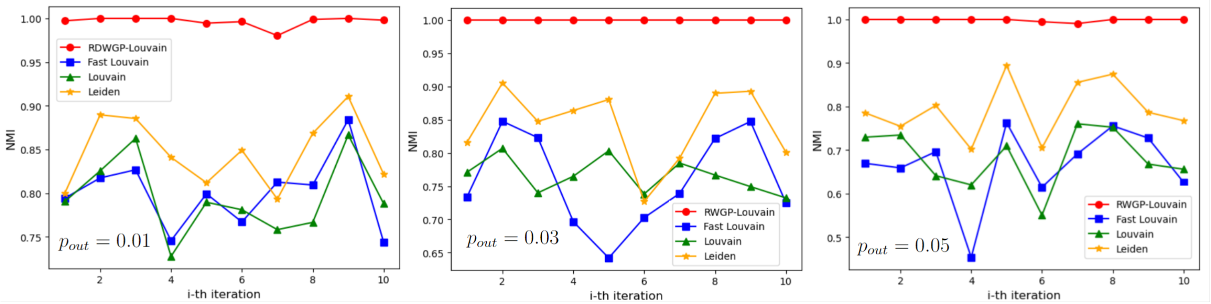

Figure 8: NMI was observed in **Experiment 2** using the Gaussian random generator model with $N \in [2000; 4000]$ and $m \in [50; 70]$.

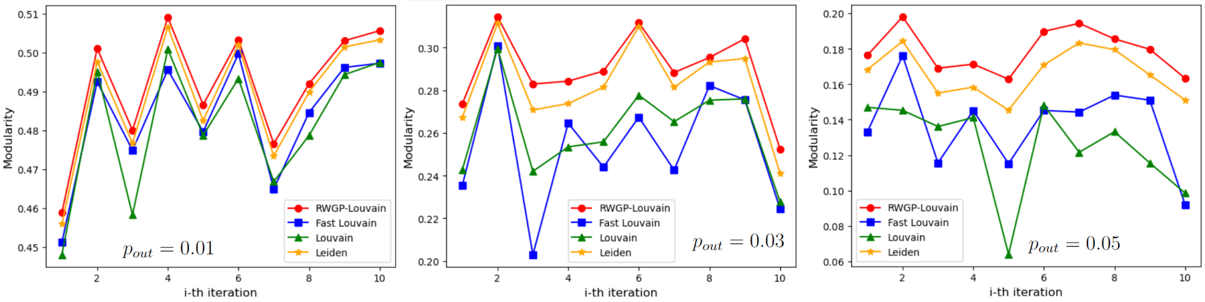

Figure 9: Modularity was observed in **Experiment 3** using the Gaussian random generator model with $N \in [4000; 8000]$ and $m \in [50; 70]$.

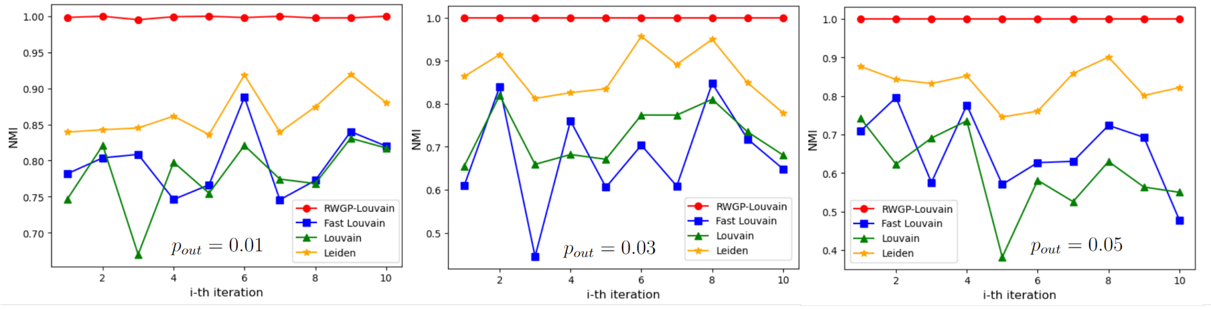

Figure 10: NMI was observed in **Experiment 3** using the Gaussian random generator model with $N \in [4000; 8000]$ and $m \in [100; 120]$.

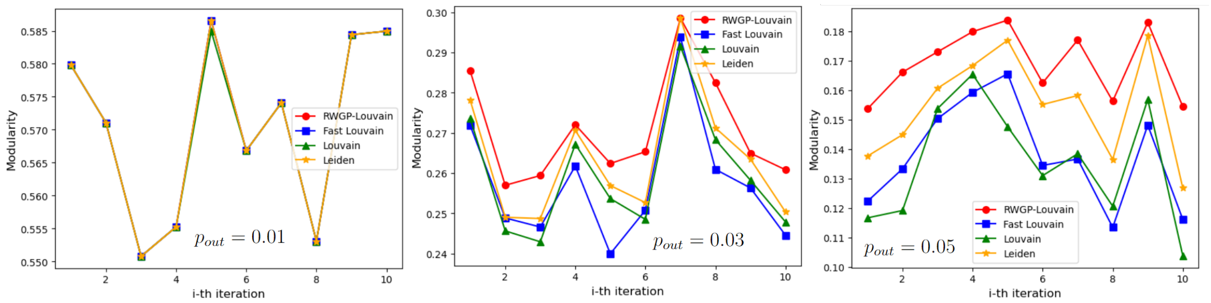

Figure 11: Modularity was observed in **Experiment 1** using the Planted-l partition model with $g \in [30; 50]$, $l \in [30; 50]$.

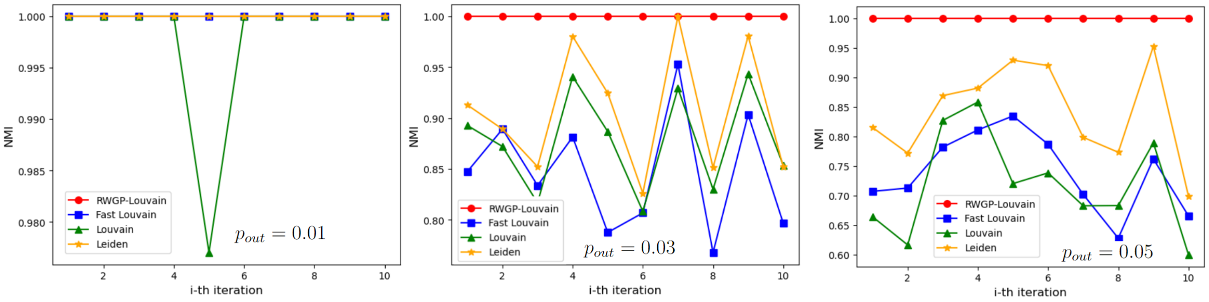

Figure 12: NMI was observed in **Experiment 1** using the Planted-l partition model with $g \in [30; 50]$, $l \in [30; 50]$.

$l \in [30; 50]$. For RWGP-Louvain Algorithm, t is set to 25. The results are depicted in Figures 11 and 12.

Experiment 2: Random Graphs from the Planted-l Partition Model

Using the Planted-l Partition Model, we set the number of communities (l) and the size of each community (g) are selected with a uniform distribution within the intervals $g \in [50; 70]$ and $l \in [50; 70]$. For RWGP-Louvain Algorithm, t is set to 25. The results are illustrated in Figures 13 and 14.

4.4 Experiments on real data

Before performing the experiments, we will introduce some famous real data used in this section.
Zachary's karate club: Wayne W. Zachary examined a karate club's social network from 1970 to 1972, detailed in [29]. The network, featuring 34 members and capturing interactions beyond the club, gained prominence as a community structure example in networks following its analysis by Michelle Girvan and Mark Newman in 2002 [8].
College football: The college football network, examined in [8], serves as a benchmark for community detection. It illustrates the games played by Division I colleges in the autumn of 2000, with each vertex representing a football team and each edge representing a regular season game. With 115 vertices and 616 edges, the network can be divided into 12 communities based on athletic conferences, each comprising 8 to 12 teams.

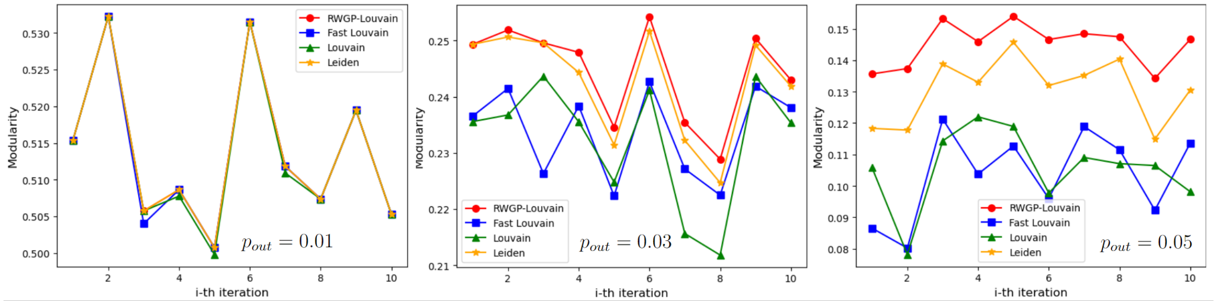

Figure 13: Modularity was observed in **Experiment 2** using the Planted-l partition model with $g \in [50; 70]$, $l \in [50; 70]$.

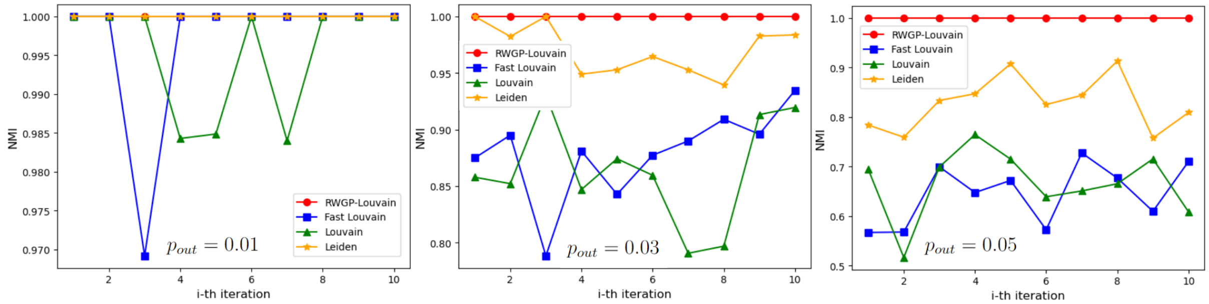

Figure 14: NMI was observed in **Experiment 2** using the Planted-l partition model with $g \in [50; 70]$, $l \in [50; 70]$.

Jazz network: Data was sourced from The Red Hot Jazz Archive digital database [24], comprising 198 bands active between 1912 and 1940, predominantly in the 1920s. The database identifies musicians in each band, but distinguishing their temporal involvement is challenging, hindering the study of the collaboration network's temporal evolution. The remaining 1275 musicians' names are dispersed across the bands.
Metabolic network: As in [10], a metabolic network encompasses the entire metabolic and physical processes governing a cell's physiological and biochemical characteristics. It includes metabolic reactions, pathways, and the regulatory interactions orchestrating these reactions.

In addition, we will also perform experiments on the following real data: Hamster households, hamster friendships, and Asoiaf. These data we can see in [12].

Now, we will conduct experiments to compare the effectiveness of our proposed Random Walk Graphs Partition Louvain Algorithm (RWGP-Louvain) with existing algorithms, namely the original Louvain algorithm [2], the Fast Louvain algorithm [28], and the Leiden algorithm [25] on real data. After obtaining the clustering results, we will calculate modularity (using the formula 1.1) and record the modularity results in the following table 1.

4.5 Conclusion of the experiments

The above results show that our algorithms are efficient in almost all experiments.

- In the Subsection 4.2, we perform some simple experiments using two models Gaussian random generator model and Planted-l partition model to compare our Random Walk Graphs Partition Algorithm 1 and Random Walk Graphs Partition Algorithm 2 with Louvain algorithm [2], Newman's Spectral Method [16]. We see that our algorithm works well on graphs with clear community structures. On graphs with unclear structures, our Random Walk Graphs Partition Algorithm 2 algorithm still works more efficiently and has less computational complexity than Newman's Spectral Method.

- In Subsection 4.3, we performed experiments comparing the effectiveness of our proposed Random Walk Graphs Partition Louvain Algorithm and the Louvain uciteLouvain, Fast Louvain Algorithm [28] and Leiden algorithm [25] through graphs on randomly generated graphs using Gaussian random generator model and Planted-l partition model. The algorithms we investigated on graphs with apparent community structures produce good results. Our Random Walk Graphs Partition Louvain Algorithm performs much better than the Louvain, Fast Louvain, and Leiden algorithms for graphs with unclear community structures.

- Furthermore, when performing experiments using real data, our Random Walk Graphs Partition Louvain Algorithm is also more effective than the remaining algorithms on some real data.

5 Conclusion and further work

This paper introduced a novel approach to community detection through graph partitioning by leveraging a random walk strategy akin to the spectral method presented in [16], referred to as the Random Walk Graph Partition Algorithm. Our proposed algorithm demonstrates superior computational efficiency compared to the spectral algorithm discussed in [16].

Moreover, recognizing the Random Walk Graph Partition Algorithm as a phase within the Louvain algorithm, we introduce a new algorithm called the Random Walk Graph Partition Louvain Algorithm. This algorithm retains computational complexity equivalent to the Louvain algorithm while achieving enhanced efficiency, particularly in scenarios involving graphs with ambiguous community structures.

To validate the rationale and efficacy of our proposed algorithms, we conduct experiments on randomly generated and real-world datasets. The results underscore the effectiveness of our approaches, reinforcing their applicability in practical community detection scenarios.

Acknowledgments

This research was supported by the International Centre of Research and Postgraduate Training in Mathematics, Institute of Mathematics, project code: ICRTM04-2021.01, and the postdoctoral scholarship of Vingroup Innovation Foundation, scholarship code: VINIF.2023.STS.53.

References

[1] S. Bakhtar, H.A. Harutyunyan. A new metric to compare local community detection algorithms in social networks using geodesic distance. J Comb Optim 44, 2809–2831 (2022).

[2] V. D. Blondel1, J.-L. Guillaume1, R. Lambiotte1 and E. Lefebvre1, Fast unfolding of communities in large networks, Journal of Statistical Mechanics: Theory and Experiment, Volume 2008, October 2008.

[3] F. R. K. Chung, Spectral Graph Theory, CBMS Regional Conference Series in Mathematics, No. 92.

[4] D. T. Dat, D. D. Hieu, P. T. H. Duong, Community detection in directed graphs using stationary distribution and hitting times methods, Social Network Analysis and Mining volume 13, Article number: 80 (2023).

[5] S. v. Dongen. Graph Clustering by Flow Simulation. PhD thesis, University of Utrecht, May 2000.

[6] S. Fortunato, "Community detection in graphs," Physics Reports, vol. 486, pp. 75–174, 2010.

[7] G. W. Flake, S. Lawrence, C. L. Giles, and F. M. Coetzee. Self-organization and identification of web communities. Computer, 35(3):66-71, 2002.

[8] M. Girvan, M.E.J Newman, Community structure in social and biological networks, Proc. Natl. Acad. Sci. USA 99 (2002) 7821–7826

[9] D. D. Hieu, P. T. H. Duong, Overlapping community detection algorithms using Modularity and the cosine (preprint).

[10] H. Jeong , B. Tombor, R. Albert, Z. N. Oltvai, and A.-L. Barabási, The large-scale organization of metabolic networks. Nature 407, 651–654, 2000.

[11] J. Kleinberg and S. Lawrence. The structure of the web. Science, 294(5548):1849-1850, 2001.

[12] J. Kunegis. KONECT – The Koblenz Network Collection. In Proc. Int. Conf. on World Wide Web Companion, pages 1343–1350, 2013.

[13] T.O. Kvålseth, Entropy and correlation: Some comments. IEEE Trans. Syst. Man Cybern. 1987, SMC-17, 517–519.

[14] L. Lovász. Random walks on graphs: a survey. In Combinatorics, Paul Erdős is eighty, Vol. 2 (Keszthely, 1993), volume 2 of Bolyai Soc. Math. Stud., pages 353– 397. János Bolyai Math. Soc., Budapest, 1996.

[15] C. Moore. The Computer Science and Physics of Community Detection: Landscapes, Phase Transitions, and Hardness. Bull. EATCS 121, 2017.

[16] M. E. J. Newman, Spectral methods for network community detection and graph partitioning, Phys. Rev. E, vol. 88, p. 042822, October 2013.

[17] M. E. J. Newman, Modularity and community structure in networks. Proceedings of the National Academy of Sciences of the United States of America. 103 (23): 8577–8696(2006).

[18] M. E. J. Newman. Fast algorithm for detecting community structure in networks. Physical Review E, 69(6):066133, 2004.

[19] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. Physical Review E, 69(2):026113, 2004.

[20] M.E.J. Newman and M. Girvan, Finding and evaluating community structure in networks, Phys. Rev. E 69 (2004) 026–113.

[21] P. Pons and M. Latapy. Computing communities in large networks using random walks, Journal of Graph Algorithms and Applications, volume 10, no. 2, 2006, Pages 191–218, 2006.

[22] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A.-L. Barabási. Hierarchical Organization of Modularity in Metabolic Networks. Science, 297(5586):15511555, 2002.

[23] J-F. Rural, K. Venkatesan, T. Hao, T. Hirozane-Kishikawa, A. Dricot, et al. Towards a proteome-scale map of the human protein-protein interaction network. Nature 437: 1173. 2005.

[24] The Red Hot Jazz Archive, available at http://www.redhotjazz.co

[25] V.A. Traag, L. Waltman, and N.J. Van Eck, From Louvain to Leiden: guaranteeing well-connected communities. Sci Rep 9, 5233 (2019). https://doi.org/10.1038/s41598-019-41695-z

[26] J. Yang and J. Leskovec, "Overlapping community detection at scale: a nonnegative matrix factorization approach," in ACM International conference on Web search and data mining, Feb. 2013, pp. 587–596.

[27] L. Waltman, and N. J. van Eck, A smart local moving algorithm for large-scale modularity-based community detection. Eur. Phys. J. B 86, 471, https://doi.org/10.1140/epjb/e2013-40829-0 (2013).

[28] J. Zhang, J. Fei, X. Song, J. Feng, An Improved Louvain Algorithm for Community Detection, Mathematical Problems in Engineering, vol. 2021, Article ID 1485592, 14 pages, 2021. https://doi.org/10.1155/2021/1485592.

[29] Zachary, W. An information flow model for conflict and fission in small groups. J. Anthropol. Res. 33, 452–473, 1977.

Graph, $G = (\|V\|, \|E\|)$	Louvain	RWGP-Louvain	Leiden	Fast Louvain
Karate [10], $G = (,)$	0.23948060486	0.2450690335	0.2404667981	0.24046679815
Metabolic network [10], $G = (453, 2025)$	0.29602170400	0.3054383173	0.296554793	0.29486578265
College football [8], $G = (115, 613)$	0.5357493566	0.5400072917	0.53574935665	0.53161915964
Jazz network [24], $G = (198, 2742)$	0.28338616257	0.285041265858	0.282838052	0.0.282149224
Hamster households [12], $G = (921, 4032)$	0.16785046267	0.2058422255	0.1983055447	0.1940693450
Hamsters friendships [12], $G = (1858, 12534)$	0.30967346281	0.3308066312	0.32164593361	0.3095750742
Asoiaf [12], $G = (796, 2823)$	0.51824388226	0.53709741434	0.54042492159	0.51858836421

Table 1: In this table, we present the values of Modularity (using the formula 1.1) corresponding to the clustering results of the algorithms.