Comparative Analysis of Community Detection Algorithms: Synchronous, Asynchronous, and Fast Label Propagation on Synthetic and Empirical Networks

- Complex Networks project -

Matilde Grassi

November 2024

Abstract

This project evaluates and compares the performance of three community detection algorithms: the synchronous label propagation algorithm, the asynchronous label propagation algorithm and the fast label propagation algorithm. The comparison focuses on execution speed and clustering quality across different types of networks, including synthetic networks such as Erdős-Rényi, Barabási-Albert, Forest Fire, Stochastic Block Model and empirical networks. The implementation utilizes Python's NetworkX and Timeit libraries, with source code available on Github [2].

1 Introduction

The detection of clusters is a fundamental task in complex network analysis, with applications ranging from social network analysis to biological networks. This project compares the performance of three prominent algorithms for community detection: the **synchronous label propagation algorithm** (Syn. LPA), the **asynchronous label propagation algorithm** (Asyn. LPA), and the **fast label propagation algorithm** (FLPA). The comparison focuses on both speed and quality of the clustering results. The project follows the comparative analysis presented in the article [4], which pioneered the development and implementation of the FLPA algorithm in C.

The first part of the study involves testing these algorithms on synthetic networks, including **Erdos-Rényi** (E-R), **Barabási-Albert** (B-A), **Forest Fire**, and **Stochastic Block Model** (SBM) generated with varying parameters, such as the number of nodes, average degree, and other structural properties, to simulate different types of topologies. For SBM networks, the quality of the detected communities is evaluated using the **Normalized Mutual Information** (NMI) which measures the similarity between two clustering results by comparing the amount of shared information, normalized to account for the number of clusters and nodes, with values ranging from 0 (no similarity) to 1 (perfect agreement). This approach facilitates a comparison of the algorithms' ability to accurately identify communities in controlled settings while considering performance across diverse synthetic network types.

The second part focuses on empirical networks, starting with small real-world networks where the partitioning is known. In these cases, the NMI is calculated to evaluate the quality of the clustering in comparison to the actual network structure. The analysis then extends to larger empirical networks, examining both the computation time and the modularity of the communities detected by the algorithms. The **modularity** provides a measure of the quality of the community structure and is used

to assess how effectively each algorithm identifies meaningful clusters in large and complex datasets.

All codes for the project are written in Python. The **NetworkX** library is utilized for network analysis, synthetic network generation and community detection, while the **Timeit** library is employed to measure the computation times of the algorithms. The complete source code and additional materials are available on the project's Github repository [2].

By analyzing these three algorithms across synthetic and empirical networks, the study aims to provide a comprehensive comparison of their efficiency and accuracy, highlighting the trade-offs between computational speed and the quality of the community detection results.

2 Theoretical analysis of LPA and FLPA

2.1 Label propagation algorithm

The label propagation algorithm is a method for detecting communities in networks by iteratively propagating labels through the graph. The three variants of this algorithm tested in the project are presented below.

2.1.1 Synchronous label propagation algorithm

In the synchronous version, all nodes update their labels simultaneously at each step. Initially, every node is assigned a unique label. During each iteration, each node adopts the label that is most frequent among its neighbors; ties are resolved randomly. This process repeats until the labels stabilize or a maximum number of iterations is reached, at which point, nodes with the same label are grouped into communities. The synchronous approach ensures all updates are based on the state of the network from the previous iteration, making it parallelizable but potentially prone to oscillations. The algorithm is simple, scalable, and requires no prior knowledge of the number of communities but may produce non-deterministic results due to randomness in tie-breaking.

For this project, the synchronous LPA is implemented using the function `nx.community.label_propagation_communities(Graph)` from the NetworkX library.

2.1.2 Asynchronous label propagation algorithm

The asynchronous label Propagation Algorithm is a variant of LPA where nodes update their labels one at a time, in a random order. Initially, each node is assigned a unique label. During each step, a node selects the most frequent label among its neighbors and immediately updates its own label based on this choice; ties are resolved randomly. Since updates happen one node at a time, subsequent nodes can consider the most recent updates of their neighbors during the same iteration. This approach avoids the simultaneous update dependency of the synchronous version, often leading to faster convergence and reducing the risk of oscillations, though it is less suited for parallelization. Nodes with the same label at convergence form communities.

For this project, the asynchronous LPA is implemented using the function `nx.community.asyn_lpa_communities(Graph)` from the NetworkX library.

2.1.3 Fast label propagation algorithm

The fast label propagation algorithm is an optimized version of the LPA, designed to speed up the community detection process. Like LPA, each node in the graph has an associated label, and nodes update their labels based on the majority label of their neighbors. However, FLPA improves efficiency by using a queue to track only the nodes that need to be processed. When a node's label changes, its neighbors with a different label are added to the queue, ensuring that only nodes with changed labels in their neighborhood are considered in subsequent updates. This significantly reduces the number of nodes that need to be processed, making the algorithm faster. At each step, a node is removed

from the queue and its label is updated, and this process continues until the queue is empty. FLPA guarantees that when the algorithm terminates, each node's label is maximal, meaning no other label is more frequent in its neighborhood [4].

This method speeds up the label propagation process by focusing on the most relevant nodes, unlike LPA, which considers all nodes in every iteration.

For this project, the Fast Label Propagation Algorithm is implemented using the function `nx.community.fast_label_propagation_communities(Graph)` from the NetworkX library.

3 Analysis on synthetic networks

The evaluation of community detection algorithms on synthetic networks provides a controlled environment to study their performance under varying network parameters and topologies.

3.1 Erdos-Rényi network

The Erdos-Rényi (E-R) networks were generated with 10,000 nodes and an average degree $< k >$ increased from 10 to 140. This allowed for a systematic exploration of how varying edge density impacts the algorithms' execution time and clustering outcomes.

The E-R graph is generated in Python using the NetworkX library. To improve computational efficiency, the function `nx.fast_gnp_random_graph` was used instead of the standard `nx.erdos_renyi_graph`.

The following function was used to create the ER networks:

```
def generate_er_graph(nodes, avg_degree):
    p = avg_degree / (nodes - 1)  # Probability of edge creation
    G = nx.fast_gnp_random_graph(nodes, p)  # Faster version of ER graph
    return G
```

This function computes the connection probability p based on the desired average degree and generates an ER graph accordingly.

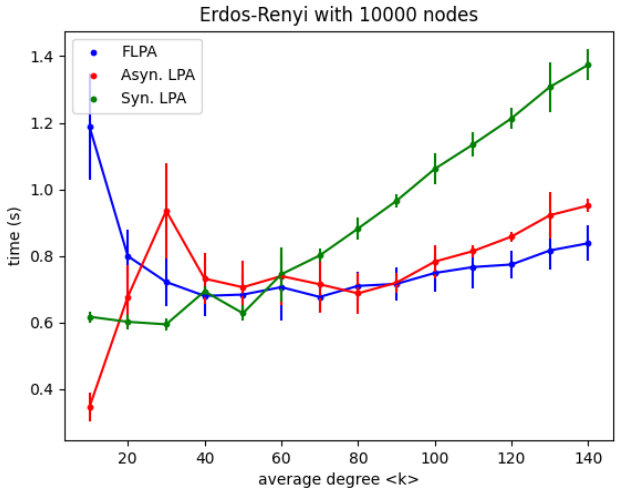


Figure 1: Execution times for synchronous LPA, asynchronous LPA, and FLPA on E-R networks with 10,000 nodes and varying average degree ⟨k⟩. Results represent the mean and standard deviation over 50 runs, reflecting the randomness inherent in the algorithms' processes.

The results of this analysis are shown in the plot in Figure 1. It is evident that FLPA outperforms the other two algorithms for denser networks, i.e. $< k > \geq 80$. In fact, as the average degree increases, FLPA exhibits a slower growth in execution time compared to the other methods. Synchronous LPA demonstrates the highest computational cost for denser networks due to the simultaneous label updates for all nodes. Asynchronous LPA lies in between the two, offering moderate efficiency. However, it is important to note that the FLPA algorithm does not appear to be as efficient for lower edge densities, probably due to the fact that the E-R graph lacks a well-defined community structures. This suggests that FLPA is especially well-suited for handling large, dense networks, where minimizing execution time is critical.

3.2 Barabási-Albert network

The Barabási-Albert (B-A) networks were also generated with 10,000 nodes, and with the average degree ⟨k⟩ increased from 10 to 140.

Similar to the ER networks, the Barabási-Albert networks were created using the NetworkX library.

```
m = average_degree // 2
G = nx.barabasi_albert_graph(nodes, m)
```

In this case, the parameter m determines the number of edges each new node connects to. This produces a scale-free network with a power-law degree distribution.

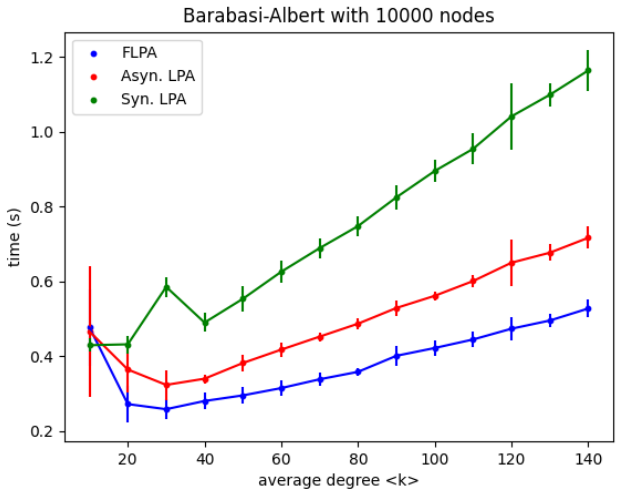


Figure 2: Execution times for synchronous LPA, asynchronous LPA, and FLPA on B-A networks with 10,000 nodes and varying average degree ⟨k⟩. Results represent the mean and standard deviation over 50 runs, reflecting the randomness inherent in the algorithms' processes.

The results, shown in Figure 2, indicate that FLPA consistently outperforms both LPA variants in terms of execution time even at lower average degrees. However, the distinction becomes more apparent as ⟨k⟩ increases, with FLPA showing a more significant advantage in denser networks. On the other hand, synchronous LPA becomes substantially slower as the average degree rises, while asynchronous LPA remains more efficient, but still lags behind FLPA in larger, denser networks. These findings suggest that FLPA is particularly advantageous for large-scale, high-density networks, where minimizing execution time becomes increasingly important.

3.3 Forest Fire network

The Forest Fire networks were generated with increasing number of nodes, ranging from 20,000 to 100,000. The networks were generated with a burning probability p_b set to 0.5, a parameter that

governs the likelihood of propagating "fire" during the graph's growth process.

The implementation of the Forest Fire network in Python utilized the following function:

```
def forest_fire_graph(n, burning_prob):
    G = nx.DiGraph()  # Directed graph
    G.add_node(0)  # Start with the first node

    for new_node in range(1, n):
        # Randomly select an ambassador node to start the "fire"
        ambassador = random.choice(list(G.nodes))
        G.add_node(new_node)
        G.add_edge(new_node, ambassador)

        # Propagate the fire with probability burning_prob
        frontier = [ambassador]
        while frontier:
            next_frontier = []
            for node in frontier:
                neighbors = list(G.neighbors(node))
                random.shuffle(neighbors)
                for neighbor in neighbors:
                    if random.random() < burning_prob:
                        G.add_edge(new_node, neighbor)
                        next_frontier.append(neighbor)
            frontier = next_frontier
    return G
```

This function builds a directed Forest Fire graph by iteratively adding nodes, connecting them to an initial ambassador, and propagating connections probabilistically according to p_b.

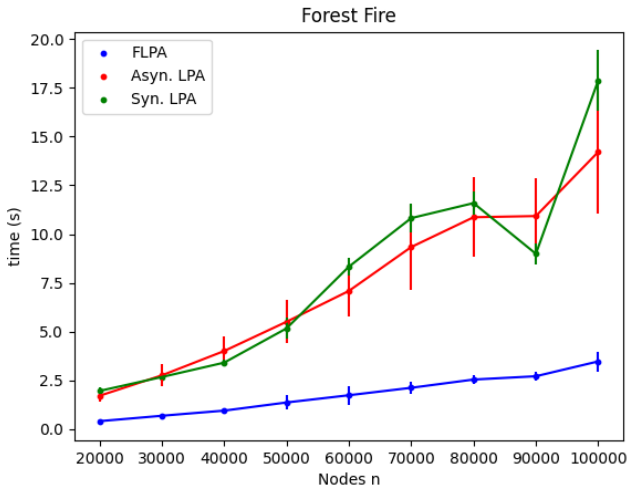


Figure 3: Execution times for synchronous LPA, asynchronous LPA, and FLPA on Forest Fire networks with increasing number of nodes from 20000 to 100000 and burning probability set to 0.5. Results represent the mean and standard deviation over 50 runs, reflecting the randomness inherent in the algorithms' processes.

The performance analysis, illustrated in Figure 3, highlights key differences in computational efficiency among the algorithms. FLPA emerges as significantly more efficient than both synchronous and asynchronous LPA across all tested network sizes, with its advantage becoming increasingly pronounced as the number of nodes grows.

Conversely, the computational costs of synchronous and asynchronous LPA are comparable, with neither method showing a clear efficiency advantage over the other. However, both approaches experience

a more substantial increase in execution time as the network size increases compared to FLPA. This trend underscores FLPA's scalability and suitability for handling larger and more complex networks, such as those generated by the Forest Fire model.

3.4 Stochastic Block Model network

The Stochastic Block Model (SBM) is commonly used to create synthetic networks with a clear community structure. It defines a set of blocks (partitions) and assigns nodes to these groups with probabilities governing edge creation both within and between groups. In this study, SBM networks were generated with 10000 nodes divided into 100 equal-sized groups, average degree fixed at 10 and increasing mixing parameter μ from 0.1 to 0.9. This last parameter controls the proportion of edges between communities relative to the edges within communities, with lower values of μ corresponding to stronger community structures and higher values representing more random and less discernible communities.

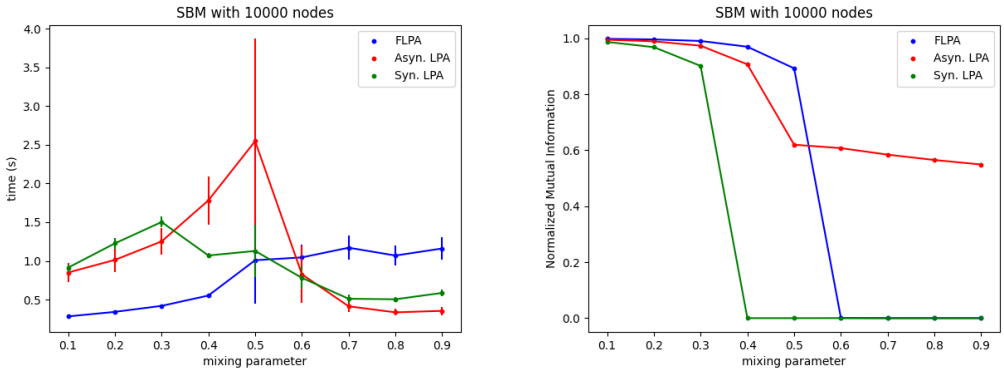
The implementation of SBM networks in Python utilized the following function:

```
def create_sbm(num_nodes, num_groups, avg_degree, mu):
    nodes_per_group = num_nodes // num_groups
    sizes = [nodes_per_group] * num_groups
    remainder = num_nodes % num_groups
    for i in range(remainder):
        sizes[i] += 1

    p_in = ((1 - mu) * avg_degree) / (nodes_per_group - 1)
    p_out = (mu * avg_degree) / (num_nodes - nodes_per_group)
    probs = np.full((num_groups, num_groups), p_out)
    np.fill_diagonal(probs, p_in)

    G = nx.stochastic_block_model(sizes, probs, seed=42)
    return G
```

This implementation calculates the probabilities of edge creation both within (p_{in}) and between (p_{out}) groups based on the average degree and mixing parameter.



(a) (b)

Figure 4: (a) represents th execution times for synchronous LPA, asynchronous LPA, and FLPA on SBM networks with increasing mixing parameter while (b) represents the NMI. Results represent the mean and standard deviation over 50 runs, reflecting the randomness inherent in the algorithms' processes.

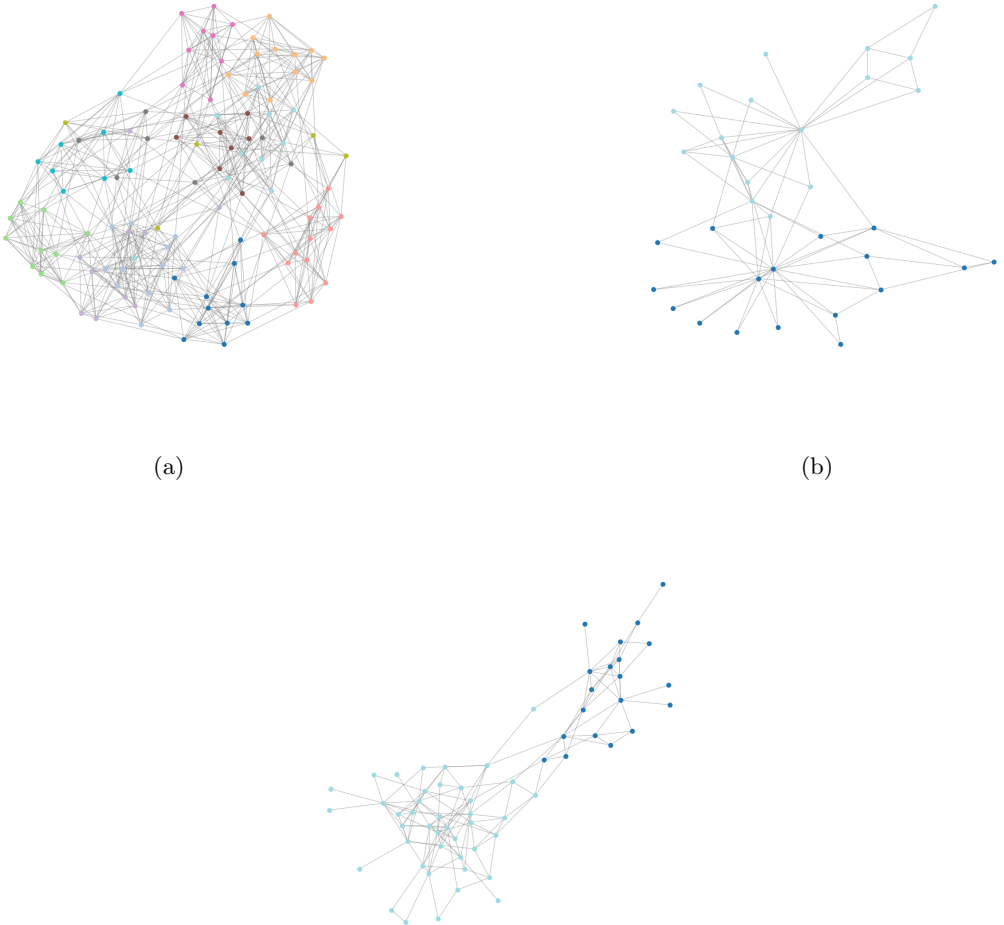
The results of this analysis are shown in the two plots 4a and 4b. Focusing on the left plot, we can observe that for $\mu < 0.5$, where communities are well-defined, FLPA is the most efficient, maintaining consistently low computation times, while asynchronous and synchronous LPA require more iterations to converge. For $\mu > 0.5$, as communities lose structure, FLPA's computation time increases, reflecting its effort to process less coherent partitions, whereas both LPA variants show reduced times as they

converge to trivial solutions faster.

The plot on the right shows the Normalized Mutual Information (NMI) for the SBM with increasing mixing parameters. For low values of μ, all three algorithms achieve high NMI, indicating that they effectively detect the predefined community structure. However, as μ increases, the performance of the algorithms diverges. The synchronous LPA experiences a rapid decline in NMI, starting from $\mu = 0.3$ and reaching zero as μ increases further. Asynchronous LPA starts to decrease earlier than FLPA, but it stabilizes at a NMI value around 0.6 even for higher μ values, indicating some degree of community structure remains. In contrast, FLPA shows the slowest decrease in NMI, since it remains effective at detecting communities until $\mu \geq 0.6$, after which it gradually approaches zero as expected when the network loses its cluster-like structure.

4 Analysis on empirical networks

4.1 Small empirical networks



(a) (b)

(c)

Figure 5: Graph diagrams of the football network (a), the karate network (b) and the dolphins network (c).

To qualitatively compare the performance of the three algorithms, we calculated the NMI on three small empirical networks with a known sociological division of nodes into communities. This approach allows us to evaluate the accuracy of the detected partitions by comparing them with the predefined clusters. In this case, the algorithms were not compared in terms of computational time since the networks are too small for the timing measurements to be informative. For each network, we provide the corresponding graph diagram, where the true clusters are visualized with different colors.

The three analyzed networks were taken from the network repository [3], and they include the karate network [5] (Figure 5b), a friendship network among members of a university karate club divided into two factions (34 nodes, 78 edges); The dolphins network [1] (Figure 5c), which represents social associations observed among dolphins off the coast of New Zealand, divided into two sociological groups (62 nodes, 159 edges); The football network (Figure 5a), representing games played between U.S. college football teams during the 2000 regular season, where each college belongs to one of twelve conferences (115 nodes, 616 edges).

The NMI results, plotted in the graph 6, show that the performance of the three algorithms does not differ significantly. As expected, from a qualitative perspective, FLPA is very similar to LPA, since both algorithms consistently identify partitions that align well with the known sociological divisions.

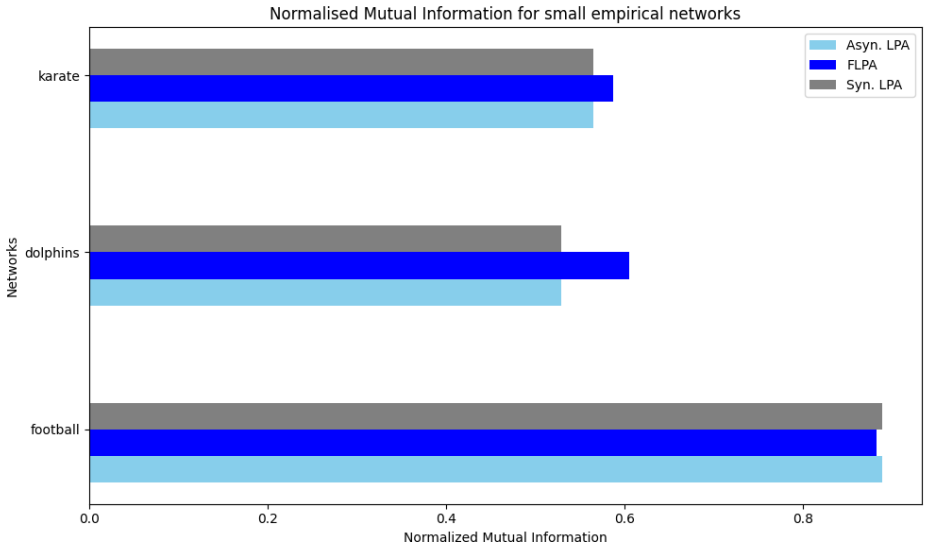


Figure 6: Normalized Mutual Information for synchronous LPA, asynchronous LPA, and FLPA on small empirical networks. Results represent the mean over 500 runs, reflecting the randomness inherent in the algorithms' processes.

4.2 Large empirical networks

We now discuss the results obtained from testing the algorithms on six large empirical networks (from [3]). These networks, characterized by different topologies and edge densities, allow us to evaluate the performance of the algorithms on real-world data from various domains.

The first network, com-dblp, represents a coauthorship network extracted from the DBLP computer science manuscript database in 2012. This is a one-mode projection derived from a bipartite graph of computer scientists and their publications, where nodes represent scientists, and edges indicate collaborations through shared publications. The second network, academia-edu, captures a snapshot of follower relationships among users of Academia.edu, a platform for sharing research papers, scraped in 2011. In this directed network, nodes represent users, and an edge from node i to node j signifies that user i follows user j. The third network, Google-plus, is a snapshot of user connections on Google+, collected in 2012. Here, nodes represent users, and a directed edge from i to j indicates that user i

added user j to their circle. The fourth network, Marvel-universe, is a collaboration network where two Marvel characters are linked if they appear together in the same comic book. The fifth network, pgp-strong, represents the strongly connected component of the Pretty-Good-Privacy (PGP) web of trust among users, collected in November 2009. Lastly, epinions-trust is a "who-trusts-whom" online social network from the consumer review platform Epinions.com. In this network, members can decide to "trust" other users, and these trust relationships are combined with review ratings to determine which reviews are prominently displayed to other users.

Network	Nodes	Edges	Algorithm	Modularity	Time (s)	Speedup
com-dblp	425 957	1 049 866	Asyn. LPA	0.62	113.82 ± 35.90	x3.4
			Syn. LPA	0.65	60.72 ± 1.93	x1.8
			FLPA	0.66	33.13 ± 1.03	
academia-edu	200 169	1 022 883	Asyn. LPA	0.30	50.76 ± 15.13	x4.7
			Syn. LPA	0.26	37.10 ± 0.28	x3.4
			FLPA	0.29	10.88 ± 0.96	
google-plus	211 187	1 143 411	Asyn. LPA	0.78	34.88 ± 7.46	x3.6
			Syn. LPA	0.78	36.40 ± 0.44	x3.8
			FLPA	0.80	9.58 ± 1.16	
marvel-universe	19 428	95 497	Asyn. LPA	0.61	2.22 ± 0.48	x4.1
			Syn. LPA	0.65	2.12 ± 0.04	x3.9
			FLPA	0.62	0.54 ± 0.04	
pgp-strong	39 796	197 150	Asyn. LPA	0.68	5.59± 1.43	x5
			Syn. LPA	0.71	4.71 ± 0.04	x4.3
			FLPA	0.70	1.10 ± 0.07	
epinions-trust	75 888	405 740	Asyn. LPA	0.07	6.40 ± 1.21	x3.7
			Syn. LPA	0.08	7.40 ± 0.05	x4.3
			FLPA	0.06	1.74 ± 0.11	

Table 1: Modularity and computational time for each algorithm in each network. Results represent the mean and standard deviation over 20 runs, reflecting the randomness inherent in the algorithms' processes.

Modularity is a metric used to assess the quality of the partitioning of the network into communities. It compares the density of edges within communities to the density of edges between communities. A higher modularity indicates a better division of the network into cohesive communities. To implement it we used the function `nx.community.modularity(G, communities)` from the NetworkX library.

The results, including computational time and modularity scores, are reported in the table 1. Notably, the modularity values achieved by the three algorithms are very similar for all networks, indicating that they produce comparable quality in terms of community detection.

However, despite achieving comparable modularity values, FLPA consistently outperforms the other two algorithms in terms of computational time. On the com-dblp network, FLPA completes in 33.13 ± 1.03 seconds, significantly faster than Synchronous LPA (60.72 ± 1.93 seconds) and Asynchronous LPA (113.82 ± 35.90 seconds). This trend is evident across all networks, such as on the academia-edu network, where FLPA is approximately four times faster than Asynchronous LPA while maintaining similar modularity.

In terms of speedup, FLPA demonstrates a clear advantage, achieving improvements ranging from 3.4× to 4.7× compared to Asynchronous LPA. This highlights that, at comparable modularity levels, FLPA is computationally more efficient, making it an attractive option for community detection in large networks.

References

[1] D. Lusseau; K. Schneider; O. J. Boisseau; P. Haase; E. Slooten; S. M. Dawson. "The Bottlenose Dolphin Community of Doubtful Sound Features a Large Proportion of Long-Lasting Associations". In: *Behavioral Ecology and Sociobiology, Vol. 54, No. 4, pp. 396-405* (2003). URL: http://dx.doi.org/10.1007/s00265-003-0651-y.

[2] Matilde Grassi. *Community detection algorithms: FLPA and LPA - A Complex Network Project*. 2024. URL: https://github.com/MatildeGrassi/Comparative-Analysis-of-Community-Detection-Algorithms-FLPA-and-LPA.git.

[3] Tiago P. Peixoto. *The Netzschleuder network catalogue and repository*. 2020. URL: https://networks.skewed.de/.

[4] Vincent A. Traag and Lovro Šubelj. "Large network community detection by fast label propagation". In: *Scientific Reports* 13.1 (Feb. 2023), p. 2701. ISSN: 2045-2322. DOI: 10.1038/s41598-023-29610-z. URL: https://doi.org/10.1038/s41598-023-29610-z.

[5] W. W. Zachary. "An information flow model for conflict and fission in small groups". In: *Journal of Anthropological Research 33, 452-473* (1977). URL: https://www.jstor.org/stable/3629752.