A Survey of Community Detection: Algorithms, Applications, and Beyond!

Jake Jasmer*
Carleton College

Tony Ni†
Carleton College

Aidan Roessler‡
Carleton College

Yang Tan§
Carleton College

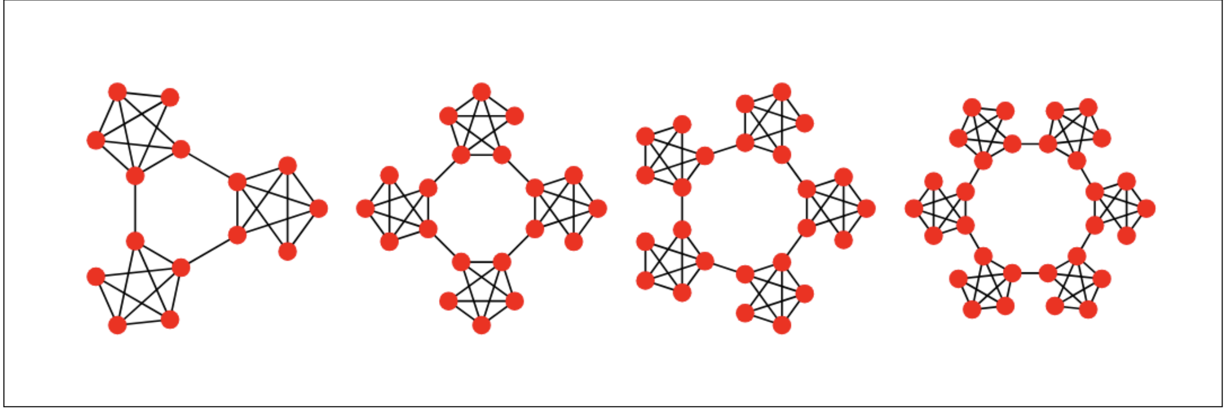

Figure 1: Caveman graphs of different sizes [30]

Abstract

The community detection problem is a common graph problem that involves detecting clusters of nodes in a graph that maximize some measure of quality Q. To gain insight into the problem as a whole, we studied three community detection algorithms with different approaches to finding communities: the Girvan-Newman algorithm, the Louvain method algorithm, and the Basic Variable Neighborhood Search algorithm. We ran these algorithms on synthetically generated graphs and graphs constructed from actual data from a variety of disciplines and compared their performance on a suite of metrics. We find that for our graphs that represent real-world data, there is no universally optimal community detection algorithm, as differences in graph structure and size and the varying definitions of real-world communities across applications significantly impact performance, leading no one algorithm to perform consistently the best across all of the metrics we collected.

1 Introduction

Graphs are one of the most fundamental data structures in computer science. A graph $G = (V,E)$ consists of a set of vertices/nodes V and a set of edges E, where each edge is an unordered pair of two vertices. Graphs can simply model many complex structures across various fields.

One such application is community detection. While there are many variations of the community detection problem, they all aim to identify communities of vertices within a given graph. Communities are typically defined as "a group of nodes having similar affiliations different to rest of the [graph]" [31]. In simpler terms, a community is a group of vertices where the edge connections between members are stronger than the connections to vertices outside the group. The specific way the community detection problem is formulated can vary across applications, as we discovered. A general formal definition could be as follows (credit to Layla Oesper):

Input: A graph $G = (V,E)$ (which could be weighted or unweighted) and an objective function Q that measures the quality of a partitioning of vertices.

Output: A partitioning of V into disjoint subsets $\mathscr{C} = \{C_1, C_2, ..., C_k\}$ (where $\bigcup_{i=1}^{k} C_i = V$) such that $Q(\mathscr{C})$ is maximized/minimized over all such partitions.

One such quality measure to use for Q is modularity. It measures whether the weight of the edges between the nodes within the same community is stronger than if the edges were randomly distributed. A more detailed explanation can be found in Sec. 2.3.1.

Due to its effectiveness in predicting the strength of communities, modularity serves as the metric that many community detection algorithms attempt to maximize, including all of the ones we implemented.

Modularity is the metric of choice for many algorithms in both classical categories of community detection approaches we studied: agglomerative methods and divisive methods.

In agglomerative methods, "[communities] are combined iteratively (bottom-up)" if they have a high similarity score [17]. This method begins with each vertex as its own community and then merges communities that result in the highest positive change in the Q value. The advantage of using such methods is that they are able to effectively find smaller communities, as they can focus on local structures and refine the community boundaries gradually. However, the disadvantage is that there is little recourse for correcting an incorrect community merge, and it will propagate up into the final communities, leading to a suboptimal solution that is difficult to escape.

In contrast, divisive methods use a top-down approach and work by removing edges in a given graph that are most likely to separate distinct communities [17]. These types of algorithms quantitatively decide which edge is most likely to separate communities,

*e-mail: jasmerj@carleton.edu
†e-mail: nit@carleton.edu
‡e-mail: roesslera@carleton.edu
§e-mail: tany2@carleton.edu

often by calculating a betweenness centrality value for an edge, and then removing that edge. Edge betweenness centrality is defined as the number of shortest paths between any two given nodes in a graph that run through an edge [1]. Removing edges with high betweenness centrality isolates vertex subsets with stronger interconnections, making it a useful strategy for detecting communities, which are characterized by dense internal connections and sparse external links. Since the metric they calculate for each edge is the sole way that these algorithms choose which edge to remove, choosing an effective method to determine which edges to cut is important for the quality of the communities that these algorithms return.

Community detection has a wide variety of applications because revealing hidden structures in large networks (which can easily be represented with a graph) has become extremely valuable in a world increasingly built on large complex networks. This usefulness also extends to the natural world, particularly in biological applications, as many networks naturally exist within these systems.

In the biological world, Protein-Protein Interactions (PPI) can be modeled by graphs and are a strong candidate for community detection because they exhibit clear communities and in some cases expert defined communities for commonly studied organisms. In a PPI network, a vertex represents a gene or a protein coded by a gene and an edge represents the interaction between two genes/proteins. Finding communities in such a network helps to identify proteins that work together to achieve a biological function. These communities may correspond to molecular complexes or specific segments of biological pathways. Identifying these communities has several use cases. For instance, proteins of unknown function or those with poor annotations can be assigned to a potential role by identifying the communities they belong to. Also, identifying functional modules can help design drug targets for disease treatments, such as identifying the central protein in cancer-related communities that can be targeted to modulate a disease pathway. Since PPI networks often map the relationships between all the proteins an organism contains, the graphs that represent them are often very large. For example, the graph representation of the PPI network for *Saccharomyces cerevisiae* (Yeast) has 6,532 nodes and 229,696 edges and 20,644 nodes and 241,008 edges for *Homo sapiens* (Humans) [24].

Another biological application that is a strong candidate for community detection is finding groups of neurons with similar functions. Brains are made up of complex arrangements of individual neurons structured similarly to a graph. Each neuron has three main parts: the dendrites where incoming signals are received, the soma or cell body where decisions are made regarding which signals to send out, and the axon where the signal is sent out. Each neuron's axon connects to the dendrites of neighboring neurons, mirroring how a graph's edges work. The axon and dendrites together make directed edges, while the cell bodies are the nodes. Finding communities helps to identify neuronal circuits with specific functionality [10, 12, 13]. Higher modularity in graph representations of networks of neurons has also been found to correspond with improved memory and increased neuroplasticity, which is the ability of a group of neurons to rewire themselves in order to function differently [13]. It should be noted though that neuroplasticity is highly correlated with age. Additionally, modularity seems to be altered in people with mental disorders like schizophrenia and depression. Overall, representing the brain as a graph with communities, calculating its modularity, and running modularity-based algorithms are valuable techniques for gaining deeper insights into the brain and its functions [13].

Widening our scope, another highly relevant use case for community detection is urban movement. In these contexts, graphs represent real-world transportation networks, with nodes typically representing locations, such as transit stations, an individual's location at a given time, or neighborhoods. Edges represent connections between these locations, like transit lines, an individual's movement, or the flow of traffic. A key idea in modeling urban movement involves trajectories, which are sequences of locations linked by edges over time. Each node in the graph, referred to as a stamp node, corresponds to a specific location at a particular time, while the edges represent the movement or flow between these locations. The goal of community detection in these graphs is to identify zones or clusters where strong connectivity or frequent interactions occur, such as commuter hubs, neighborhoods with high travel demand, or areas exhibiting similar transportation patterns. Identifying these communities offers several applications, including optimizing public transit routes, improving traffic management, and designing more efficient ride-sharing systems. For example, by detecting clusters of heavily connected areas, urban planners can allocate resources more effectively or create dedicated transit corridors. Given that these graphs often represent entire cities or regions, they are typically very large, with thousands of nodes and millions of edges. Therefore, analyzing such graphs requires scalable algorithms, like the Louvain method [21,22], to handle their complexity and size.

It is clear from these examples that the community detection problem spans a wide variety of disciplines, for which the relevant graphs vary greatly. These differences suggest there is no one community detection algorithm that works best for every discipline and real-world application. To gain a better understanding of their strengths and weaknesses, we implemented community detection algorithms representing a variety of approaches.

2 METHODS

2.1 Algorithms

To test if we could find meaningful and useful communities in our selected datasets, we implemented a famous agglomerative community detection algorithm, the Louvain Method, and a famous divisive one, the Girvan-Newman algorithm. We also decided to implement the Basic Variable Neighborhood Search (BVNS) algorithm as it did not cleanly fall into either category.

2.1.1 Girvan-Newman Algorithm

The first of the algorithms we implemented was a modified version of the Girvan-Newman algorithm as described in the original 2002 paper "Community structure in social and biological networks" [14]. It is arguably the most popular divisive algorithm. The high level steps of our version of the algorithm, are as follows:

1. Calculate the betweenness for all edges in the graph.
2. Remove the edge with the highest betweenness.
3. Store a list of set representations of the current connected components (communities) and calculate its modularity.
4. Recalculate betweennesses for all edges affected by the removal.
5. Repeat from step 2 until no edges remain.
6. Return the iteration of communities with the highest modularity.

While steps 2, 5, and 6 are relatively straight forward, calculating the betweenness for edges in steps 1 and 4 contributes significantly to technical complexity and the runtime of the algorithm. This is because calculating edge betweenness involves using the method originally proposed by Ulrik Brandes in his paper "A faster algorithm for betweenness centrality" [9]. This method iterates over every node and for every iteration runs Breadth First Search (BFS) for unweighted graphs and Dijkstra's Algorithm for weighted graphs to find every shortest path between every pair of nodes in the graph.

Once it has done this it then stores the proportion of all of those shortest paths that pass through each edge in a dictionary where edges are the keys and a proportion are the values.

Our implementation of BFS has a runtime of $O(|V|+|E|)$, matching its conventional runtime. Our implementation of Dijkstra's algorithm has a runtime of $O((|V|+|E|)\log|V|)$, since it is implemented in the conventional way using a priority queue implemented with a heap. Since our implementation of Brandes' approach performs BFS or Dijkstra's Algorithm for each node in the graph, its runtime is $O(|V|(|V+|E|) = O(|V|^2+|V||E|) = O(|V||E|)$ for unweighted graphs and $O(|V|((|V|+|E|)\log|V|)) = O(|V||E|\log|V|)$ for weighted graphs. This matches the proven runtimes of Brandes approaches for both undirected and directed graphs from the paper in which they were originally outlined [9].

Since we calculate the edge betweenness for each of the $|E|$ edges we remove, the final runtime of our implementation is $O(|E|(|V||E|)) = O(|V||E|^2)$ for unweighted graphs which matches Girvan and Newman's original runtime [14]. It is $O(|E|(|V||E|\log|V|)) = O(|E|^2|V|\log|V|)$ for weighted graphs.

While this runtime is polynomial, as Girvan and Newman note in their original paper, this algorithm is not very practical for dense graphs or very large graphs and performs best on sparse graphs [14]. Assuming we have a sparse unweighted graph, where $|V|=|E|$, the runtime of the algorithm is roughly equal to $O(|V|^3)$ or if we set $n=|V|$, $O(n^3)$.

In our implementation, we represented communities as a list of sets, where each set contained integers or strings corresponding to the vertices in that community. This allowed for quick insertions and deletions of nodes in the communities and ensured that each community was made up of distinct nodes.

As stated previously, our implementation is a modified version of Girvan-Newman. This is because in their original paper outlining it, Girvan and Newman did not explicitly state that their algorithm maximizes modularity [14]. However, most modern approaches we researched do, so we decided to do the same in our implementation [3]. Additionally, we believe modularity is the most concrete metric for generating accurate communities because it indicates whether communities exhibit significantly stronger internal connections than would occur by chance, which approximates the accuracy of communities well.

2.1.2 Louvain Algorithm

The second algorithm we implemented was the Louvain method, which is a popular hierarchical, greedy optimization algorithm. It aims to optimize the modularity of a network by iteratively assigning nodes to communities and merging them in order to maximize modularity. The high-level steps of the Louvain algorithm are as follows [8]:

1. Assign each node to its own community (initial partition).
2. For each node, consider moving it to a neighboring community.
3. Calculate the modularity of the current community configuration. If modularity has increased, keep the new community assignment; otherwise, retain the original one.
4. Repeat steps 2-3 for all nodes until no further modularity improvements are found.
5. Merge communities to form a new graph where each community is treated as a single node. It is possible to run Louvain on this graph again to obtain a better result.
6. Return the final community structure.

Steps 2 and 3, which involve node reassignment and modularity optimization, are the key components of the algorithm. As we mentioned in the Introduction, the modularity calculation determines the quality of the community structure by comparing the fraction of edges within a community to the expected fraction if the edges were randomly distributed. Once all nodes are reassigned or determined to be optimally placed, the communities are merged to form a new graph where each community becomes a supernode. The process is repeated iteratively on this new graph, allowing for a hierarchical detection of communities.

The time complexity of the Louvain method is dominated by the modularity optimization procedure. For each node, we check each of its neighbors, leading to an iteration over the graph's edges. In the worst case, the time complexity per node is proportional to the degree of the node, and the total complexity is typically written as $O(|V|\log|V|)$.

As with many greedy algorithms, the Louvain method is prone to getting trapped in local optima. This means that, while it may find a division that maximizes modularity locally, it might not always find the global optimum. Thus, in some cases, the final communities returned may not represent the communities with a global maximum modularity.

Despite this limitation, the Louvain method remains one of the most effective and practical algorithms for community detection, especially for large and complex networks. Its scalability and simplicity make it widely used in practice. It is particularly well-suited for sparse graphs, where the modularity optimization process can be efficiently performed. However, for very large and dense graphs, the algorithm may become more susceptible to finding local optima.

2.1.3 Basic Variable Neighborhood Search Algorithm

Basic variable neighborhood search (BVNS) does not fit neatly into the aforementioned categories of agglomerative and divisive algorithms. It instead uses randomization to search the space around the current grouping of communities. BVNS starts with a random grouping of communities, and then iterates over the following steps [19]:

1. Shaking
2. Local search
3. Comparison

Shaking refers to randomly shifting around nodes. This is done for k nodes from 1 to some integer, k_{max}, resulting in k shaken groupings [19]. The grouping with the largest Q is then used for local search. Local search involves taking every node and moving it to every other possible community. It is essentially an exhaustive exploration at a distance of 1 [19]. The comparison step refers to comparing the best grouping from local search to the original grouping from before the shaking step, best meaning it has the largest Q. The best grouping out of those two possibilities will become the original grouping for future iterations. Since BVNS is an iterative algorithm and lacks a definitive stopping point, we stop after 100 iterations, adhering to the recommendation in "Variable Neighborhood Search Approach to Community Detection Problem" by Jovanović et al. [19]. Additionally, the number of starting communities significantly contributes to the end result, so we run this algorithm for $2-\sqrt{|V|}$ starting communities and choose the best result. $\sqrt{|V|}$ is an arbitrary stopping point that was chosen for its ability to easily scale the number and size of communities with the size of the graph.

While BVNS can achieve modularity results that are comparable to the other algorithms discussed, it is less than ideal when it comes to time complexity. BVNS's time complexity is dominated by the local search step, which checks moves for every node: $O(|V|)$, and

then checks the modularity for each of those moves: $O(|E|)$, resulting in $O(|V| \cdot |E|)$. We also need to consider how many moves we make for each node, which is equal to the number of communities minus 1.

As stated above, our implementation starts with anywhere between 2 and $\sqrt{|V|}$ communities, running one time for each possible starting number of communities. Thus, it runs $\sqrt{|V|} - 2$ times. For each time BVNS runs, it starts with $\sqrt{|V|}$ communities, then $\sqrt{|V|} - 1$, $\sqrt{|V|} - 2$, ..., 2. Since the time complexity depends on the number of communities, which is best approximated by the number of starting communities, we want to add a term for the number of starting communities. However, this number is different every time, so we have to take the average case, which is halfway between $\sqrt{|V|}$ and 2: $(\sqrt{|V|} - 2)/2$. Once we add this term, we get $O(|V|^{3/2} \cdot |E|)$. Now, we can finally consider the number of times this algorithm is run: $\sqrt{|V|} - 2$. So, adding it in we get: $O(|V|^2 \cdot |E|)$.

The number of iterations is also considered, but it is a constant so it is left out of the equation. However, our use of 100 iterations is large enough it becomes relevant for the runtime, even if not included in the formal definition of time complexity. When considering this coefficient, we get $O((|V|^2 \cdot |E|) \cdot 100)$ in our case.

We can make this better by creating a BVNS-specific modularity function that only calculates the changes from a single node moving communities, which is effectively constant time if the graph is sparse. In this case, we need only calculate the change for the degree of the node of interest, which will be small for a sparse graph. This changes the runtime of modularity calculation in the local search step from $O(|E|)$ to $O(1)$, resulting in a $O(|V|^2 \cdot i)$ algorithm. Although this approach is better than the alternative, it is still remarkably slow.

BVNS has efficient memory usage, needing to only store the best set of communities and the new one to compare to at any given point. Since the number of communities depends on the number of nodes, it has a space complexity of $O(|V|)$.

2.2 Datasets

2.2.1 Baseline Datasets

To verify the accuracy of our implementations, we ran them on a number of baseline simple graphs that represent real world scenarios, but had a small node, edge, and community counts so that we could verify their accuracy while being able to quickly collect our suite of metrics for each.

One such baseline we used to test our implementations was a graph Girvan and Newman tested their original algorithm on, Zachary's Karate Club as described in the paper "An Information Flow Model for Conflict and Fission in Small Groups" by Wayne W. Zachary [32]. We chose this classic example in the community detection field because it has defined ground-truth communities, and only two of them. In his original paper, Zachary details how a university Karate club split into two groups over a conflict between two members [32]. He uses a graph to represent the two separate groups where vertices are individuals and edges are the strength of social relationships between individuals. We also selected this to use to test our implementations on as it was easily available through the Network X library [4] that we used to aid in our implementations.

Another baseline graph we checked our algorithms against was a graph representation of the 2000 Division I season in college football which Girvan and Newman ran on their original implementation [14]. In this graph, nodes are college football teams and edges represent games between the teams the nodes represent. We chose this graph as a baseline because if our algorithms are working correctly, they should be able to somewhat accurately identify the college football conferences as communities in the graph, since the 8 - 12 teams in each conference play each other more than teams outside of their conference.

In order to further test our implementations on simple baseline graphs with easily identifiable communities, we ran them on a series of synthetic datasets.

2.2.2 Synthetic Datasets

We used two different types of graphs for our synthetic datasets: ring of cliques and stochastic block model [16]. As shown in Figure 2, a ring of cliques graph consists of k cliques, each of size n, connected by single links. Each clique forms a complete graph. Our ring of cliques graphs serve as examples of "trivial" community detection problems to test the performance of our implemented algorithms.

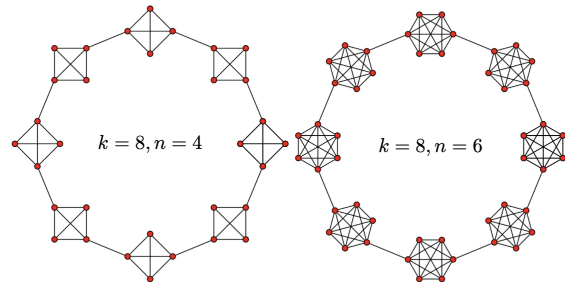

Figure 2: Ring of Cliques. k denotes number of cliques and n denotes size of cliques. Adapted from [18].

As shown in Figure 3, the stochastic block model (SBM) partitions nodes into n blocks of specified sizes, connecting node pairs independently based on a probability matrix that controls intra-community density p and inter-community density q. The SBM-generated graphs serve as challenging community detection scenarios that better capture the complexity of real world datasets.

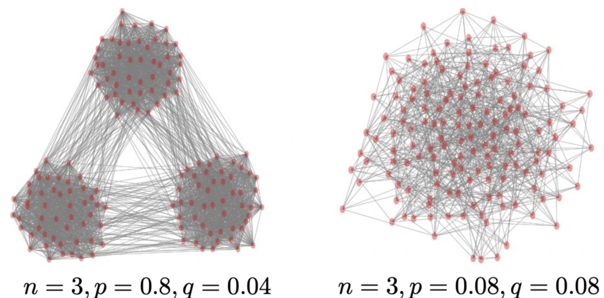

Figure 3: Stochastic Block Model. n denotes number of communities, p denotes intra-community density, and q denotes inter-community density. Adapted from [29].

2.2.3 Urban Movement Dataset

To apply our community detection algorithms to an urban movement dataset, it must include a defined geographical area and a group of moving objects with their recorded timestamps over a given period. While several real-world datasets fit these criteria, such as GPS traces of pet cats [2] or Microsoft Geolife GPS [6], they lack a well-defined ground truth for community structures, making it difficult to evaluate the accuracy of our results.

To address this limitation, we chose to replicate a simulation from "Detecting spatial community structure in movements" by Guo et al. [15]. In this simulation, we define a rectangular area

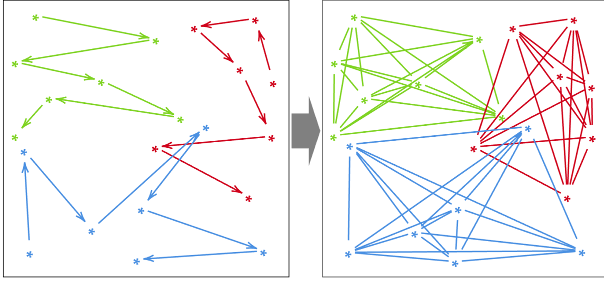

Figure 4: Illustration of the graph construction process, where each trajectory forms a complete subgraph.

partitioned into several subregions, which serve as the ground truth communities. A group of moving agents is then simulated to traverse the space, generating trajectories that form activity patterns within these subregions. We applied our algorithms to this simulated dataset, compared the detected communities to the known ground truth, and analyzed the accuracy and effectiveness of the results. In the following paragraphs, we provide a detailed explanation of the simulation methodology.

The simulation generates a set of trajectories, $T_1, T_2, \dots, T_n$, each consisting of m locations. A trajectory $T_i = \{s_{ij}\}$ is a sequence of m points, where each point s_{ij} represents a specific location. Let $S = \{s_{ij}\}$, with $|S| = n * m$, be the set of unique locations from all trajectories.

A graph $G = (V, E)$ is constructed using these $n * m$ locations as vertices. Each trajectory T_i forms a complete subgraph, connecting every pair of distinct locations s_{ix}, s_{iy} with an edge. This results in G being composed of n disconnected cliques, where each clique represents a single trajectory. Figure 4 illustrates this process, though colors in the figure are used for visualization only and do not indicate trajectory distinctions in G.

To improve computational efficiency, especially in large datasets with many location points, an initial spatial clustering step is performed. For instance, in a dataset of taxi trips, numerous drop-off and pick-up points might exist within a small radius around a subway station. Instead of treating each point as unique, clustering them together reduces redundancy while maintaining analytical accuracy. A simple k-means clustering method, with k representing the number of clusters, is used to aggregate similar locations. The original graph G is then transformed into $G' = (V', E')$, where $|V'| = k$, replacing individual locations with cluster centers as nodes.

In our simulation, we set k to 0.4 times the total number of original vertices, as this consistently yielded clearer results.

To establish spatial contiguity among points, Thiessen polygons (Voronoi diagrams) are generated around each vertex in V', enclosing every location within a polygon. A community detection algorithm is then applied to G' to partition V' into communities. The sum of the areas of the corresponding polygons defines the activity areas of the trajectories.

2.2.4 PPI Dataset

We also ran our algorithms to detect communities in the *Saccharomyces cerevisiae* (yeast) PPI network, which has “ground-truth” communities based on laboratory biological research [23, 27]. As shown in Figure 5, a vertex represents a protein and an edge represents the interaction between two proteins. Community detection identifies proteins that function together, which corresponds to protein complexes or segments of biological pathways. This can assist in the annotation of protein functions and in the discovery of drug targets.

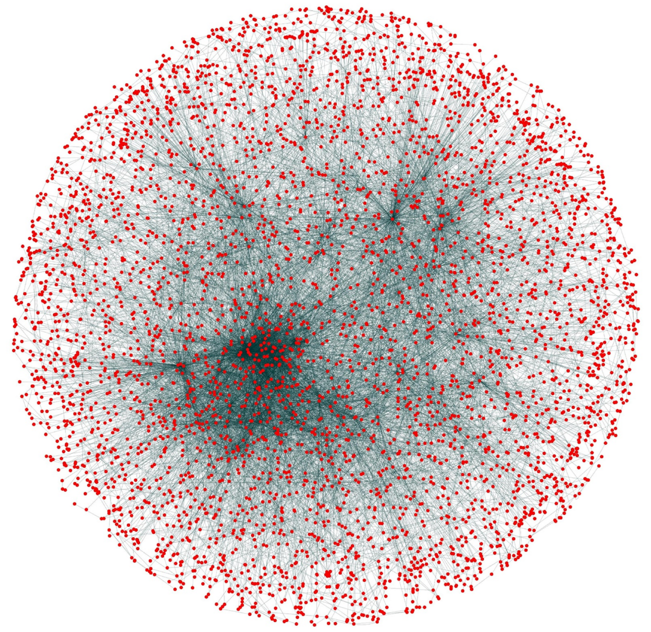

Figure 5: Yeast PPI network. Adapted from [26].

2.2.5 Neuronal Connectivity Dataset

Another application of community detection algorithms we considered was neuronal connectivity, specifically using a dataset of *C. elegans* neurons from “Comprehensive analysis of the C. elegans connectome reveals novel circuits and functions of previously unstudied neurons” by Scott W. Emmons [12]. This data comes from figure 1 in Emmons’ paper, but it has been updated so our results have slightly fewer communities. In this dataset, vertices represent neurons and edges represent synapses. Community detection identifies neuronal circuits with specific functionality.

It is important to note that neurons have a specific direction, so this graph is best defined as directed instead of undirected. We use the spectral method of Leicht and Newman to account for this change, following the methodology of Emmons [12, 20]. The main difference lies in the directed modularity calculation, explained in Sec. 2.3.1. Additionally, since our implementations only run on undirected graphs, we have converted the dataset to be undirected. This is done by combining edges that go both ways between pairs of nodes into one edge with the sum of their weights, and then by converting the one-way directed edges into undirected edges. But, we still use the directed version for modularity, which is possible because the nodes do not change. This allows for a comparison between our implementations and the spectral method.

2.3 Evaluation Metrics

We used the following metrics to evaluate the communities our algorithms detected. Modularity was collected for each dataset we ran our algorithms on. Effective conductance and adjusted rand index were collected in our experiments running our algorithms on our synthetic data. For our PPI dataset, we collected an additional set of metrics standard in the PPI research community [11].

2.3.1 Modularity

As mentioned in our introduction, modularity is a metric that many algorithms optimize for, including all of the algorithms we implemented. This is because it serves as a good reflection of the intrinsic quality of the communities generated. Thus, we decided to collect

it to use it as a "source of truth" metric across all of our implementations.

The exact definition of modularity varies, but the definition we are using for our purposes is the following:

$$Q = \frac{1}{2m} \sum_{u,v} \left[A_{uv} - \frac{k_u k_v}{2m}\right] \delta(c_u, c_v) \quad [20]$$

Where:

- $2m$ = the total number of half edges
- A_{uv} = the actual value of the edge weight for the edge (u,v) (is 0 if no such edge exists).
- $\frac{k_u k_v}{2m}$ = the expected weight for edge (u,v) (probability of two half edges being connected). k_u is the degree of node u and k_v is the degree of node v.
- $\delta(c_u, c_v)$ = the Kronecker delta function. Evaluates to 1 if vertex u and v are in the same community and 0 otherwise.

Essentially, calculating modularity involves summing the actual edge weights of all edges in the graph only in the same communities minus the sum of the expected weights of those edges, which allows us to compute whether the edge weights within communities are stronger than what would be expected by chance. However, due to the normalization factor $\frac{1}{2m}$, the range of Q is $[-1,1]$. A Q value of 0 indicates that the community structure is no better than random, meaning that the total weight of the edges within the communities is comparable to what would be expected by chance if we drew edges between nodes at random by turning all edges into half edges and randomly connecting half edges. Half edges refer to the individual endpoints of each edge. In other words, each edge is made up of two half edges. Positive modularity values suggest a stronger community structure than random, while negative values indicate a weaker structure than random.

For directed graphs, this equation changes slightly to account for in and out half edges. k_u becomes k_u^{out} and k_v becomes k_v^{in}. Additionally, since we are no longer using half edges we only need to use m instead of $2m$. The full directed modularity equation is as follows, all variables are the same as the normal equation:

$$Q = \frac{1}{m} \sum_{u,v} \left[A_{uv} - \frac{k_u^{out} k_v^{in}}{m}\right] \delta(c_u, c_v) \quad [20,25]$$

2.3.2 Effective Conductance (EC)

We employed EC to measure the fraction of edges crossing between communities relative to the total connections within the smaller community, averaged over all community pairs. Lower conductance means more separated communities. In mathematical terms, let $G = (V,E)$ be a graph with a partition $\mathscr{C} = \{C_1, C_2, \ldots, C_k\}$ into k communities. The EC is defined as:

$$\text{effective-conductance}(\mathscr{C}) = \frac{1}{|\mathscr{P}|} \sum_{(C_i,C_j)\in\mathscr{P}} \text{conductance}(G, C_i, C_j)$$

where:

- $\mathscr{P} = \{(C_i,C_j) \mid C_i, C_j \in \mathscr{C}, i < j\}$ is the set of all community pairs.
- $\text{conductance}(G, C_i, C_j)$ is the conductance between communities C_i and C_j.

The conductance between two sets S and T is given by:

$$\text{conductance}(S,T) = \frac{|\text{cut}(S,T)|}{\min(\text{vol}(S), \text{vol}(T))}$$

where:

- $|\text{cut}(S,T)|$ is the number (or total weight) of edges crossing between sets S and T.
- $\text{vol}(S) = \sum_{v\in S} d(v)$ is the sum of degrees of nodes in S.
- $\text{vol}(T) = \sum_{v\in T} d(v)$ is the sum of degrees of nodes in T.

2.3.3 Adjusted Rand Index (ARI)

We used ARI to quantify the similarity between two partitions of by comparing the assignments of data points to communities. Higher ARI means better alignment with ground truth. Mathematically, the ARI is computed as:

$$\text{ARI} = \frac{\sum_{ij}\binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}}$$

where:

- n_{ij} is the number of elements common to cluster i in the ground truth and cluster j in the predicted partition.
- a_i is the sum of elements in ground truth cluster i.
- b_j is the sum of elements in predicted cluster j.
- n is the total number of elements.

2.3.4 PPI-Specific Metrics

Let

$$\mathscr{C}_p = \{c_1, c_2, \ldots, c_{N_p}\}$$

be the set of predicted communities and

$$\mathscr{C}_r = \{c'_1, c'_2, \ldots, c'_{N_r}\}$$

be the set of reference (ground-truth) communities. For any predicted community $c \in \mathscr{C}_p$ and any reference community $c' \in \mathscr{C}_r$, define the overlap ratio as:

$$O(c,c') = \frac{|c \cap c'|}{|c| + |c'|}$$

where $|c|$ denotes the number of nodes in community c.

Given an overlap threshold θ (e.g., $\theta = 0.3$), we define the following classification metrics:

True Positive (TP) for Predicted Communities

A predicted community $c \in \mathscr{C}_p$ is considered a true positive if there exists at least one reference community $c' \in \mathscr{C}_r$ such that:

$$O(c,c') > \theta.$$

Thus, the total number of true positives among the predicted communities is:

$$\text{TP}_{\text{pred}} = \left|\left\{c \in \mathscr{C}_p \,\middle|\, \exists c' \in \mathscr{C}_r \text{ such that } O(c,c') > \theta\right\}\right|.$$

This counts the number of predicted communities that sufficiently match at least one reference community. Higher values indicate that more of the algorithm's predictions correspond to real communities.

False Positive (FP) for Predicted Communities

A predicted community that does not have any matching reference community (i.e., the overlap with every $c' \in \mathscr{C}_r$ is at most θ) is considered a false positive:

$$\text{FP} = |\mathscr{C}_p| - \text{TP}_{\text{pred}}.$$

This counts the number of predicted communities that do not match any reference community. Lower values indicate fewer erroneous predictions by the algorithm.

True Positive (TP) for Reference Communities
A reference community $c' \in \mathscr{C}_r$ is considered a true positive if there exists at least one predicted community $c \in \mathscr{C}_p$ such that:

$$O(c, c') > \theta.$$

Thus, the number of true positives for the reference set is:

$$\text{TP}_{\text{ref}} = \left| \left\{ c' \in \mathscr{C}_r \,\middle|\, \exists c \in \mathscr{C}_p \text{ such that } O(c, c') > \theta \right\} \right|.$$

This counts the number of reference communities that are successfully identified by at least one predicted community. Higher values indicate better coverage of the ground truth.

False Negative (FN) for Reference Communities
A reference community that has no matching predicted community is considered a false negative:

$$\text{FN} = |\mathscr{C}_r| - \text{TP}_{\text{ref}}.$$

This counts the number of reference communities that were not identified by any predicted community. Lower values indicate fewer missed communities by the algorithm.

Precision

$$P = \frac{\text{TP}_{\text{pred}}}{\text{TP}_{\text{pred}} + \text{FP}} = \frac{\text{TP}_{\text{pred}}}{|\mathscr{C}_p|}$$

Precision measures the fraction of predicted communities that match reference communities. It ranges from 0 to 1, with higher values indicating better prediction quality. A high precision means that most of the algorithm's predictions correspond to real communities, with few false alarms.

Recall

$$R = \frac{\text{TP}_{\text{ref}}}{\text{TP}_{\text{ref}} + \text{FN}} = \frac{\text{TP}_{\text{ref}}}{|\mathscr{C}_r|}$$

Recall measures the fraction of reference communities that were successfully identified. It ranges from 0 to 1, with higher values indicating better completeness. A high recall means that the algorithm successfully identifies most of the real communities in the network.

F-measure (F1-score)

$$F = 2 \times \frac{P \times R}{P + R}$$

This is the harmonic mean of Precision (P) and Recall (R). F-measure ranges from 0 to 1, with higher values indicating a better balance between precision and recall. It provides a single metric that balances the trade-off between finding all real communities and avoiding false predictions.

Accuracy

$$\text{Accuracy} = \sqrt{P \times R}$$

This is the geometric mean of Precision and Recall. Accuracy ranges from 0 to 1, with higher values indicating better overall performance. Unlike the arithmetic mean, this geometric mean is more sensitive to imbalances between precision and recall, penalizing algorithms that perform very well on one metric but poorly on the other.

Composite Score

$$\text{Composite Score} = P + R + \text{Accuracy}$$

This is a sum of Precision, Recall, and Accuracy, providing a comprehensive evaluation metric. The Composite Score ranges from 0 to 3, with higher values indicating better overall performance across all dimensions. This metric is particularly useful for comparing different algorithms when no single metric is clearly more important than the others.

2.4 Experimental Setup

The majority of our code for our project was written in Python 3.13. You can find a repository containing a `main.py` file to run our algorithms on all of our datasets (minus our synthetic datasets) here. All experiments involving our ring of cliques and stochastic block model synthetic data, as well as our PPI dataset, were run on a Carleton College server with 32 cores and 360 GB of RAM available. All other experiments were run on a 2021 MacBook Pro with an Apple M1 Pro chip and 32 GB of RAM.

3 Results

3.1 Baseline Graphs

3.1.1 Karate Club

To have a quick and consistent way to verify that our algorithms worked to identify communities and matched Network X's implementations of them, we ran them on Zachary's Karate Club graph first since it was a small and relatively sparse graph with only 34 vertices and 78 edges, and had two easy to understand ground truth communities [32].

We found that our Girvan-Newman implementation matched Network X's implementation and the original authors' results. As seen in Figure 6 it only misclassified individual 2 when we restricted it to only return two communities and returned the same communities as Network X without this restriction [3, 14]. However, we found that our implementation of the Louvain method did not exactly match Network X's. As shown in Table 1, our implementation found 3 communities with a modularity of 0.43, whereas the Network X implementation found 4 communities with a modularity of 0.44. This was expected because we only run our algorithm on two iterations of the graph – we run it on the original graph, then the resulting graph from the first iteration. In contrast, by default, Network X runs their version of Louvain for as many iterations as needed until the improvement in modularity between iterations is less than $0.1 * 10^{-6}$ [5]. Since BVNS is a relatively new algorithm, as it was only proposed in 2023, we were unable to find any implementation of it from Network X or online, so we did not compare our implementation with any others [19].

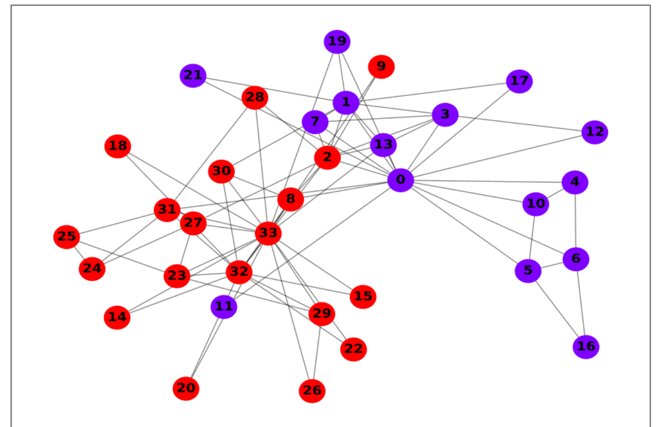

Figure 6: Zachary's Karate Club graph (edge weights omitted) with two communities labeled by our Girvan-Newman implementation

Dataset	$\|V\|$	$\|E\|$	Expected # Communities	Algorithm	# Communities Found	Time (s)	Peak Memory (MB)	Modularity
Karate Club	34	78	2	Girvan-Newman	5	0.27	0.07	0.38
				Louvain	**3**	**0.11**	0.07	0.43
				BVNS	5	14.54	**0.03**	**0.44**
College Football	115	613	12	Girvan-Newman	**10**	28.08	29.19	**0.60**
				Louvain	5	**3.06**	0.20	0.58
				BVNS	15	276.46	**0.04**	0.49
Protein-Protein Interaction	519	679	236	Girvan-Newman	115	190.33	36.50	**0.95**
				Louvain	**162**	**59.54**	21.30	0.78
				BVNS	34	19462.02	**19.1**	0.40
Urban Movement	360	2925	6	Girvan-Newman	**5**	19592.44	1.40	**0.67**
				Louvain	4	**557.48**	1.35	0.65
				BVNS	26	15581.97	**0.41**	0.27
C. Elegans Neurons	473	5025	7	Girvan-Newman	69	8115.11	30.25	0.35
				Louvain	**10**	**265.79**	2.23	**0.53**
				BVNS	27	7845.69	**0.06**	0.25

Table 1: An overview of our results from running each of our implementations on choice datasets, including the number of communities found. Modifications were made to transform all datasets into Network X undirected graphs where necessary. The Girvan-Newman algorithm was run with default parameters. The Louvain method was run with two iterations. BVNS was run with 100 iterations and a k_{max} of 3 for all datasets except for Karate Club and College Football where k_{max} had a value of 4. A bold number in the "# Communities Found" column indicates the result closest to the number of expected communities. In all other columns, a bold number indicates the best result found for that metric for a dataset.

Interestingly though, as seen in Figure 7, when we set our implementations to maximize modularity and not look for a certain defined amount of communities, we observed that all of them found more communities than the ground-truth two communities. This is likely because the two real life communities (the two new karate clubs that formed after the original split), were made up of subgroups of people that had stronger ties to each other than to other members of their club. For instance, in the final communities returned by our Girvan-Newman implementation, individual 2 is in a community with individuals 24, 25, 27, 28, and 31 even though individual 2 was part of a different community than all of them in the real world data. Since all of our algorithms ultimately maximize modularity, we can conclude that modularity may not be the best metric for capturing real-world communities in this scenario.

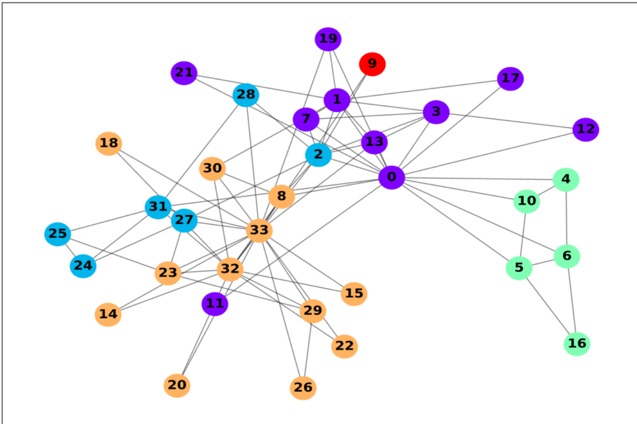

Figure 7: Zachary's Karate Club graph (edge weights omitted) labeled with communities that maximize modularity according to our Girvan-Newman implementation

All of our algorithms returned communities that lead to very similar modularity values that only varied by up to 0.06 as shown in Table 1. These values are close enough that we cannot meaningfully compare them.

The same is true for the peak memory in megabytes used by each algorithm. As shown in Table 1, while BVNS used less than half of the peak memory used by Girvan-Newman and Louvain, since the Karate Club graph is relatively small, the difference (0.04 mb) is small enough that it is challenging to meaningfully compare these results.

We did find a significant difference between BVNS and our other implementations in terms of actual runtime in seconds. In fact, as shown in Table 1, BVNS had an actual running time (14.54 s) that was more than an order of magnitude higher than Louvain (0.11 s) and Girvan-Newman (0.27 s). Although BVNS's time complexity is comparable to Louvain or Girvan-Newman's, it does have a constant factor of 100 that makes it $O(100(|V|^2 \cdot |E|))$. From the results above it is evident that this constant factor makes a significant impact even running on a small and sparse graph like the Karate Club graph. It also makes sense that Girvan-Newman has a slightly worse actual runtime compared to Louvain, as Girvan-Newman's runtime for directed graphs depends on $|E|$ and $|V|$, whereas Louvain's is only dependent on $|V|$.

Overall, Zachary's Karate Club graph served as a good baseline graph for us to test our algorithms. Its small size was beneficial in that it was easy to visualize the differences between the communities returned when maximizing modularity versus the real-life communities. That said, its limited size means that it is hard to compare the modularity and peak memory returned by our algorithms, since all of our results for these two metrics were very similar. However, even with its small size, the run-time discrepancy between BVNS and our other two algorithms was evident.

3.1.2 College Football

To verify our algorithms were performing as expected on a slightly larger and more complicated graph representing a real-world scenario we ran our algorithms on a graph representing the 2000 College Football Division I season and collected our suite of metrics during this experiment.

Due to the fact that we purely used this graph as a baseline and there are 12 ground truth communities and 115 nodes, making the interpretation of the returned communities more difficult, we decided not to analyze the returned communities in as much detail as our other datasets. However, due to its intermediate size between the small Karate Club graph and our larger graphs, the metrics we collected from running our algorithms on this dataset are useful nonetheless.

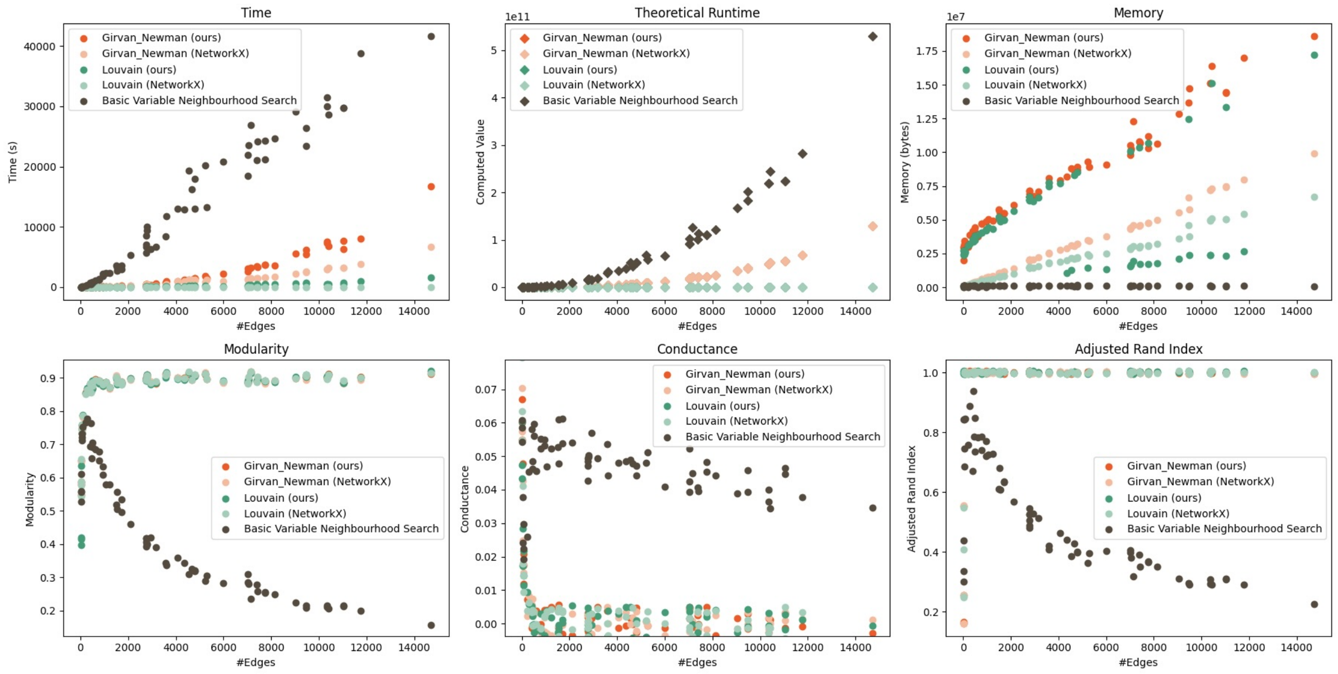

Figure 8: Community detection in sparse ring of cliques graphs, with density ≈ 0.1.

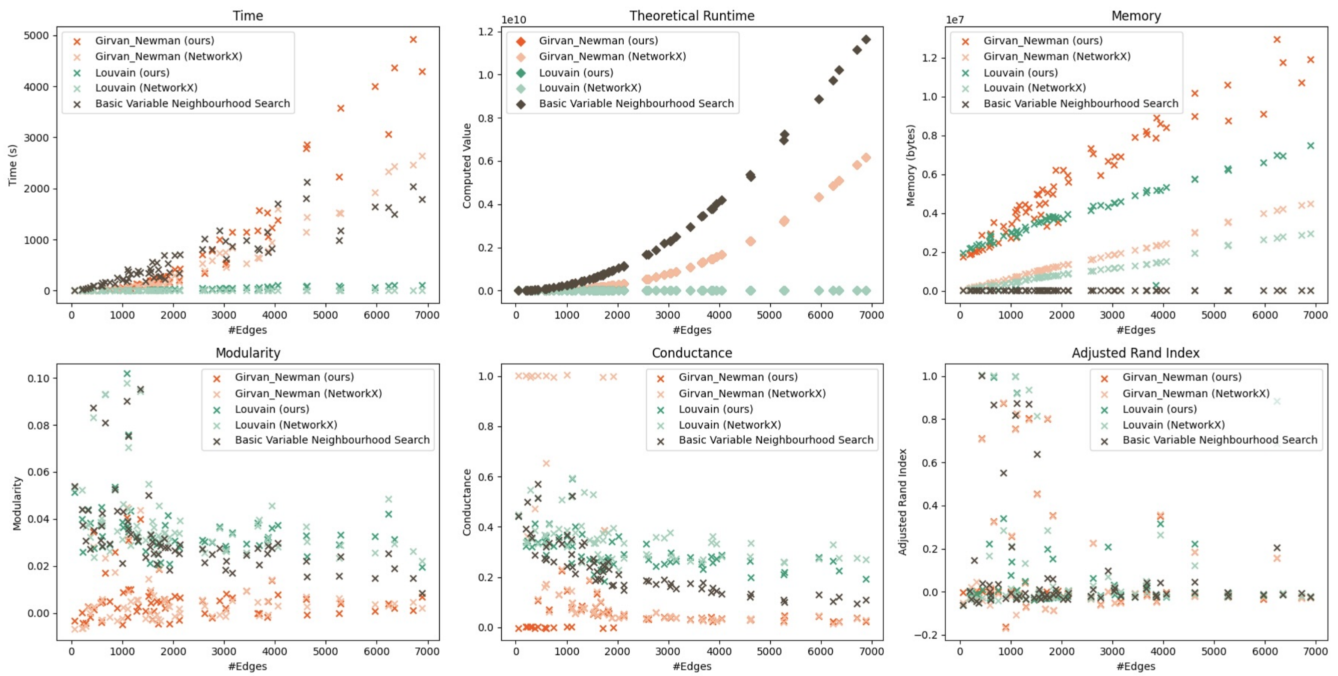

Figure 9: Community detection in dense graphs generated by SBM, with density ≈ 0.8.

When it comes to modularity, Girvan-Newman and Louvain produce similar values that are not significantly different enough to compare. However, as shown in Table 1, BVNS produced communities that led to a modularity value (0.49) significantly lower than those produced by Girvan-Newman (0.60) or Louvain (0.58). This is likely due to the fact that in order to produce better communities on graphs with more nodes, the number of iterations BVNS goes through should be increased, but we hold the number of iterations constant at 100. This can also be seen in the results in Table 1 for the rest of our graphs, as BVNS consistently generates communities with significantly lower modularity as the number of nodes increases.

Interestingly, the difference in peak amount of memory that Girvan-Newman uses compared with Louvain and BVNS is most pronounced in this graph. Girvan-Newman uses 29.19 mb, Louvain 0.20 mb and BVNS only 0.04 mb. This is likely due to the fact that our Girvan-Newman implementation holds all of the nodes and edges of the current version of the graph it is processing in memory at the same time. In this case, nodes are represented as strings and edges as tuples that contain the two strings that represent the nodes it connects. Since Louvain and BVNS do not store all of this information in memory, and nodes are represented as strings in the College Football graph but as integers in the Karate Club graph, Girvan-Newman ends up using significantly more memory than either of them when run on the College Football graph.

This graph's larger size is most apparent in the difference between the actual runtime and peak memory of Girvan-Newman and Louvain on display. As shown in Table 1, Girvan-Newman has an actual runtime of 28.08 s whereas Louvain takes only 3.06 s. It follows that the larger the graph the larger the difference we expect to see in the actual run-times of Girvan-Newman and Louvain, because Girvan-Newman has a runtime dependent on both $|V|$ and $|E|$ whereas Louvain has a runtime dependent only on $|V|$. Additionally, this graph contains only 81 more nodes than the Karate Club graph, but 535 more edges, which is arguably the largest contributor to Girvan-Newman's runtime being much longer than Louvain's. However, since our real-world data does not consistently increase in size, these results should be interpreted with a degree of scrutiny. For a more direct comparison between runtime and graph size, we turn to our synthetic data.

3.2 Synthetic Data

3.2.1 Ring of Cliques and Stochastic Block Model Synthetic Graphs

For our synthetic dataset benchmarks, we present results from two distinct scenarios: (1) a sparse ring of cliques (referred to as "sparse ring"), representing an easier detection challenge, and (2) a dense stochastic block model graph (referred to as "dense SBM"), representing a more difficult challenge. Complete results are available in Appendix A. The sparse ring configuration presents a relatively straightforward community detection scenario. Each clique forms a complete graph internally, with cliques connected to their neighbors by only single edges. This creates a clear community structure that algorithms should readily identify. In contrast, the dense SBM configuration presents a more challenging scenario. Despite having well-defined ground truth communities, this model incorporates numerous inter-community edges (determined by the inter-community density parameter). These additional connections potentially dilute key metrics used by various algorithms, such as the betweenness centrality calculations in Girvan-Newman, making community boundaries less distinct.

As shown in Figure 8, for sparse ring networks, the measured runtime patterns largely correspond to the theoretical Big-O complexity expectations, with BVNS exhibiting the slowest performance. However, both Girvan-Newman and Louvain consume more memory than BVNS. Additionally, our implementations of Girvan-Newman and Louvain are more memory-intensive than their NetworkX counterparts. Regarding community quality metrics, Girvan-Newman and Louvain perform equally well in terms of modularity, while BVNS shows declining performance as graph size increases. This decline can be attributed to BVNS using a fixed number of optimization steps regardless of graph size. In terms of conductance, Girvan-Newman and Louvain achieve the lowest values, indicating well-separated communities, unlike BVNS. The ARI is highest for Girvan-Newman and Louvain, indicating greater overlap with ground truth communities, while BVNS shows significantly lower and decreasing ARI values with increasing graph size, mirroring its modularity performance.

As illustrated in Figure 9, when transitioning from simple, sparse graphs to challenging, dense graphs, we observe several key shifts in algorithm performance. The runtime of Girvan-Newman eventually exceeds that of BVNS as network density increases. Similarly, the peak memory usage of Girvan-Newman significantly surpasses that of Louvain in dense network scenarios. All algorithms demonstrate reduced effectiveness with dense SBM graphs, with Girvan-Newman experiencing the most substantial performance decrease. This could be attributed to the fact that edge betweenness centrality becomes less useful and more expensive to compute in dense graphs where many edges span community boundaries. The performance of Louvain also decreases, which could be attributed to the fact that Louvain greedily optimizes for modularity. With a higher number of edges, it encounters a 'resolution limit,' preventing it from detecting communities below a certain size that depends on the total number of edges, leading to over-merging.

3.2.2 Simulated Urban Movement Graphs

To simulate graph representations of real-world urban movement data, We generate a series of synthetic trajectory datasets within a rectangular study area using the following steps. First, the rectangular area is randomly divided into six arbitrary-shaped regions: Red, Pink, Green, Light Green, Blue, and Light Blue. Second, we generate 9 clusters of trajectories, each containing 10 trajectories, where each trajectory (a moving object) has 10 sampled location points. The spatial distribution of points on a trajectory depends on which cluster it is in. We sample locations based on the requirements explained below.

There are two types of clusters of moving objects:

- **Type 1**: These have a 90% chance of staying in one region and a 10% chance of moving randomly, which we call noise (cluster IDs 1-6).
- **Type 2**: These have a 90% chance of staying in two regions (equal presence in both regions) and also move randomly 10% of the time (cluster IDs 7-9).

The configuration of these 9 clusters is shown in Table 2. We also generate a second dataset with 20% noise in each trajectory, as shown in Table 3. Next, a weighted graph G is constructed, with 360 points (i.e., $0.4 \times 10 \times 10 \times 9$), as explained in Sec. 2.2.3.

Cluster ID	Number of trajectories	Prob. in Red	Prob. in Pink	Prob. in Green	Prob. in Light Green	Prob. in Blue	Prob. in Light Blue	Prob. of Noise	Total
1	10	90%						10%	100%
2	10		90%					10%	100%
3	10			90%				10%	100%
4	10				90%			10%	100%
5	10					90%		10%	100%
6	10						90%	10%	100%
7	10	45%	45%					10%	100%
8	10			45%	45%			10%	100%
9	10					45%	45%	10%	100%

Table 2: Synthetic data of trajectories with 10% noise or random moves.

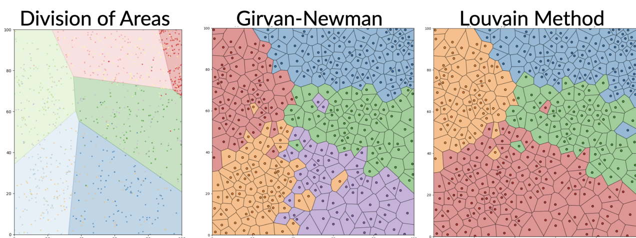

Figure 10: Comparison of ground truth with under 10% noise with results from Girvan-Newman and Louvain

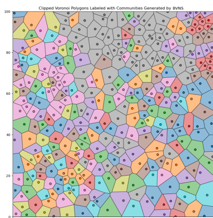

Figure 11: Results of running BVNS on the same division of areas seen in Figure 10

Then, We apply our three community detection algorithms (Girvan-Newman, Louvain, and BVNS) to the resulting graph. Each discovered community is represented by a unique, randomly assigned color.

The results with 10% noise are shown in Figure 10 and Table 1. The right side represents the ground truth division of regions, while the left side shows the results of the Girvan-Newman and Louvain algorithms. BVNS is not included in this comparison because it performed very poorly, as shown in Figure 11. Specifically, BVNS failed to detect any meaningful communities, which is expected since its tactic contains lot of randomness. Among the three algorithms, Girvan-Newman effectively identifies distinct areas with high precision but is computationally expensive, taking 19,592.44 seconds (nearly six hours) to run. In contrast, Louvain is significantly faster, completing in just 557 seconds (about 10 minutes). However, Louvain's greedy approach sometimes merges neighboring regions too aggressively, particularly in the bottom two areas.

Cluster ID	Number of trajectories	Prob. in Red	Prob. in Pink	Prob. in Green	Prob. in Light Green	Prob. in Blue	Prob. in Light Blue	Prob. of Noise	Total
1	10	80%						20%	100%
2	10		80%					20%	100%
3	10			80%				20%	100%
4	10				80%			20%	100%
5	10					80%		20%	100%
6	10						80%	20%	100%
7	10	40%	40%					20%	100%
8	10			40%	40%			20%	100%
9	10					40%	40%	20%	100%

Table 3: Synthetic data of trajectories with 20% noise or random moves.

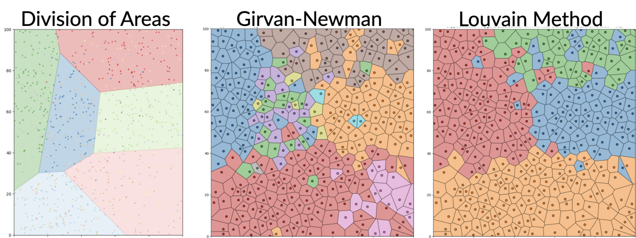

Figure 12: Comparison of ground truth with under 20% noise with results from Girvan-Newman and Louvain.

When noise increases to 20%, the results are shown in Figure 12. Louvain maintains its efficiency and continues identifying major boundaries, though some merging still occurs. Meanwhile, Girvan-Newman becomes overly sensitive, producing many small communities and failing to form clear boundaries. For instance, it fragments the blue region into multiple small communities and, while detecting the pink area, fails to accurately define its boundaries. Additionally, Girvan-Newman remains computationally inefficient, taking 3,235.12 seconds, whereas Louvain completes in just 335.55 seconds.

Overall, Girvan-Newman is highly sensitive to the movement of objects in trajectories, as it detects high-betweenness edges, while Louvain tends to merge more communities by prioritizing dense internal connections. However, Girvan-Newman is significantly slower; in this experiment, it often requires hours to run, while Louvain takes no more than 20 minutes. This inefficiency arises because Girvan-Newman repeatedly computes betweenness, which involves costly shortest-path calculations resulting in an overall runtime of $O(|E|^2|V|\log|V|))$, whereas Louvain employs a greedy heuristic to merge nodes and aggregate communities efficiently, with a complexity of $O(|V|\log|V|)$. Moreover, while Girvan-Newman is highly sensitive to community structures, it becomes unreliable in high-noise settings. In such cases, it can be oversensitive, splitting more communities than the ground truth and failing to detect accurate boundaries. This is plausible because Girvan-Newman removes high-betweenness edges first, and in noisy conditions, these edges may not always correspond to true community boundaries, leading to inaccurate splits. On the other hand, Louvain, though less sensitive, still identifies clear and meaningful boundaries, making it the more suitable algorithm for large and complex datasets.

3.3 Real-world Data

3.3.1 PPI

As shown in Table 4, for community detection in our yeast PPI network, Louvain demonstrates the best performance in the majority of the evaluation metrics. Comparing Louvain (NX) to the second-best method in each metric from Table 4 reveals striking advantages. Louvain (NX) is roughly 99.8% faster in runtime (0.103 s vs. 59.54 s) and consumes about 95% less memory (1.074 MB vs. 20.026 MB). In terms of clustering quality, it achieves a 0.47% higher F-measure (0.216 vs. 0.215) and 0.70% higher accuracy (0.432 vs. 0.429). Finally, its Composite Score is approximately 1.31% higher than the second-best (1.009 vs. 0.996). Unlike patterns observed in synthetic datasets, Girvan-Newman performs significantly worse than Louvain on real-world PPI networks. This performance disparity may be attributed to Girvan-Newman's reliance on high-betweenness edge removal, a strategy better suited for breaking down large communities rather than analyzing graphs with numerous small communities or pre-existing disconnected components. In contrast, Louvain's community formation process only considers adjacent communities when relocating nodes, ensuring that a path must exist between all nodes in a community. This constraint prevents the algorithm from constructing communities that span disconnected components. BVNS lacks this constraint, allowing it to form communities across disconnected components, which may explain its reduced effectiveness on PPI networks that naturally contain multiple disconnected subgraphs. In addition, similar to the patterns observed in synthetic datasets, Network X's implementation is more efficient for Girvan-Newman and Louvain. Their version of Louvain also achieves a better F-measure, accuracy, and composite score, which can be attributed to the fact that we only run our algorithm on two iterations of the graph – we run it on the original graph, then the resulting graph from the first iteration. In contrast, the Network X version runs until the improvement in modularity between iterations is less than $0.1*10^{-6}$.

Algorithm	# Communities	Modularity	Runtime (s)	Peak Memory (MB)	F-measure	Accuracy	Composite Score
Louvain (ours)	162	0.780	59.540	22.340	0.205	0.393	0.834
Louvain (NX)	114	0.953	**0.103**	**1.074**	**0.216**	**0.432**	**1.009**
Girvan-Newman (ours)	115	**0.954**	190.330	38.265	0.215	0.429	0.996
Girvan-Newman (NX)	115	**0.954**	155.027	20.874	0.215	0.429	0.996
BVNS	34	0.400	19462.016	20.026	0.000	0.191	0.337

Table 4: Community detection in yeast PPI graph.

3.3.2 *C. Elegans* Neuronal Connectivity

Algorithm	# of Communities	Directed Modularity	Undirected Modularity	ARI
Spectral Method	7	0.52	0.49	1.00
Girvan-Newman	69	0.37	0.35	**0.47**
Louvain	10	**0.53**	**0.53**	0.43
BVNS	27	0.23	0.25	0.02

Table 5: Community detection algorithms in the *C. elegans* graph. The directed approach is based on a Leicht-Newman algorithm and implemented using the cdlib library [20,25]. ARI is the assigned rand index between the directed approach's communities and the other communities, explained in detail in section 2.3.3.

As shown in Table 5, Louvain and the spectral method have the best modularity in the directed version of the graph with 0.53 and 0.52 respectively. However, the Girvan-Newman communities are the most similar to the spectral method's communities with an ARI of 0.47 to Louvain's 0.43. BVNS is far behind across all metrics other than memory usage, as can be seen in Table 1.

Similar to our PPI dataset, Girvan-Newman performs worse than Louvain for neuronal connectivity. Its lower modularity may be attributed to its tendency to overcount the number of communities by making more divisions than necessary. With more than double the number of communities of any other method, this seems like a likely scenario. The number of communities can also potentially explain the similar modularities of the spectral method and Louvain algorithm. They both had the fewest communities and the highest modularity, indicating that fewer communities may be better for this graph. Despite Louvain having higher modularity and fewer communities, the Girvan-Newman communities are the most similar to the spectral method communities. This similarity could be due to the Girvan-Newman algorithm finding subcommunities of the larger circuits. Girvan-Newman is able to uncover precise differences, but at the cost of potentially creating too many communities. One possibility is that the Girvan-Newman communities are similar to the spectral method communities when you combine them together. BVNS's poor performance is possibly due to the number of iterations. With 100 iterations for almost 500 vertices, it may not have had enough time to move vertices to ideal communities.

4 Discussion

4.1 Key Insights from Our Results

If there is one key takeaway from our project, it's that no single algorithm is universally optimal for community detection with real-world data. We were able to reach this conclusion because of the wide variety of datasets that we used. They differed greatly in their size and in their contextual definition of communities. For example, as shown in Table 1, our College Football dataset has 115 nodes, 613 edges, and the ground truth communities are 12 Division I college football conferences. In contrast, the *C. Elegans* Neurons dataset has 473 nodes, 5,025 edges, and the ground truth communities are seven different circuits of neurons in the nematode *C. Elegans*. This great difference we found among just a few datasets means that it is impossible to say that one algorithm is definitively the best for finding communities in real-world data, and our results verify this.

When we turn to our results in Table 1, we find that no single algorithm performs the best on all the metrics we collected across all of our datasets or even within a single dataset. Additionally, only two of our algorithms performed the best for one metric across datasets: Louvain in actual runtime (s) and BVNS in Peak Memory (mb). If one wanted to minimize either, then the respective algorithms would be the clear choice. However, both have significant trade-offs.

BVNS uses the least amount of peak memory, but on average across our datasets performs the worst when it comes to every other metric. So, while there is a use case for it if one wants to minimize memory usage at all costs, it definitely is not universally optimal. One other possibility is to use BVNS to find the maximum modularity set of communities. BVNS is the only one of our algorithms that can continuously improve with more time, so it is possible to use BVNS to find communities with the maximum modularity for a graph with enough iterations. However, while modularity is a good metric for identifying useful communities, it is not perfect. In fact, for situations like the Karate Club and College Football datasets with existing hand-labeled communities based on real-world observations, these communities have worse modularity than the communities found using our algorithms. So, while we can potentially use BVNS to find the maximum modularity set of communities, there is not necessarily a good reason to do so.

Louvain has the best actual runtime across all of our datasets, which is supported by it having the best theoretical runtime of all three of our algorithms, $O(|V|log|V|)$. In contrast, BVNS has a runtime of $O(100(|V|^2 \cdot |E|))$ and Girvan-Newman $O(|V||E|^2)$ for unweighted graphs and $O(|E|^2|V|log|V|)$ for weighted graphs. However, it is often inconsistent with the quality of communities it produces. As Table 1 shows, it only produced the highest modularity communities for the *C. Elegans* dataset. This may be due to the fact that, like other greedy approaches, it is prone to getting trapped in local optima. It also only produces a number of communities that most closely matches the expected number of communities when running on our Karate Club and *C. Elegans* datasets. Additionally, it consistently has the second smallest peak memory usage among the three algorithms we looked at. Therefore, while Louvain may seem like the overall best choice of the three algorithms we studied, in a scenario where consistently maximizing modularity is crucial, one should consider running other algorithms in addition to Louvain, one of which could be Girvan-Newman.

In some ways, Girvan-Newman's strengths and weaknesses are the opposite of Louvain's. As shown in Table 1, it more consistently finds the highest modularity communities, finding them in three out of five real-world scenarios that we examined. However, this comes with the significant trade-off that in all datasets we looked at except for Zachary's Karate Club, Girvan-Newman used significantly more memory than Louvain or BVNS. This difference peaked in our *C. Elegans* Neurons dataset where Girvan-Newman used 28.02 mb more memory at the peak of its memory usage than Louvain and 30.19 mb more than BVNS. Additionally, compared to Louvain, it has a worse runtime for every data set we looked at.

Interestingly, for the two graphs with the highest edge counts, Urban Movement and *C. Elegans* Neurons, Girvan-Newman had a runtime worse than even BVNS. This is because one of the major downsides of Girvan-Newman's divisive approach to findings communities is that it leads to a runtime that is not only dependent on $|V|$ like the runtimes of BVNS and Louvain, but also $|E|^2$. However, because of this approach, we theorize that Girvan-Newman may be better at finding communities in scenarios with many ground truth communities than Louvain. Since Girvan-Newman removes one edge per iteration, it alters the community structure more gradually than Louvain, which merges communities each iteration. This slower, more granular division may allow Girvan-Newman to preserve a greater number of communities, making it potentially more effective in scenarios with many ground-truth communities. We see this reflected in our results in Table 1, as for all datasets except for Protein-Protein Interaction, Girvan-Newman produces more communities than Louvain. To verify the accuracy of this theory, we would repeat our experiments with synthetic data and collect the amount of communities generated by all of our algorithms to compare them.

All of these interpretations should be taken with a degree of skepticism though. Due to time constraints, we only ran each experiment once on only five real-world datasets. So while our datasets represented a variety of real-world disciplines, given more time, we would have addressed a number of limitations of our experimental setup.

4.2 Strengths and Limitations of Our Approach

Overall, we had a good diversity of datasets to run our implementations of our algorithms on. We ran our implementations and their Network X equivalents on two types of synthetic graphs with varying edge counts, real world data that we or outside sources had transformed into graphs with varying node and edge counts, and a graphs we constructed to simulate real world urban movement data. This allowed us to see how our implementations performed in a wide variety of scenarios, better revealing their strengths, weaknesses, and the contexts in which they perform best.

However, to further verify the accuracy of the claims we made, we would expand the datasets we ran our implementations on to more graphs of varying sizes and ones that covered more real-world scenarios. We would also run more algorithms on these graphs, starting with an improved version of Louvain, the Leiden method. The Leiden method was created to address a common issue with the Louvain method which is that it might generate communities with multiple disconnected components. The Leiden method addresses this by adding an additional step named Refinement to ensure that no communities are disconnected [28].

As we explained in our Methods section, we ran our experiments in different environments. Given more time, we would adapt all of our experiments to run on the server provided to us so that the conditions under which our experiments ran would be the same. That said, the memory and CPU usage of any one given instance of our algorithm was not significant, so it is likely that the environment in which we ran single instances of our algorithms did not significantly influence our final metrics for actual runtime or peak memory usage.

Ideally, we would also have increased the number of iterations that BVNS went through based on the size of the graph, but this would have significantly increased its already less than ideal runtime. Since BVNS makes one locally optimal move per iteration, basing the number of iterations on the number of nodes would allow it to make one locally optimal move per node. While the moves in the local search step would likely be more unevenly distributed, this change would probably help BVNS find solutions with comparable modularity. However, the way we implemented BVNS is only usable with small graphs.

As shown in our results from our synthetic data, our algorithms generally performed worse than their Network X counterparts on all of the metrics we looked at. However, it should be noted that since BVNS is a relatively new algorithm, there was no implementation from Network X or elsewhere to compare it to. Given more time, we would investigate how to better optimize our implementations. We have identified that Network X calculates modularity differently from us because it uses a slightly different equation [7]. We theorize that part of why Network X implementations tend to have a better actual runtime is because this method of calculating modularity involves iterating over only all nodes in a graph, not all of the edges like our approach does. Given more time, we would further investigate the differences between this alternative method of calculating modularity and our own and potentially implement it ourselves to further optimize our implementations.

5 Conclusion

No one algorithm performs the best consistently across all metrics and datasets we examined. This leads us to conclude that one should utilize a variety of algorithms with different approaches to finding communities when attempting to detect communities in real-world datasets. Each algorithm and category of algorithms has its own strengths and weaknesses, useful for different real-world scenarios.

6 Acknowledgments

We thank Layla Oesper for advising us throughout this project. Her feedback and guidance were incredibly helpful to us. Also, thank you to Mike Tie for providing us with computational resources for our project.

References

[1] Betweenness Centrality. 2

[2] Cats UK | Data Import & EDA Starter. 4

[3] girvan_newman — NetworkX 3.4.2 documentation. 3, 7

[4] Karate Club — NetworkX 3.4.2 documentation. 4

[5] louvain_communities — NetworkX 3.4.2 documentation. 7

[6] Microsoft Geolife GPS Trajectory Dataset | Kaggle. 4

[7] modularity — NetworkX 3.4.2 documentation. 13

[8] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, Oct. 2008. doi: 10.1088/1742-5468/2008/10/P10008 3

[9] U. Brandes. A faster algorithm for betweenness centrality*. *The Journal of Mathematical Sociology*, 25(2):163–177, June 2001. Publisher: Routledge _eprint: https://doi.org/10.1080/0022250X.2001.9990249. doi: 10.1080/0022250X.2001.9990249 2, 3

[10] S. J. Cook, T. A. Jarrell, C. A. Brittin, Y. Wang, A. E. Bloniarz, M. A. Yakovlev, K. C. Q. Nguyen, L. T.-H. Tang, E. A. Bayer, J. S. Duerr, H. E. Bülow, O. Hobert, D. H. Hall, and S. W. Emmons. Whole-animal connectomes of both Caenorhabditis elegans sexes. *Nature*, 571(7763), July. doi: 10.1038/s41586-019-1352-7 2

[11] S. Dilmaghani, M. R. Brust, C. H. C. Ribeiro, E. Kieffer, G. Danoy, and P. Bouvry. From communities to protein complexes: A local community detection algorithm on PPI networks. *PLoS ONE*, 17(1):e0260484, Jan. 2022. doi: 10.1371/journal.pone.0260484 5

[12] S. W. Emmons. Comprehensive analysis of the C. elegans connectome reveals novel circuits and functions of previously unstudied neurons. *PLOS Biology*, 22(12):e3002939, Dec. 2024. Publisher: Public Library of Science. doi: 10.1371/journal.pbio.3002939 2, 5

[13] J. O. Garcia, A. Ashourvan, S. Muldoon, J. M. Vettel, and D. S. Bassett. Applications of Community Detection Techniques to Brain Graphs: Algorithmic Considerations and Implications for Neural Function. *Proceedings of the IEEE*, 106(5):846–867, May 2018. doi: 10.1109/JPROC.2017.2786710 2

[14] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, June 2002. Publisher: Proceedings of the

National Academy of Sciences. doi: 10.1073/pnas.122653799 2, 3, 4, 7

[15] D. Guo, H. Jin, P. Gao, and X. Zhu. Detecting spatial community structure in movements. *International Journal of Geographical Information Science*, 32(7):1326–1347, July 2018. doi: 10.1080/13658816.2018.1434889 4

[16] A. Hagberg, P. J. Swart, and D. A. Schult. Exploring network structure, dynamics, and function using NetworkX. Technical Report LA-UR-08-05495; LA-UR-08-5495, Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), Jan. 2008. 4

[17] M. A. Javed, M. S. Younis, S. Latif, J. Qadir, and A. Baig. Community detection in networks: A multidisciplinary review. *Journal of Network and Computer Applications*, 108:87–111, Apr. 2018. doi: 10.1016/j.jnca.2018.02.011 1

[18] Jesse. Answer to "Drawing a ring of cliques in tikz-graphs", Jan. 2015. 4

[19] D. Jovanović, T. Davidović, D. Urošević, T. J. Krüger, and D. Ramljak. Variable Neighborhood Search Approach to Community Detection Problem. In I. Georgiev, M. Datcheva, K. Georgiev, and G. Nikolov, eds., *Numerical Methods and Applications*, pp. 188–199. Springer Nature Switzerland, Cham, 2023. doi: 10.1007/978-3-031-32412-3_17 3, 7

[20] E. A. Leicht and M. E. J. Newman. Community Structure in Directed Networks. *Physical Review Letters*, 100(11):118703, Mar. 2008. doi: 10.1103/PhysRevLett.100.118703 5, 6, 12

[21] Y. Li, S. Cheng, Y. Feng, Y. Zhang, P. Angeloudis, M. Quddus, and W. Y. Ochieng. Developing a novel approach in estimating urban commute traffic by integrating community detection and hypergraph representation learning. *Expert Systems with Applications*, 249:123790, Sept. 2024. doi: 10.1016/j.eswa.2024.123790 2

[22] J. Liu and Y. Yuan. Exploring dynamic urban mobility patterns from traffic flow data using community detection. *Annals of GIS*, 30(4):435–454, Oct. 2024. doi: 10.1080/19475683.2024.2324393 2

[23] S. Pu, J. Wong, B. Turner, E. Cho, and S. J. Wodak. Up-to-date catalogues of yeast protein complexes. *Nucleic Acids Research*, 37(3):825–831, Feb. 2009. doi: 10.1093/nar/gkn1005 5

[24] S. Rahiminejad, M. R. Maurya, and S. Subramaniam. Topological and functional comparison of community detection algorithms in biological networks. *BMC Bioinformatics*, 20(1):212, Dec. 2019. doi: 10.1186/s12859-019-2746-0 2

[25] G. Rosetti. cdlib.algorithms.rb_pots. 6, 12

[26] S. Saha, P. Chatterjee, S. Basu, M. Nasipuri, and D. Plewczynski. FunPred 3.0: improved protein function prediction using protein interaction network. *PeerJ*, 7:e6830, May 2019. Publisher: PeerJ Inc. doi: 10.7717/peerj.6830 5

[27] D. Szklarczyk, R. Kirsch, M. Koutrouli, K. Nastou, F. Mehryary, R. Hachilif, A. L. Gable, T. Fang, N. Doncheva, S. Pyysalo, P. Bork, L. Jensen, and C. von Mering. The STRING database in 2023: protein–protein association networks and functional enrichment analyses for any sequenced genome of interest. *Nucleic Acids Research*, 51(D1):D638–D646, Jan. 2023. doi: 10.1093/nar/gkac1000 5

[28] V. A. Traag, L. Waltman, and N. J. van Eck. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, 9:5233, Mar. 2019. doi: 10.1038/s41598-019-41695-z 13

[29] J. Wang, J. Cao, W. Li, and S. Wang. CANE: community-aware network embedding via adversarial training. *Knowledge and Information Systems*, 63(2):411–438, Feb. 2021. doi: 10.1007/s10115-020-01521-9 4

[30] E. W. Weisstein. Caveman Graph. Publisher: Wolfram Research, Inc. 1

[31] B. Yang, D. Liu, and J. Liu. Discovering Communities from Social Networks: Methodologies and Applications. In B. Furht, ed., *Handbook of Social Network Technologies and Applications*, pp. 331–346. Springer US, New York, NY, 2010. doi: 10.1007/978-1-4419-7142-5_16 1

[32] W. W. Zachary. An Information Flow Model for Conflict and Fission in Small Groups. *Journal of Anthropological Research*, 33(4):452–473, 1977. Publisher: [University of New Mexico, University of Chicago Press]. 4, 7

A Appendix A

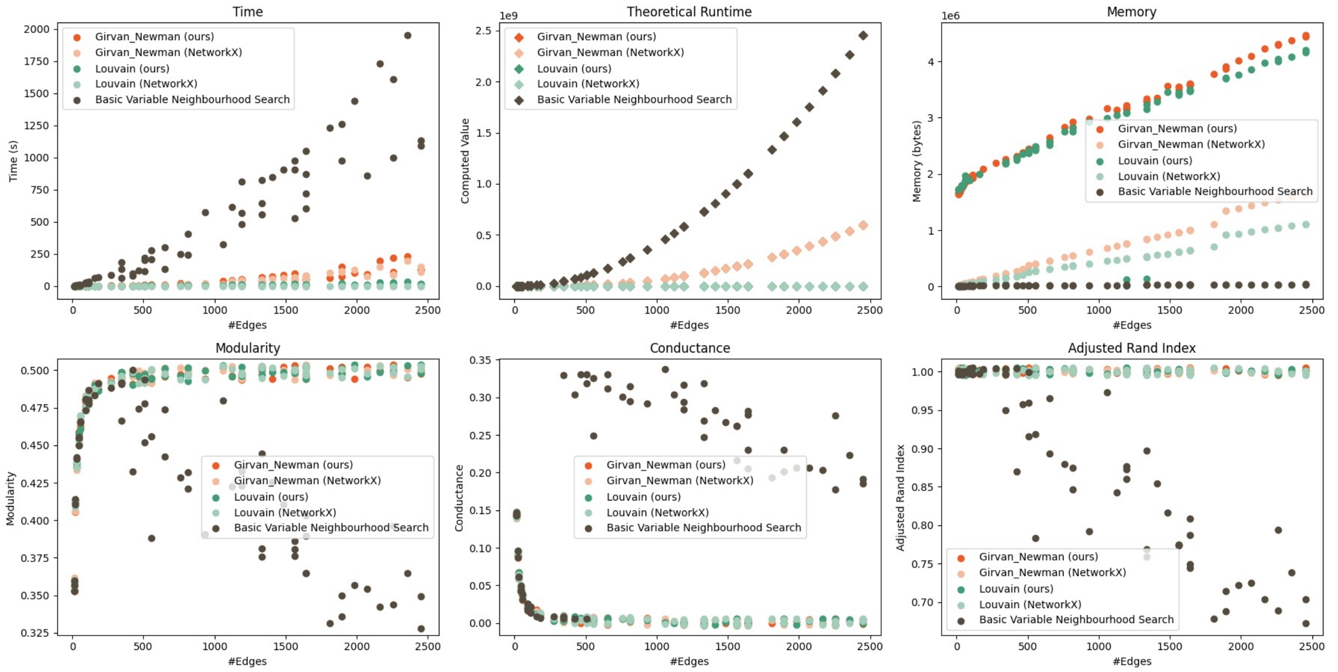

Figure 13: Community detection in dense ring of cliques graphs, with density ≈ 0.5.

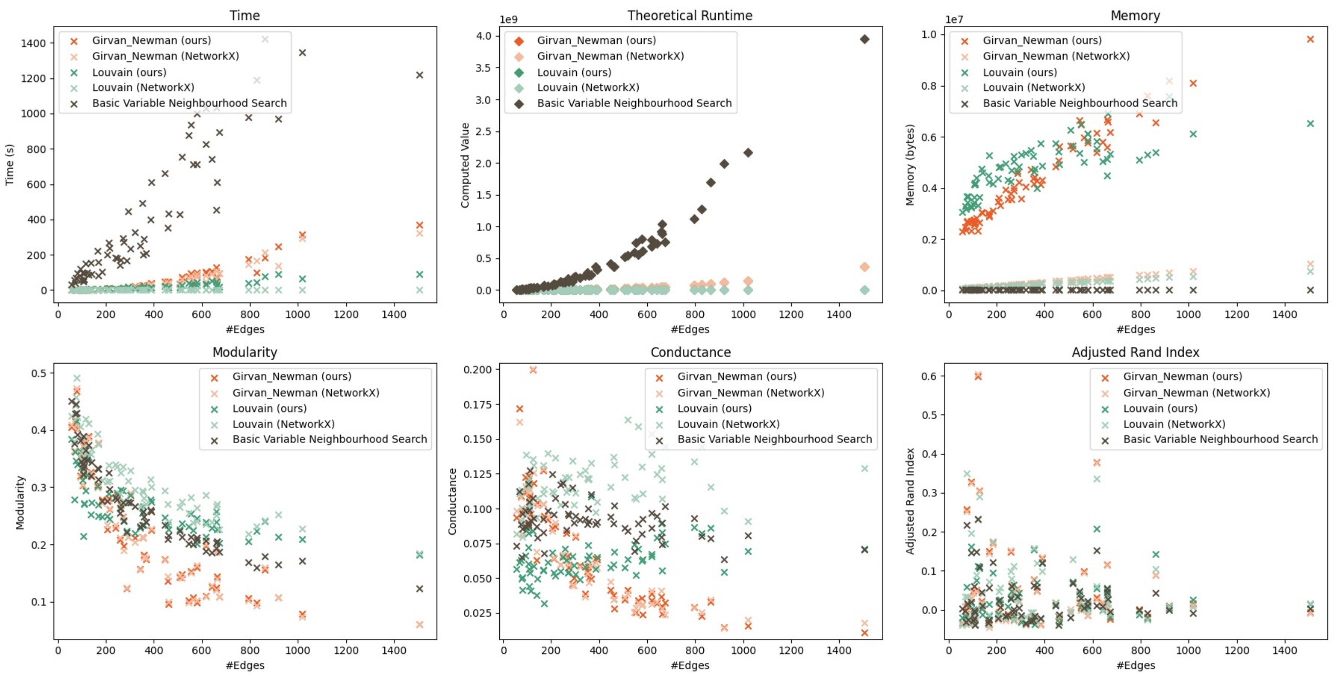

Figure 14: Community detection in sparse graphs generated by SBM, with density ≈ 0.1.