A Similarity-Based Genetic Approach for Discovering Community Structure Within Social Networks

Student Name: D.R. Elkins
Supervisor Name: I.A. Stewart
Submitted as part of the degree of BSc Computer Science to the
Board of Examiners in the Department of Computer Sciences, Durham University

Abstract—The field of social network analysis has grown more prominent in recent years and as such problems in the field are seeing novel approaches. The problem of community detection is traditionally based in graph theory and aims to partition a graph into "communities" of nodes which are "closer" to each other than nodes outside of the communities. Some modern solutions apply meta-heuristic methodologies, specifically genetic algorithms. The following paper aims to critique a promising genetic algorithm presented in the GA-BCD paper. The GA-BCD paper claims to outperform traditional methods in both time and result quality while differing significantly from other genetic algorithms. This paper attempted to recreate the results of the paper and in doing so, found major discrepancies. The various aspects of designing a genetic algorithm for community detection are assessed and through this process a novel algorithm (S-BGACD) was developed which outperforms GA-BCD by applying alternative methodologies for initialization, mutation, and crossover.

Index Terms—Clustering, Evolutionary computing and genetic algorithms, Graphs and networks, Social networking

1 Introduction

The field of social network analysis aims to study and quantitatively analyse the structure of social networks. Social networks represent some form of social grouping as a graph [1]. The social groupings can vary wildly, including such topics as bottlenose dolphin interaction [2], political blogs sharing links to other blogs [3], and friend groups in a karate club [4]. In all of these topics, individuals within the subject group are represented as nodes and interactions between individuals are represented by edges.

These graphs can be analysed in a range of ways, with community structure being one of the most common aspects. Within social settings it is common for some groups to form, these could be friend groups within a school, teams within an office, or collaborating institutions within a field of research. The aim of community detection is to discover these groupings from the structure of their social network alone [5].

Community detection has been applied to many problems across fields. Discovering community structures within internet users could allow geographically close users with similar interests to be grouped into a cluster which accesses a mirror server to improve user experience [5], [6]. Detecting groups of users with similar online shopping habits allows for efficient advertising [7]. Political affiliations of users of social platforms can be tracked to better understand the online political landscape [8]. There are public health applications such as epidemic diseases, grouping and detecting cancers, and studying organ function [9], [10], [11], [12], [13], [14]. Across the field, datasets are growing and becoming more complex. This is mostly due to the rise of social network services (SNS) such as Facebook and Instagram, which contain social networks of unprecedented size and interaction. These SNS networks also have the most obvious needs for community detection. This has caused a growth in need for efficient and high quality algorithms [15].

This problem has been solved using many algorithms with varying definitions of communities, closeness, and quality. A large number of these algorithms are based around optimizing the modularity of a system, a method of quality assessment. Modularity focuses on having many edges within communities, comparing the count of in community edges to the expected number of edges from a random partitioning [16]. These approaches remain very popular to this day. Other algorithms use unique understandings of how communities work to discover new ones, such as the random walk algorithm [17]. Alternatively, genetic algorithms have been applied. Genetic algorithms work along the principles of evolution. They start with an initial random population defining the partitions of the network. These are then assessed by some objective function or "fitness" function that assigns quality for each member of the population. The next generation is then created from this population using various crossover methods where members of the population are combined, or mutations, where small changes are made to existing members of the population and are then added to the next generation. The algorithm that is the subject of this paper is a genetic algorithm known as GA-BCD [18]. The GA-BCD paper proposes a novel quality assessment method called "group similarity density". Group similarity density focuses on the "similarity" of nodes within communities, a metric that uses the neighbours of any two nodes to quantify how closely

connected they are. The concept of using node similarity for community detection has been used previously by Ying Pan et al. which merges nodes together to create communities based on similarity [19]. The combination of a genetic structure and a new objective function sets this proposed algorithm apart from the majority of the field. The claimed performance marginally surpasses that of the state-of-the-art algorithms and is more time efficient, see [16, Fig. 1].

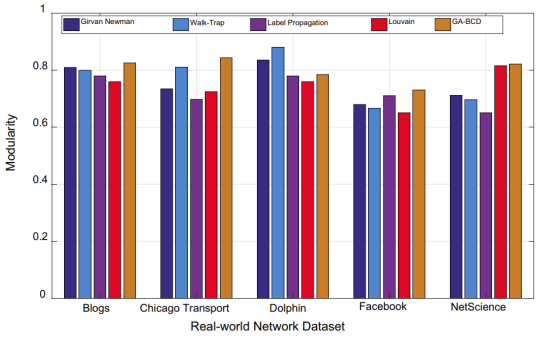


Fig. 1. The modularity score of various algorithms on the datasets listed on the X axis. The results of GA-BCD can be seen in orange. This graph contains the claims of performance made by the GA-BCD paper relative to the state of the art. It is important to note the scale of the graph is off, the actual values should vary between 0.4 and 0.9

The main goal of this paper is to have a look at the GA-BCD algorithm, recreate it, and understand in order to create an improved version of it employing techniques from many genetic algorithms in the field. With normal heuristic algorithms such as the popular Girvan-Newman(GN) algorithm, there are few operations performed with specific reasoning [20]. In GN they work with a single metric, edge betweenness, and a single operation, removing an edge. This creates an understandable algorithm with very little to alter. This is not the case for meta-heuristic algorithms. The meta-heuristic approaches tend to be far more complex with a greater number of operations and metrics, leading to less understandable algorithm design. By understanding the reasoning behind design choices and what their alternatives are in GA-BCD, it is easier to understand the sub-field of genetic community detection algorithms. The GA-BCD algorithm can be cut into six major steps: initial population generation, genetic representation, objective function, selection process, crossover operations, and mutation operations. All of these are designed differently between algorithms and are all important to an algorithm's performance. At the end, the new algorithm developed through the understanding gained in previous sections will be tested against GA-BCD, as well as various comparison algorithms.

2 Related Work

The community detection problem dates back to around the 1980s, when it was used primarily for sociological work [1]. For the first 20 years of the problem's study, the hierarchical clustering algorithm was the most prevalent [20]. In this approach, the edges of a graph are granted weights based off of some structural calculation. All edges are then removed from the graph and progressively added back in based on the order of weight, the edges with the highest weight being added back first. A tree is created by linking nodes in the order of first connection. The resulting tree is hierarchical, with the height of a node's connection to a grouping of nodes indicating the strength of its connection to that grouping, with lower connections being stronger. This tree can then be sliced to create communities. This approach was somewhat effective but did not perform especially well under testing.

The field was then revolutionized in 2002 with the publication of the Girvan-Newman algorithm [20]. The introduction of this algorithm revitalised the field and brought about a new generation of algorithms. The focus of this algorithm was on edge removal, ranking edges based on their "edge betweenness" value. Edge betweenness is defined as the ratio of the number of shortest paths between any two nodes that contain that edge and the total number of shortest paths in the graph. The assumption is that the edges with the highest betweenness value are the edges most between communities. This algorithm is still part of the state-of-the-art today for its quality of partitions, however the algorithm is quite time-consuming, requiring weight recalculations after every edge removal. Two years after this publication, Girvan and Newman revolutionized the field again with the introduction of modularity (Q) [16]. Modularity is also still very commonly used today as the primary measure of quality of a community partition. Modularity determines quality by comparing the number of edges between nodes in a community within a partition to the number of edges between nodes in communities within a random partitioning of nodes.

Shortly after the publication of modularity, optimization of modularity became a focal point of many algorithms. Specifically noteworthy is the algorithm presented by Clauset et al. [21]. A common critique of the GN algorithm is that it is computationally intensive with a running time of $O(n^2m)$ or $O(n^3)$ if $m \approx n$ where m is the number of edges and n is the number of nodes in a graph. The algorithm presented by Clauset et al. uses greedy modularity optimization in order to improve performance times on larger graphs. Their algorithm merges nodes together into communities that maximize their modularity. These merges are tracked, and a tree is created, similar to hierarchical clustering. Their algorithm boasts a near linear runtime of roughly $O(n \log^2 n)$ but with worse partition quality than GN. This introduced the modularity optimization approach to community detection, which remains a key part of the field. The most well known of these is the Louvain algorithm, which offers high quality community partitions in a time effective manner [22]. Louvain starts by merging together nodes to form communities that maximize modularity. It then creates a new graph where each node represents a community and each edge is weighted by the number of edges in the original graph that connect the two communities. These are then merged again using a weighted modularity calculation, and the steps are repeated until it is impossible to improve modularity any further. This algorithm is extremely fast and able to be run on massive weighted graphs, making it a popular choice.

Modularity optimization has been critiqued for struggling with "resolution limit". Resolution limit occurs when an algorithm fails to detect small communities within a much larger network [23]. There are various algorithms

that are modularity independent or combine modularity with other metrics, such as label propagation and Walktrap algorithms [17], [24]. Label propagation works by labelling a few nodes in the base graph and then labelling unlabelled nodes based on the labels of their neighbours. This is an extremely quick algorithm that generates good network partitions that are not quite at the quality level of other algorithms. Walktrap uses random walks on a graph combined with modularity to generate state-of-the-art partitions in a somewhat slow manner. Walktrap is based on the idea that when performing random walks on a graph, the walk will get "stuck" within communities, visiting those nodes in the community much more frequently when stuck. There have also been alternative quality measures proposed, such as modularity density (D) [25]. Modularity density aims to address the resolution limit problem by accounting for differing community sizes. It does this by comparing the average degree inside the community to the average degree to outside the community. However, modularity density adds complexity with the addition of a parameter λ needing to be tuned for good results.

The balance between speed and quality of partition is difficult to strike in community detection. The use of genetic algorithms aims to provide both by combining efficient, broad search space discovery using crossover and local improvements through node mutation. The first group to apply the genetic approach to community detection was Tasgin et al. [26], [27]. In their attempt they define a few key aspects that would become standard choices in designing a genetic approach, however the algorithm is still a modularity optimization algorithm. They develop an initial population using a simplified label propagation algorithm [24]. They use label-based partition representation where chromosomes are a list where an index indicates the node and the value at that index indicates the community it is a member of. The representation of a partition is extremely important for the performance and design of the algorithm. The crossover and mutation operations depend somewhat upon this choice. In this case, the operations are quite simple. For crossover, a portion of node labels from one chromosome is applied to the same nodes in a different chromosome. In mutation, some nodes are granted the label of a neighbouring node in the graph. This is a very simple approach to genetic design which works alongside existing techniques. While an interesting concept, it does not make any leaps. Soon after this algorithm was published, Clara Pizzuti published the GA-Net algorithm, a large step from the Tasgin algorithm [28]. GA-Net is distinct due to the novel objective function community score (CS) as well as the use of locus-based chromosome representation which are both popular in genetic community detection today [29]. Locus representation has values at indices representing a node which the index node is in a community with. The advantage of this approach is dynamic changes in the number of communities in the partition, however this can massively broaden the search space due to each index being able to have more values than the number of communities, as is the case with label-based representation. Label-based representation usually has only a set number of communities from population generation, but has a much smaller search space. GA-Net also applies uniform crossover where a binary mask is generated indicating which parent the child which take the value of at each index. This is done to maintain safe solutions where all edges required by the representation exist. The majority of genetic approaches use these building blocks in some form. They also employ neighbour mutation similar to that of Tasgin's algorithm. There are some algorithms such as one proposed by Gong et al. which employ modularity density [25], [30]. Though this can create superior results, the algorithms are complicated by the need to tune the modularity density parameter, λ which is required in their modified modularity density equation D_λ. GA-BCD adopts some of these foundational design choices, but opts for alternative mutation with random choice mutation and multipoint crossover. It most notably employs the objective function Z described in Eq. 5.

Some genetic algorithms choose to alter their structure while applying familiar techniques. Gong et al. presented one such approach, which applies hill-climbing to the problem [30]. The hill climbing structure ensures that each generation improves upon the previous generation. It does so by only accepting a child if its fitness is higher than its parent. While this approach does increase speed of convergence, it has two major downfalls, the first of which is efficiency. This approach requires constant fitness calculations to be performed for each mutation which, depending on the time required to calculate fitness, can drastically increase time taken. The second issue is a tendency to find local maxima and not properly explore the search space. As a response to the second problem, Shang et al. proposed their MIGA algorithm [31]. The MIGA algorithm applies the simulated annealing methodology to maintain the quicker convergence while addressing local maxima by adding a probability to accept a child with worse fitness in the interest of greater exploration of the search space. This algorithm also employs modularity as the objective function to simplify parameters. Jia et al. proposed a differential evolutionary algorithm, which differs in structure and mutation methodology from standard algorithms [32]. In differential evolution all solutions are mutated and crossed over with each child being directly compared to its parent, the one with the greater fitness remains in the population while the other is removed. They employ modularity as their objective function in this case and use a mutation method involving vector operations known as Rand/I followed by a clean-up protocol to generate strong candidates more consistently than simple random mutation.

3 Solution

Within this section, all aspects of designing a genetic algorithm are covered, presenting possible choices for each step and explaining the pros and cons of each. The impact and usefulness of select techniques is assessed by being tested on the popular Dolphins and Zachary's Karate Club graphs [2], [4]. Finally, a hybridized version of the GA-BCD algorithm is presented.

3.1 Genetic Representation

The two main options for genetic representation are label-based and locus-based representation. Label-based repre-

sentation comes from the original genetic algorithm for community detection [26]. It is still very commonly used, being employed in many prominent genetic approaches, including GA-BCD [18], [30], [31], [32]. Within label-based representation, each node is granted an index within a list. The value at each of those indexes indicates community membership. For example, a partition $\boldsymbol{C}$ containing 3 communities $\{c_1, c_2, c_3\}$ where $c_1 = \{1, 4\}, c_2 = \{3\}, c_3 = \{2, 5\}$ would be represented by the vector $\boldsymbol{k} = [1, 3, 2, 1, 3]$ (see Fig. 2).

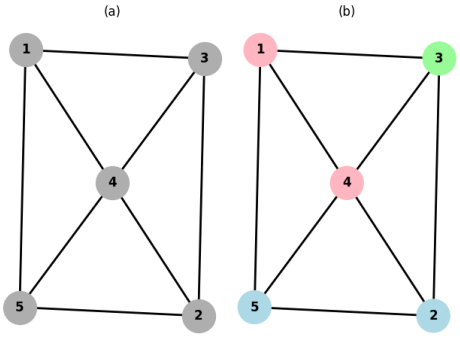


Fig. 2. Graph (a) is an unlabelled graph with the same structure as defined above. (b) is a partitioned version of that graph with colour indicating community.

The locus-based representation, introduced to the field by Pizzuti, represents communities by saying what node is in the same community as the node at that index [28]. This approach is roughly as common as the label-based approach being used widely. Returning to the graph depicted in Fig. 2, the same partition $\boldsymbol{C}$ would be represented instead by $\boldsymbol{k} = [4, 5, 3, 1, 2]$.

Label-based representation is usually chosen for its smaller search space. In label-based representation, each index is only able to take k values where k is the number of communities in a partition. This creates a search space of $O(k^n)$ where n is the number of nodes in the graph. Within locus-based representation, each index can take on the value of any node and therefore has a maximum spacial complexity of $O(n^n)$. The spacial complexity of label-based representation tends to lead to faster convergence due to k usually being far less than n.

Locus-based representation is usually chosen for its flexibility within graphs where the number of communities is unknown. This is due to the number of communities being able to change dynamically. Take for example the chromosome $\boldsymbol{k} = [2, 3, 1, 5, 4]$, here $\boldsymbol{k}$ has 2 communities $c_1 = \{1, 2, 3\}$ and $c_2 = \{4, 5\}$. By changing the value at index 3 to node 5, communities 1 and 2 are merged into a single community $c_1 = \{1, 2, 3, 4, 5\}$. Communities can also be split similarly. With label-based representation, the chromosome will stick with the specific number of communities it is granted after population generation. These drastic structural changes can be good for exploring a broader range of solutions while maintaining some existing community structures. This can, however, make it difficult to make fine adjustments once near a maximum.

3.2 Initial Population

When running a genetic algorithm, the results can be highly sensitive to the quality of the initial population. Methods of population creation aim to strike a balance between starting with a chromosome of high initial fitness value and having a wide enough range of chromosomes to properly explore the search space.

The simplest method is to generate the initial population almost completely randomly. This method is much more geared toward exploration than alternatives. This is the method used by GA-BCD as well as many other prominent algorithms. Within this approach, chromosomes must be "safe" to be accepted into the population. Only safe chromosomes are added to prevent extremely poor or impossible partitions from being in the initial population. There are varying interpretations of safe chromosomes. In some implementations, it is simply that nodes in a community must be connected in the base graph. This is the notion used by GA-BCD [18]. This is the most random approach to population initialization used. Others employ a guided initialization such as in GA-Net [28]. In GA-Net an initial population is randomly generated and then repaired to prevent impossible chromosomes. GA-Net employs a locus-based approach so indexes where the value is a node that is not connected to the index node have their value replaced with a neighbouring node.

A wide range of algorithms are used for population initialization. Some genetic algorithms use simple random walks to create populations based on the same heuristic of the Walktrap algorithm [17], [33]. A much larger amount use a simplified label propagation initialization [24], [30], [32]. In this initialization, all nodes are granted a random initial community label. Then random nodes are selected and grant all neighbours the same community label, this is done αn times where α is a scaling factor and n is the number of nodes. This approach creates much stronger initial populations but can lead to a less diversity. This can also be problematic in densely connected graphs. This does introduce some variance in the number of communities in the solution, adding in some flexibility for graphs of unknown community structure.

When tested on the dolphins graph 100 times each, the label propagation initialization outperformed the safe random initialization by 35% with no difference in time taken. The partitions were scored by their modularity. (see Table 1). As the improvement of performance is quite distinct, all performance tests from now on will employ this.

TABLE 1
Performance of Differing Population Generation.

Safe Random Generation (Original)	**Fitness**	**Modularity**
Mean	3.05×10^6	0.082
Median	2.95×10^6	0.086
Label Propagation Generation	**Fitness**	**Modularity**
Mean	3.34×10^6	0.111
Median	3.19×10^6	0.108

3.3 Objective Function

The objective function provides a metric to assess the quality of members of the population. By maximizing this function, the genetic algorithm creates strong solutions. This is the guiding force of the algorithm, and as such it is extremely important to select one that accurately assesses the quality of a partition.

Modularity was the most revolutionary and to this day most common metric for evaluating community partitions [16]. Modularity uses the difference between the number of all edges that are between members of a community and the number of all edges that have at least one end in the community. The goal of modularity is to reward partitions that minimize the number of edges between communities. Modularity can fail to recognize small communities within much larger graphs due to a problem known as resolution limit [23]. This can be damaging to the partitions of natural data, as not all communities are of similar size. This can be outweighed within genetic approaches due to the simplicity of the function and strong results otherwise. It is defined using A, which is the adjacency matrix, k_i which is the degree of node i, and $\delta(c_i, c_j)$ which is 1 if the nodes i and j are in the same community and 0 if they are in different communities. In some implementations, a parameter γ is added to address the resolution limit issue [34] [23].

$$Q = \frac{1}{2m}\sum_{ij}(A_{ij} - \gamma\frac{k_i k_j}{2m})\delta(c_i, c_j) \tag{1}$$

To address the resolution limit issue, modularity density was proposed [25]. Modularity density improves upon the performance of modularity in graphs which contain much smaller communities relative to the total graph size by taking into account the number of nodes in a community. This function can outperform modularity in heterogeneous graphs, however, to get the strongest performance the parameters must be tuned. It is defined using average degrees of nodes as opposed to total edges as in modularity. It is defined using $L(C_i, C_j)$ which is the number of edges between nodes in communities i and j, as well as $|C_i|$ which is the number of nodes in community i. The calculation is done for all communities in a graph $C_1...C_m$. There is also a parameter λ which changes the ratio of the weighting between intra-community edges and inter-community edges, with 1 being fully intra-community edges and 0 being fully inter-community edges.

$$D = \sum_{i=1}^{m}\frac{2\lambda L(C_i, C_i) - 2(1-\lambda)L(C_i, \overline{C_i})}{|C_i|} \tag{2}$$

Community score works using sub-matrices of the adjacency matrix of a graph [28]. Each sub-matrix represents a community. It uses the mean number of edges within the community as well as the total number of edges within a community to reward densely connected communities. Each community is represented by a sub-matrix $S(I, J)$ where I is a subset of the rows in the adjacency matrix, and J is a subset of the columns. These are the rows and columns that represent the nodes in the community within in the adjacency matrix. The mean value of a row i is defined to be $a_{i,J}$, this is also the ratio of nodes within the community that node i has an edge to and nodes it does not have an edge to. Finally, the volume (v_s) of a sub-matrix S is the sum of values within the sub-matrix. The parameter r is used to change how strongly dense matrices are weighted. The calculation of CS for the sub-matrices $S_1...S_m$ representing communities $C_1...C_m$ is shown in Eq. 3. CS differs from Q and D by not taking into account the number of inter-community edges. While this may be more effective in some cases, careful tuning of the parameter r is required to achieve good results, especially in graphs where community structure is less well-defined.

$$CS = \sum_{c}^{m}(\frac{\sum_{i\in I}(a_{ij})^r}{|I|} \times v_{S_c}) \tag{3}$$

The GA-BCD paper optimizes an original function known as group similarity density(Z) [18]. Group similarity density employs similarity equations which determine how similar any two nodes are using the neighbours of each node. The selection of a similarity equation can drastically change the performance of an algorithm and as such increases the complexity of applying Z.The goal is to form communities containing the most similar nodes. The function is defined using $S(a, b)$ which is the similarity of node a and node b, C_i the community i, and k the number of communities. The function is split between the similarity density of a single community in Eq. 4 and the full group similarity in Eq. 5.

$$SD_i = \sum_{a,b\in C_i} S(a, b) \tag{4}$$

$$Z = (1 + \frac{1}{\sqrt{k}})\prod_{i=1}^{k} SD_i \tag{5}$$

The structure of this equation can cause calculate time to increase when communities are larger. This is because the similarity of every pair in a community must be calculated, which increases exponentially with community size. The structure of the equation also prevents it from being applied to a genetic algorithm with a flexible amount of communities. Due to the multiplication of individual community densities, a higher number of communities with lesser individual similarity density will likely be chosen over fewer, stronger, communities that would better fit the graph. Consider Table 2 where GA-BCD has been tested on the Dolphins graph with the inputs of two communities and eight communities [2]. The time taken for two communities is 13.5 times as long and produces a Z of 0.1% that of eight communities, but greatly outperforms eight in terms of modularity. It should be noted that running Louvain and Walktrap algorithms on the same graph produces a partition of eight to ten communities [17], [22].

TABLE 2
The Performance of GA-BCD on Dolphins Depending on Number of Communities

# of Communities	Time (s)	Z	Q
2	63.57	5,533	0.300
8	4.86	4.3×10^6	0.085

In order to assess the difference of each objective function on the performance of GA-BCD, each have been run 100

times on the Zachary's Karate Club testing graph with the original structure of GA-BCD [4]. This is a very small graph with only 34 nodes labelled into two communities. The results were assessed using normalized mutual information, which compares how similar two partitions of the same graph are. The partitions are compared to the ground truth labelling of the graph.(See Table 3)

TABLE 3
The Performance of Objective Functions Assessed by NMI on Zachary's Karate Club

Objective Function	Avg. NMI
Modularity (Q)	0.7168
Modularity Density (D)	0.6973
Community Score (CS)	0.1701
Group Similarity Density (Z)	0.1668

From the table it is clear that D and Q are quite strong performers on smaller graphs such as this one where CS and Z struggle. This assessment is not comprehensive and only serves to display differences on a small graph with poorly tuned functions. This is, however, indicative of the simplicity of applying each of the objective functions to graphs as well as the flexibility of their application range.

3.4 Selection

The process of selection is how the genetic algorithm enforces the objective function above. It culls the poorest solutions and ensures that only the strongest members of a population move on to create children.

The vast majority of genetic algorithms for community detection, including GA-BCD, apply the same selection strategy. $\mu + \lambda$ selection uses the objective function to rank the quality of different chromosomes and generate a new generation of children from the best and keeping the strongest parents. In $\mu+\lambda$ selection, the top μ chromosomes are selected as the elite of the generation. These chromosomes then create λ children through the processes of mutation and crossover. Mutation and crossover are performed probabilistically on the children, occasionally allowing for copies of a parent to be added to the population unchanged The selected chromosomes or parents have a probability of being chosen as the parent of a child proportional to their fitness with the fittest solutions having the best chance. The goal of $\mu + \lambda$ is to increase the likelihood that generations will strictly improve over time, while also ensuring the search space is properly explored by mutation and crossover functions.

The largest issue with this approach is the need to parameterize μ. Too high of a μ value can slow convergence on a strong solution by allowing poorer solutions to influence the next generations. Having too low of a μ value can prevent the search space from being properly explored, getting stuck at a single local maximum. This approach also allows for chromosomes with poorer fitness to be created and added to the population. This can be prevented using the hill-climbing or differential evolution approaches [30], [32]. The strictness of these methodologies massively disincentives exploration and can prevent poor initial populations from reliably reaching high quality solutions. This is done through only allowing stronger solutions into the next generation. This can slow down an algorithm as children need to be repeatedly generated and have their fitness assessed many times.

3.5 Mutation

The mutation of chromosomes introduces fine-tuning of children allowing similar, but novel solutions to be created from strong performing chromosomes, essentially performing a local search. By making small adjustments to chromosomes, they can incrementally improve in a more reliable way than with crossover, described in Sec. 3.6.

The GA-BCD algorithm employs a random mutation strategy which is commonly used in the field [18]. In random mutation, each chromosome has a set of nodes selected at random which are then moved to a different community. In their paper, they claim that the design of the algorithm, employing similarity density from Eq. 4, provides a separate local search function, well suited to the goal of mutation. The use of random mutation can be seen as quite pointless, exploring many obviously poor solutions leading to slowing convergence, as pointed out by Pizutti [28].

In order to tighten the search space, some approaches use neighbour based mutation. Within neighbour based mutation, a randomly selected node takes on the community of one of its neighbours. This can prevent very well-connected communities from breaking apart meaninglessly, while allowing nodes that may be more well-connected to a separate community to have a higher chance of moving to that community. It also works well in connection with the label based initialization from Sec. 3.2. This can be seen as similar to label propagation algorithms, which select the label of a node based on the majority of its neighbours. The neighbour mutation approach emulates this but with probabilistic changes allowing for more unique solutions.

TABLE 4
The Average Z and Q of Mutation Methods on Dolphins Graph

Mutation	Avg. Z	Avg. Q
Random	2.8×10^7	0.132
Neighbour	1.1×10^7	0.304

In Table 4 GA-BCD with both random mutation and neighbour mutation were run 50 times each on the Dolphins graph [2]. Neighbour mutation clearly generated superior results in terms of modularity but less so for group similarity density. This is because the neighbour mutation better maintains the label propagation structure from the initial population generation and explores the search space in a more reserved manner.

3.6 Crossover

Crossover acts as a global search, massively altering the structure of chromosomes to explore the search space broadly. This is done by combining two parent chromosomes to create a child containing aspects of both.

Single point crossover is the simplest form. In it, a single pivot point is chosen, where all values before that point are

taken from one parent and after are taken from another. The single point crossover approach does not work well for genetic community detection as it can merge two communities into a single one, especially in the case of two community detection. For example, the parent chromosomes [1,1,1,2,2,2] and [2,2,2,1,1,1] are the same partition, a single point crossover could create [1,1,1,1,1,1]. This also makes it much more likely for segments of the chromosomes to remain completely unchanged by crossover, leading to poor exploration of the search space. This is why the majority of algorithms use multipoint or uniform crossover, with GA-BCD using multipoint crossover. Both types of crossover work similarly, splitting up what is taken from one chromosome and granted to another in an attempt to break up the structure more. Multipoint crossover works the same as single point crossover, just with more pivot points. This serves to break up the structure somewhat but, depending on the number of points chosen, can still maintain large sections of unbroken structure. This is not necessarily positive, as nodes are assigned to indexes without any knowledge of community structure. Uniform crossover completely destroys the structure of the chromosome by randomly selecting which parent will be the giving parent for each index. This is done by creating a binary mask as shown in Table 5 and using that to dictate the giving parent. This allows further global search than multipoint crossover for lower counts of pivot points. In cases where the count of pivot points is high, they are effectively similar.

TABLE 5
Uniform Crossover

Parent 1	2	3	3	1	1	2
Parent 2	1	1	2	1	3	3
Mask	0	1	0	0	1	0
Child	2	1	3	1	3	2

3.7 Hybridized Algorithm

In this subsection, a novel hybridized algorithm is presented, Similarity-based Genetic Algorithm for Community Detection (S-BGACD). The previous sections addressed all aspects of designing a genetic algorithm for community detection, including preliminary testing. This subsection discusses each of these aspects and describes the reasoning behind the selection of each option.

S-BGACD uses label-based genetic representation for the simplicity and smaller search space. Label-based representation removes the requirement of translating the representation into usable communities for assessment present in locus-based representation. Label-based representation also works on a set number of communities, allowing for much easier comparison between S-BGACD and GA-BCD. The advantages of locus-based representation such as being able to search more broadly are accounted for in other aspects of the hybridized design.

S-BGACD uses label propagation based generation. This generation far outperformed random initialization in preliminary testing and will be far more time efficient on larger graphs. The performance improvement is due to the stronger initial population where neighbours are far more likely to be in the same community, which is very important for the objective function. The time efficiency is due to not needing to check for connectedness between nodes in the initial community creation, a relatively time inefficient process. The performance can struggle on very densely connected graphs, where extreme interconnectedness leads to too many nodes being granted the same label, removing the ability to create a nuanced initial population. This also is impacted by the inefficiency of Z on communities containing far more nodes. To counteract this, a second parameter, β, has been added which will determine the proportion of neighbours granted the label. A beta value of 0.1 would mean that every time a node is selected to grant its label to neighbours, it will grant 10% of its neighbours the label. This removes some initial quality, but allows for the algorithm to run far more efficiently.

S-BGACD continues using Z as its objective function. Algorithms employing $Q, D,$ and CS have been thoroughly explored previously. The aim of this paper is not to create the strongest possible genetic algorithm, but rather to understand and improve upon GA-BCD. By using Z, GA-BCD and S-BGACD are much more easily comparable.

S-BGACD keeps the original $\mu + \lambda$ selection as it is by far the most popularly used approach with simple and effective methodology, allowing for time efficient algorithms with good results. This selection option allows for proper exploration of the search space and sufficient exploitation of strong solutions.

Neighbour mutation is used as it performed very well in preliminary testing and is more commonly used in the strongest performing genetic algorithms in the field. Only allowing for mutations to take on the labels of neighbouring nodes decreases the search space by a large amount and prevents obviously poor mutations from taking place.

Finally, S-BGACD uses uniform crossover due to its simplistic scalability, not requiring any parameter tuning for wide exploration. Multipoint crossover performs similarly, but requires proper parameter tuning for each graph.

For a direct comparison between the design GA-BCD and S-BGACD see Table 6.

4 Results

Within this section the performance of S-BGACD and GA-BCD are assessed in comparison to the Louvain and Walktrap algorithms as well as Q-BGACD and D-BGACD, modified versions of S-BGACD that use modularity and modularity density as their objective functions. The algorithms have been tested on graphs of varying size to assess both quality of results and time required for each algorithm.

4.1 Testing Environment

The algorithms were all tested on an 11th Gen Intel® Core™ i5-1135G7 @ 2.40GHz. All algorithms are coded in Python 3.11.3. Both S-BGACD and GA-BCD use original code. The cdlib python package was used for the Walktrap algorithm with all other comparison algorithms as well as graph implementation using the NetworkX package [35], [36].

TABLE 6
Comparison of GA-BCD and S-BGACD Design

Algorithm	Genetic Representation	Initialization	Selection	Fitness	Mutation	Crossover
GA-BCD	Label-Based	Random	$\mu + \lambda$	Z	Random	Mult-point
S-BGACD	Label Based	β Label-Prop	$\mu + \lambda$	Z	Neighbour	Uniform

4.2 Testing Graphs and Algorithms

GA-BCD and S-BGACD were run for 100 generations with a population size of 100 and a time limit of 48 hours. The input number of communities was set to be the number of communities in the Louvain Solution. These algorithms were tested on the datasets described in Table 7. These datasets were chosen as they contain a wide range of node counts, edge densities, and clustering coefficients.

1) Dolphins [2] - This dataset originates from a study of 62 bottle-nosed dolphins living in a community off of a fjord in New Zealand between 1994 and 2001. The graph is directed, with each edge representing frequent interaction between the dolphins.
2) Net Science [37] - This undirected dataset depicts the structure of the network science field of study in 2006. Nodes in this graph indicate scientists in the field, with edges indicating co-authorship of a published paper.
3) Blogs [3] - This is a directed dataset showing the relationships between blogs within the 2004 US presidential election. Each node is a blog with an edge indicating a hyperlink from that blog to another blog.
4) Facebook [38] - This dataset depicts 10 friendship circles on the social media website Facebook. Each circle is a singular friends list collected from polling, with each node representing a single user and each edge representing the users are friends.

TABLE 7
Testing Graphs

Name	**# of Nodes**	**# of Edges**	**Clustering Coeff.**
Dolphins [2]	62	159	0.308
Net Science [37]	1,491	2,724	0.693
Blogs [3]	1,224	19,025	0.226
Facebook [38]	4,039	88,234	0.606

In order to have a strong understanding of the performance of GA-BCD and S-BGACD they have been compared to state-of-the-art algorithms as well as two additional versions of S-BGACD.

1) Louvain [22] - The Louvain algorithm is an extremely highly regarded algorithm using modularity optimization. It starts with all nodes being part of their own community. Nodes are then merged together based on the modularity of the merged community. These merged communities are then merged again in the same manner. This is repeated until no more modularity improvements can be made.
2) Walktrap [17] - The Walktrap algorithm is another high performance algorithm that uses random walks to generate community partitions. This works off the idea that random walks will get stuck within communities, revisiting the same nodes more than they would between communities.
3) Q-BGACD - This is a modification of S-BGACD that uses modularity as its objective function. Modularity is the most well established metric for assessing a community partition of a graph using the ratio of the density of in community edges in a given partition to the density of in community edges in a random partitioning.
4) D-BGACD - This is a modification of S-BGACD that uses modularity density as its objective function. Modularity density in an attempted improvement upon modularity that uses the average degree of nodes within a community rather than total number of edges in order to better account for graphs with communities of widely differing sizes.

4.3 Experimental Results

The testing algorithms were run on each graph for a maximum of 48 hours. The resulting modularity score of each is shown in Fig. 3 with exact numbers in Table 8. GA-BCD failed to produce any initial partition on Net Science due to the strictness of its safety checking operation. GA-BCD and S-BGACD both were unable to produce a result from the Facebook graph within 48 hours of running.

The initialization of a population is integral to the performance of any genetic algorithm. In S-BGACD a simple label propagation based initialization is used. In this, nodes are randomly labelled, and then a smaller set are chosen to pass their labels on to their neighbours. This has been modified from the traditional method with an additional β parameter to allow for its use on densely connected graphs. This method was chosen because it is both a time efficient choice, and provides a very good starting point for the algorithm. In Table 9 Both algorithms attempted to generate a population of 100 chromosomes on all testing graphs. β has been set to 1 for Dolphins and Network Science and 0.2 for Blogs and Facebook due to their density. As Network Science has 286 connected components, it is impossible for GA-BCD to generate any initial population with less than 286 communities.

In order to assess how well Z worked at guiding a genetic algorithm towards stronger solutions, the genetic algorithms were tested on the Dolphins graphs with the modularity of their best chromosome being calculated at the end of every generation. The results are shown in Fig. 4. This figure serves to demonstrate the ability of S-BGACD to

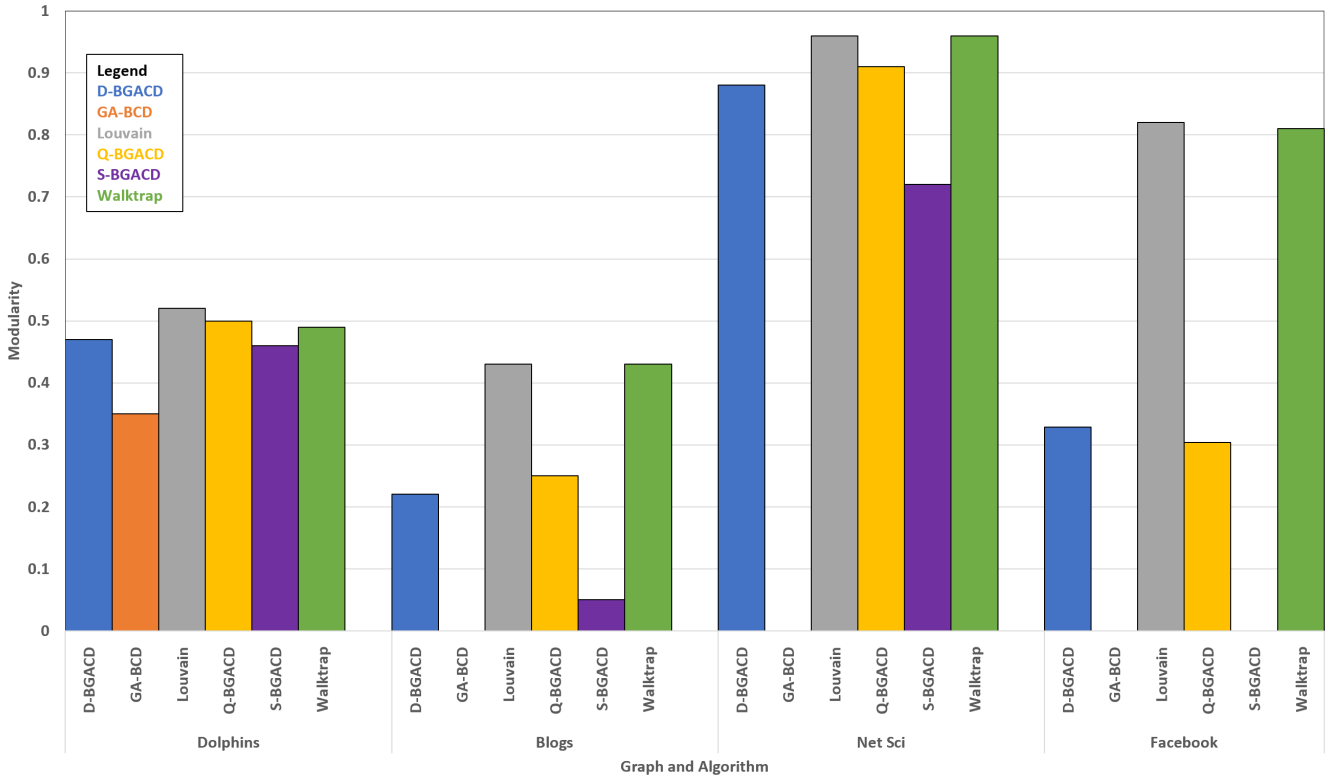


Fig. 3. The modularity scores of all testing algorithms on all testing graphs.

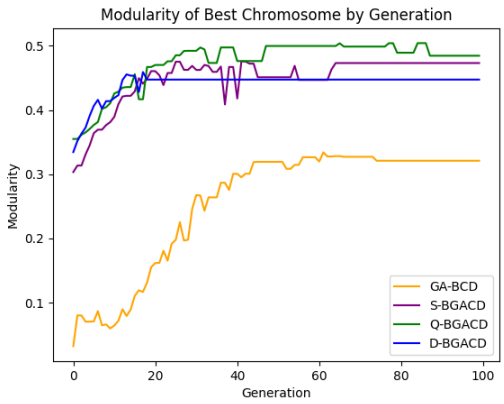


Fig. 4. The modularity score of the best chromosome in a generation for all genetic algorithms on the Dolphins graph.

approach higher modularity values with the same objective function while also showing the shortcomings of GA-BCD.

5 Evaluation

Through recreating the GA-BCD algorithm, some discrepancies were found in the results of the GA-BCD paper. In tests performed on similar graphs to those used in the paper, GA-BCD far under-performed stated results and was extremely time inefficient compared to comparison algorithms. The hybridized algorithm, S-BGACD, was able to improve on the results of GA-BCD greatly due to structural changes. Still, S-BGACD was also unable to approach the results of the Walktrap and Louvain. The main fault within both of these algorithms is the calculation of Z. The design of Z leads to extreme time inefficiency as well as poor results.

5.1 Group Similarity Density Z

In networks such as the Facebook network, where there are many nodes with fewer communities, the time inefficiency is massively consequential. This comes from the exponential increase in the number of calculations required based on the size of communities. A community with 10 nodes will require 45 similarity calculations while one with 100 will require 4950 calculations. With each calculation taking roughly 0.0001 seconds, this can lead to massive increases in time to calculate the similarity density for a single community. This means that the most time efficient partitions may be far worse than more well suited partitions requiring larger communities. This time issue is exacerbated by the choice to use Python for implementation. In cases with many iterations of comparisons, such as with genetic algorithms, writing in a compiled language can allow for far more efficient execution. The GA-BCD paper also employs parallel computing to cut down on run times further, which has not been done here. The time inefficiency of Z can be best seen in the blogs results in Table 8. Here, S-BGACD took roughly 5x as long as GA-BCD. This is entirely due to the number of communities found by each algorithm. S-BGACD merged many initial communities together to create a total of 14 communities while GA-BCD maintained 122 communities.

Z is also a poor judgement of the quality of a partition. This is shown by the massively improved performance

TABLE 8
Modularity and Timings of All Algorithms on All Test Graphs

Graph	Algorithm	Time (s)	Q
Dolphins	GA-BCD	16.18	0.35
	S-BGACD	23.84	0.46
	D-BGACD	16.27	0.47
	Q-BGACD	5.76	0.5
	Louvain	0.005	0.52
	Walktrap	0.003	0.49
Blogs	GA-BCD	1,338	-3.6×10^{-5}
	S-BGACD	6,557	0.05
	D-BGACD	4,681	0.22
	Q-BGACD	4,453	0.25
	Louvain	0.16	0.43
	Walktrap	0.22	0.43
Net Sci	GA-BCD	N/A	N/A
	S-BGACD	2,447	0.72
	D-BGACD	2,650	0.88
	Q-BGACD	2,183	0.91
	Louvain	0.12	0.96
	Walktrap	0.07	0.96
Facebook	GA-BCD	N/A	N/A
	S-BGACD	N/A	N/A
	D-BGACD	66,158	0.33
	Q-BGACD	57,984	0.30
	Louvain	0.81	0.82
	Walktrap	2.05	0.81

TABLE 9
Population Initialization Statistics

Algorithm	Dataset	Time(s)	Max. Q	Avg. Q
GA-BCD	Dolphins	0.048	0.099	-0.016
	Net Sci	N/A	N/A	N/A
	Blogs	456.8	0.001	-0.001
	Facebook	6.670	0.0004	-0.0003
S-BGACD	Dolphins	0.049	0.286	0.135
	Net Sci	1.515	0.454	0.360
	Blogs ($\beta = 1$)	1.395	0.182	0.080
	Facebook ($\beta = 1$)	6.438	0.546	0.365
	Blogs ($\beta = 0.2$)	1.513	0.038	0.022
	Facebook($\beta = 0.2$)	6.049	0.095	0.065

when using Q and D as the objective functions in place of Z in S-BGACD. D and Q functions had more consistent improvement throughout execution in comparison to Z which was extremely susceptible to local maxima as can be seen in Fig. 4 where it plateaued far below the modularity of stronger algorithms. Due to the structure of Z, a single community with lesser similarity can drastically reduce the fitness of a partition, making it difficult for large exploratory changes in the partition to be made. Graphs with communities of varying sizes are not appropriately scored as well. Z prefers many communities of similar similarity density over more accurate but varying sizes of community. This also makes it extremely reliant on the number of communities given as input. Comparing the performance of GA-BCD and S-BGACD with the objective function Q on blogs, both receiving the same input parameters. S-BGACD with Q was able to create fewer communities of extremely varying similarity density, ranging between 6.8 and 23,037 while GA-BCD only created communities with densities between 7.85 and 26.5. S-BGACD with Q finished with a far higher Q, a much more widely accepted quality metric than Z.

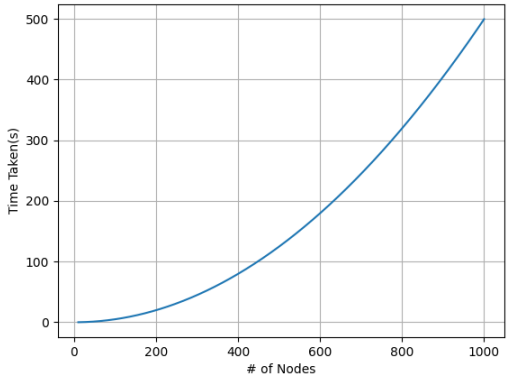


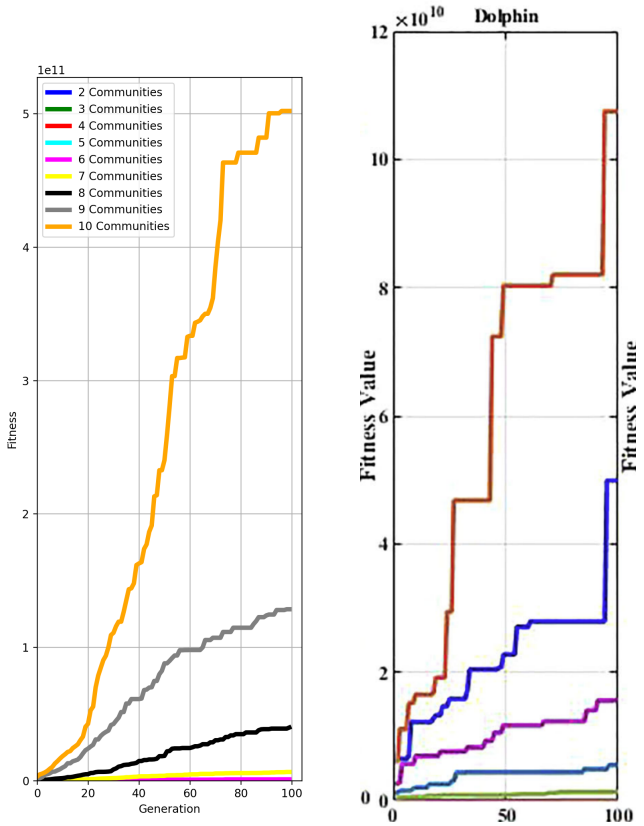
Fig. 5. Time Taken to Calculate Similarity Density for a Community Based on Size

5.2 Initialization

The random safe initialization of GA-BCD is also incredibly time-consuming, especially on sparsely connected graphs like Network Science, where random initialization failed to generate a population in 6 hours of running. Some nodes only have a singular connection, as such the random initialization followed by a safety check can make it difficult to create a safe chromosome. Network Science is especially difficult due to it containing hundreds subgraphs disconnected from the rest of the graphs. The label propagation based initialization far outperformed the random safe initialization in the quality of initial population, as can be seen in Table 9. All initial populations generated by label propagation have some level of community structure represented, while random initialization has none. This could be indicative of too many very similar communities within the population, however that is unlikely to be the case as in Figure 4 the algorithms employing this initialization still improve at roughly the same rate before plateauing. Their initial quality allows them to plateau much higher than random initialization.

6 Conclusion

The field of social network analysis is rapidly growing. A large part of this growth is the community detection problem, where social networks represented as graphs are partitioned into closely connected communities. As the size of these networks and the number of applications grow, there is a need for efficient and high quality solutions. There



(a) This Implementation (b) GA-BCD Paper

Fig. 6. Two graphs depicting the fitness change of GA-BCD on Dolphins for the implementation from this paper (a) as well as the original paper (b).

are many approaches to this problem with widely varying methodology. A relatively new method is applying genetic algorithms to the problem. These algorithms use evolutionary principles to create a high quality solution through survival of the fittest. Within this paper, a promising new algorithm known as GA-BCD was assessed, step by step, in order to improve upon the algorithm in the S-BGACD hybrid algorithm. In this process, massive discrepancies with the GA-BCD paper results were found, throwing doubt on the algorithm.

6.1 Impossible Results

The algorithm implementation was arrived at after thorough testing of GA-BCD and comparison to the fitness curves provided in the GA-BCD paper. This implementation of GA-BCD recreates all fitness curves that are possible in the GA-BCD paper (see Figure 6). It is important to note that with more communities, a difference of 3-5$\times$ between the best partitions is normal due to the product operator in Z.

This code has been recreated from the stated design in the GA-BCD paper. No original source code for GA-BCD has been publicized, and the paper presents some impossible results. Specifically, on the Net Science dataset, the paper claims results for partitions of 2 to 10 communities. This is impossible with their stated algorithm design, as there are 286 connected components within the graph, making it impossible to have any fewer than 286 communities where every node is connected to every other node in the community. There are also some large discrepancies with their graphs. First is the scale of the modularity graph, which claims much higher modularity than is possible on some graphs, see Fig. 1 and Table 8. The most egregious issue is the use of ARI and NMI metrics when assessing results. ARI and NMI are used to understand how similar any two partitions are, these are applied mainly to algorithmically gathered partitions to compare them to ground truth labellings. There are no ground truth labellings available for any sample dataset, other than in some ways Facebook. The Facebook dataset is made of 10 combined friends lists, this could be seen as a ground truth assignment, however, some nodes appear in multiple friends lists. There is no mention of what they are using as the ground truth assignment. The partitions are not being compared to any of the algorithms used to test, as no NMI or ARI values are 1 which would be expected if the same 2 partitions are compared. This combined with the large gulf between results could indicate some form of deception.

It is possible, though unlikely, that this paper's implementation differs from the implementation in the GA-BCD paper. The GA-BCD paper is somewhat opaque with parameterization. The number of nodes mutated or points for multipoint crossover are not stated. This could have some impact if these numbers are poorly parameterized, but this accounts for a very small difference in performance. The GA-BCD paper also has a poor description of how its random initialization is actually implemented. It is not specified whether nodes must be connected in the community subgraph or the complete graph. More importantly, it is only stated that nodes are only kept in the same community if they are connected. This can imply multiple different implementations, such as complete randomness, employed by this implementation, where nodes not connected to the rest of the community are then randomly reassigned to a different group. It could be that all node connections are checked and are moved to different communities, maintaining these groupings as they change community. This would be much more similar to label propagation based initialization and provide far improved initial populations. This still does not account for the claimed ability of GA-BCD to initialize on Network Science with 2-10 communities. It is highly unlikely that Z has been improperly implemented, due to the similar Z scores between this paper's implementation and those claimed in GA-BCD. The implementation in this paper is most likely not the cause for the massive result discrepancies.

6.2 Future Work

There are many opportunities for future work dealing with genetic algorithms in community detection. There is much promise in the field, with many successful algorithms outperforming traditional methods such as GA-Net [28]. More specifically, there are still many unexplored opportunities with S-BGACD. An implementation in a lower level compiled language would allow for more efficient calculation, this could be combined with parallel computing to bring execution times down to the level of the state-of-the-art. Similarity should not be abandoned as a method of assessing these community partitions. There is ample space to

improve on Z through redesigning the metric to be more efficient and less susceptible to local maxima. Similarity has already been applied effectively for non-genetic community detection [19]. The field is promising, and the discrepancies found in this paper should not be a deterrent to further exploration.

REFERENCES

[1] S. Wasserman and K. Faust, "Social network analysis: Methods and applications," 1994.

[2] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson, "The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations," *Behavioral Ecology and Sociobiology*, vol. 54, no. 4, pp. 396–405, Sep 2003. [Online]. Available: https://doi.org/10.1007/s00265-003-0651-y

[3] L. A. Adamic and N. Glance, "The political blogosphere and the 2004 u.s. election: divided they blog," in *Proceedings of the 3rd International Workshop on Link Discovery*, ser. LinkKDD '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 36–43. [Online]. Available: https://doi.org/10.1145/1134271.1134277

[4] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of Anthropological Research*, vol. 33, no. 4, pp. 452–473, 1977. [Online]. Available: http://www.jstor.org/stable/3629752

[5] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3, pp. 75–174, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0370157309002841

[6] B. Krishnamurthy and J. Wang, "On network-aware clustering of web clients," in *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2000, pp. 97–110.

[7] P. Reddy, M. Kitsuregawa, and P. Sreekanth, "0002, ssr (2002). a graph based approach to extract a neighborhood customer community for collaborative filtering." DNIS.

[8] N. Gaumont, M. Panahi, and D. Chavalarias, "Reconstruction of the socio-semantic dynamics of political activist twitter networks—method and application to the 2017 french presidential election," *PloS one*, vol. 13, no. 9, p. e0201879, 2018.

[9] A. Karataş and S. Şahin, "Application areas of community detection: A review," in *2018 International congress on big data, deep learning and fighting cyber terrorism (IBIGDELFT)*. IEEE, 2018, pp. 65–70.

[10] M. Salathé and J. H. Jones, "Dynamics and control of diseases in networks with community structure," *PLoS computational biology*, vol. 6, no. 4, p. e1000736, 2010.

[11] J. J. Bechtel, W. A. Kelley, T. A. Coons, M. G. Klein, D. D. Slagel, and T. L. Petty, "Lung cancer detection in patients with airflow obstruction identified in a primary care outpatient practice," *Chest*, vol. 127, no. 4, pp. 1140–1145, 2005.

[12] N. Haq and Z. J. Wang, "Community detection from genomic datasets across human cancers," in *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2016, pp. 1147–1150.

[13] F. Taya, J. de Souza, N. V. Thakor, and A. Bezerianos, "Comparison method for community detection on brain networks from neuroimaging data," *Applied Network Science*, vol. 1, pp. 1–20, 2016.

[14] Y. Yang, P. G. Sun, X. Hu, and Z. J. Li, "Closed walks for community detection," *Physica A: Statistical Mechanics and its Applications*, vol. 397, pp. 129–143, 2014.

[15] P. Bedi and C. Sharma, "Community detection in social networks," *Wiley interdisciplinary reviews: Data mining and knowledge discovery*, vol. 6, no. 3, pp. 115–135, 2016.

[16] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, Feb. 2004. [Online]. Available: http://dx.doi.org/10.1103/PhysRevE.69.026113

[17] P. Pons and M. Latapy, "Computing communities in large networks using random walks," in *Computer and Information Sciences-ISCIS 2005: 20th International Symposium, Istanbul, Turkey, October 26-28, 2005. Proceedings 20*. Springer, 2005, pp. 284–293.

[18] R. K. Behera, D. Naik, S. K. Rath, and R. Dharavath, "Genetic algorithm-based community detection in large-scale social networks," *Neural Computing and Applications*, vol. 32, no. 13, pp. 9649–9665, Jul 2020. [Online]. Available: https://doi.org/10.1007/s00521-019-04487-0

[19] Y. Pan, D.-H. Li, J.-G. Liu, and J.-Z. Liang, "Detecting community structure in complex networks via node similarity," *Physica A: Statistical Mechanics and its Applications*, vol. 389, no. 14, pp. 2849–2857, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378437110002050

[20] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.122653799

[21] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E*, vol. 70, p. 066111, Dec 2004. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.70.066111

[22] 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*, vol. 2008, no. 10, p. P10008, Oct. 2008. [Online]. Available: http://dx.doi.org/10.1088/1742-5468/2008/10/P10008

[23] S. Fortunato and M. Barthélemy, "Resolution limit in community detection," *Proceedings of the National Academy of Sciences*, vol. 104, no. 1, p. 36–41, Jan. 2007. [Online]. Available: http://dx.doi.org/10.1073/pnas.0605965104

[24] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Phys. Rev. E*, vol. 76, p. 036106, Sep 2007. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.76.036106

[25] Z. Li, S. Zhang, R.-S. Wang, X.-S. Zhang, and L. Chen, "Quantitative function for community detection," *Phys. Rev. E*, vol. 77, p. 036109, Mar 2008. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.77.036109

[26] M. Tasgin, A. Herdagdelen, and H. Bingol, "Community detection in complex networks using genetic algorithms," 2007.

[27] 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*, vol. 108, pp. 87–111, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804518300560

[28] C. Pizzuti, "Ga-net: A genetic algorithm for community detection in social networks," in *Parallel Problem Solving from Nature – PPSN X*, G. Rudolph, T. Jansen, N. Beume, S. Lucas, and C. Poloni, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1081–1090.

[29] Y. Park and M. Song, "A genetic algorithm for clustering problems," in *Genetic Programming 1998: Proceedings of the Third Annual Conference*, J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, Eds. University of Wisconsin, Madison, Wisconsin, USA: Morgan Kaufmann, 22-25 July 1998, pp. 568–575.

[30] M. Gong, B. Fu, L. Jiao, and H. Du, "Memetic algorithm for community detection in networks," *Phys. Rev. E*, vol. 84, p. 056101, Nov 2011. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.84.056101

[31] R. Shang, J. Bai, L. Jiao, and C. Jin, "Community detection based on modularity and an improved genetic algorithm," *Physica A: Statistical Mechanics and its Applications*, vol. 392, no. 5, pp. 1215–1231, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378437112009673

[32] G. Jia, Z. Cai, M. Musolesi, Y. Wang, D. A. Tennant, R. J. M. Weber, J. K. Heath, and S. He, "Community detection in social and biological networks using differential evolution," in *Learning and Intelligent Optimization*, Y. Hamadi and M. Schoenauer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 71–85.

[33] D. Jin, D. He, D. Liu, and C. Baquero, "Genetic algorithm with local search for community mining in complex networks," in *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*, vol. 1, 2010, pp. 105–112.

[34] M. E. J. Newman, "Networks : an introduction," Oxford, p. 224, 2010. [Online]. Available: http://www.dawsonera.com/depp/reader/protected/external/AbstractView/S9780191637766

[35] R. G., M. L., and C. R., "Cdlib: a python library to extract, compare and evaluate communities from complex networks," *Applied network science*, vol. 4, 2019.

[36] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[37] M. E. J. Newman, "Finding community structure in networks using the eigenvectors of matrices," *Phys. Rev. E*, vol. 74, no. 3, 2006.

[38] J. Leskovec and J. Mcauley, "Learning to discover social circles in ego networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper$_files/paper/2012/file/7a614fd06c3254$
$Paper.pdf$