VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Genetic Algorithms for Community Detection in Social Networks

J G Hodges

Supervisors: Hui Ma, Aaron Chen

Submitted in partial fulfilment of the requirements for
Bachelor of Engineering with Honours.

Abstract

Over the last few decades, the problem of identifying communities in social networks has been thoroughly investigated from two key angles, optimization and heuristic. In this paper we propose an optimization based approach using Genetic Algorithms that utilizes a heuristic method for mutation to improve the efficacy and accuracy of evolutionary approaches to community detection. This is achievable as the heuristic can be utilized instead of local search and the individual representation is modified to represent central nodes in a community, which reduces the number of evaluations required. This is evaluated against synthetic and real networks to prove the methods effectiveness at detecting community structure in complex social networks.

Contents

Chapter 1

Introduction

This project studies the problem of community detection and suggests an approach to solving it. In this chapter we will introduce the problem and motivation for solving it, as well as briefly outline the approach and what we aim to achieve, followed by a break down of how the remainder of the report with be organised.

1.1 Overview

The relationships between individuals in a population can be represented as a social network structure, with the individuals as the nodes and the relationships as edges. The communities within this structure are defined based on the topology of the network as a grouping of nodes that are more densely connected (or similar) to nodes internal to the community than those external. The automated identification and analysis of these communities is a significant issue in many problem domains, from marketing to healthcare due to the insight into the functionality of the network that can be gleaned from this endeavor. Over the last several decades this field has received a significant amount of attention due to how traditional algorithms fail to accurately identify communities in large-scale and complex networks within a reasonable time frame, due to the NP-complete nature of this problem, which has resulted in two main schools of approaches to solve this problem, heuristic and optimization.

Heuristic-based approaches generalize this problem and usually make assumptions about the structure of the network, such as in the Girvan Newman Algorithm where it is assumed that the nodes at the boundaries of communities will have high ”edge betweenness” values compared with that of those more internal to communities[12]. This makes them fast at solving the problem, however, this comes at the cost of some accuracy or domain-specific knowledge.

Optimization-based strategies aim to maximize an objective function, which near guarantees a high level of accuracy (provided a suitable objective function is used) - and recently this has gravitated towards the use of Evolutionary Computing, and more specifically, Genetic algorithms. However, due to the NP-hard nature of calculating this, these are often not scaleable within a reasonable time frame, and also subject to the shortcomings of their objective functions (such as the resolution limit seen in modularity [11]).

1.2 Approach

This paper investigates how we can keep the seen in optimization strategies for community detection, while simultaneously retaining the efficiency seen in the heuristic strategies.

To achieve this I have employed a Genetic Algorithm approach while utilizing a heuristic based on centrality in a network for a hybrid local search operator, where community centers gravitate towards more globally central nodes within a network. to the community detection problem could result in an accuracy higher than that seen in heuristic algorithms.

This method uses network modularity to measure the quality of partitioning communities within a network, optimizing this using GA methods. In doing this, the optimal number of partitions in the community can be automatically determined without needing prior specification. Communities are detected by using semi-local search operators to selectively explore the search space and identify nodes densely connected to their centers, without needing to know the exact number of partitions beforehand. This search method is optimal due to the biased nature of the population initialization which ensures all alleles in a chromosome are connected to at least one other allele in the same chromosome. The capabilities of this GA implementation is compared with that of other methods in terms of accuracy, determinism, and time complexity over synthetic and idealist social networks.

1.3 Aim and Objectives

This project aims to introduce a new GA-based approach hybrid local search to effectively and efficiently solve the problem of community detection. The objectives of this are to improve the efficiency of existing genetic approaches to this problem, while minimizing the loss, or even improving, in effectiveness from this gained efficiency.

1.4 Organisation

The remainder of this report is structured as follows:

1. **Chapter 2** introduces the background and related work regarding Community Detection, Genetic Algorithms, and Genetic Operators, as well as a brief overview of benchmark algorithms.

2. **Chapter 3** presents the design of the proposed algorithm, including a break down of the steps involved in this GA and top down view of the Genetic Operators

3. **Chapter 4** describes the more technical aspects of the implementation, in terms of the construction of the GA, the testing framework, and how it is run that is not directly connected to the GA itself (e.g. threading for multiple iterations)

4. **Chapter 5** presents the experimental evaluation of our proposed GA based algorithm. In particular, we will show a breakdown of benchmark datasets and algorithms used for evaluation, as well as the results output and subsequent discussion

5. **Chapter 6** presents possible extensions of this project in terms of future work, along with my conclusions on this project.

Chapter 2

Background and Related Work

2.1 Background

2.1.1 Problem Definition

A Social Network N can be defined as a graph

$$G = (V, E, W)$$

where V is a **set** of n objects (nodes or vertices), and E is a set of m links (edges) between two elements of V. A group of vertices containing a significant density of edges within them, accompanied by a lower density of edges between groups is characterized as a cluster or **community**. To deal with graphs, often the adjacency matrix, A, whose elements are denoted as A_{ij}, with the weighting of nodes i and j assigned by the function $W : V \times V \rightarrow R$ where the network contains N nodes. Accordingly, the entry at position (i, j) is 1 if there is an edge from node i to node j, otherwise, it is 0.

2.1.2 Measures of Community Structure

Community structure itself lacks a quantitative definition, instead relying on the qualitative one as briefly mentioned in the introductory chapter. This is derived from the sociological notion of a community which simply states that a community exists where the internal connections of a subgroup of objects (nodes) are denser than their external connections (edges) [50]. That is to say that a community exists where nodes are more densely connected within communities and loosely connected between communities. This has formed the basis of many attempts at a quantative definition of community structure, the important ones to this project are briefly described as follows.

Centrality

Beyond simply community detection, centrality detection is a whole additional field and could be viewed as a heuristic to be optimized in its own right for our algorithm. Instead of this, I will break down the four most commonly seen forms of measuring centrality in community detection:

Degree Centrality – This first and most simplistic is degree centrality. This is because it is the most intuitive and derives from the idea that the higher the number of connections (or edges) for which this node is the originator or destination, the more central a node is within the community [41]. This has a tendency, however, to overly generalize and therefore ignore communities that are less densely connected in favour of a more global maximum.

Closeness Centrality – Deviating from this we have closeness centrality, which indicates how close a node is to all other nodes in the network. It is calculated as the average of the shortest path length from the node to every other node in the network, where the lower the value, the more central a node is. This is typically used to find the most influental node in a subset of a graph [14].

Betweeness Centrality – Betweeness Centrality builds off of this measure, however calculates almost the inverse. Instead of calculating the sum of the shortest paths to every other node from this node, it calculates the number of shortest paths on which this node lies, i.e. An edge has high betweenness if it lies on a large number of short paths between vertices [28]. This is the method was proposed by M. Girvan and M. E. J. Newman and was utilized in their Girvan-Newman algorithm as detailed in [12]. The key problem of this approach, however, is when used in conjunction with a local search, it requires constant recalculation which comes at significant cost as the size of networks increases.

EigenVector Centrality - Finally, we have the more sophisticated view of centrality known as EigenVector Centrality, the basic idea of which is that nodes connected to other high scoring nodes will have higher centrality rankings themselves [43]. This has been used by google to rank search results in their PageRank algorithm [17] and by twitter to determine an individuals "ego network" [3]. The application of EigenVector Centrality to identifying community structure was proposed by M E J Newman in [36], which applied it to the modularity calculation. The interesting concept here is that a node can be central, even if not connected to a significant number of nodes directly, instead if the nodes it is connected to are very well connected the node will have a relatively high centrality value. The relative centrality score, x_v, of vertex v in the given network $G := (V, E)$ with $|V|$ vertices, with the **Adjacency matrix** $A = (a_{v,t})$ (i.e. $a_{v,t} = 1$ if vertex v is linked to vertex t, and $a_{v,t} = 0$ otherwise) can be defined by:

$$x_v = \frac{1}{\lambda} \sum_{t \in M(v)} x_t = \frac{1}{\lambda} \sum_{t \in V} a_{v,t} x_t$$

With $M(v)$ as the set of neighbors of v, and constant λ. Additionally, With a small rearrangement this can be rewritten in vector notation as the **eigenvector** equation

$$\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$$

In order to determine an absolute score that is comparable over a network, eigenvector values also need to be normalised such that the sum over all vertices is 1 or the total number of vertices n[1].

Modularity

Modularity is the measure of node density within a graph and is used here to measure the fitness of a partition of nodes within a network. Its applications to the community detection problem has been proposed in [36] where modularity measures the number of edges within the community against the number of edges going outside the community and gives a value between -1 and $+1$. Essentially it can be calculated by summing the number of edges identified within a community minus the number of edges expected by chance in the same community. By this metric, a Modularity score of +1 means that all the edges in a community are connecting nodes within that same community and-1 means that none of the edges in a community are connecting within that same community. The definition of Modularity for a weighted graph is as follows:

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$$

where:

- A_{ij} represents the edge weight between nodes i and j
- k_i and k_j are the sums of the weights of the edges attached to nodes i and j respectively
- m is the sum of all of the edge weights in the graph
- c_i and c_j are the communities of the nodes
- $\delta(c_i, c_j) = 1$ if $c_i == c_j$, otherwise 0

However, modularity presents two major problems for community detection: resolution limit and modularity maxima [11]. **Resolution Limit** is where the algorithm fails to detect sufficiently small communities within a network. This is dependent on the size of a network and as such one way suggested to overcome this limitation is to subdivide larger communities into smaller ones and then partition them. **Modularity Maxima** is a problem that arises from the hypothesis that the higher the modularity score, the better the quality of the community. However, as the number of nodes in a network increases so does the density of any randomly selected partition - as such it becomes increasingly hard to differentiate lower quality partitions from the best one.

The final problem of modularity for community detection is that, while in theory, optimizing this value would provide the best possible solution, in reality iterating through all possible permutations of node partitions in the network would be impractically computationally expensive, therefore heuristic algorithms are used, and as such it is an NP hard problem.

2.2 Related Work

2.2.1 Centrality based Community Detection

Centrality based Community Detection was very popular in the early 2000's after Girvan and Newman proposed their algorithm based on the sociological notion of betweenness centrality in [13], and was one of the first to deviate from more traditional heirarchical clustering methods. This method attempts to use centrality indices as the basis of the identification of community structure in order to sidestep the shortcoming of heirarchical clustering. They take a divisive approach to this problem, attempting to find the boundaries of communities by finding the nodes least central to a community, ie which ones are most between communities. I.e. Constructing communities by removing less central nodes instead of adding the strongest edges to an initially empty set. These Betweenness-based methods have also been generalized to use network components other than edges, such as the bipartite networks in [40] - which achieved a significant amount of success and is the basis of the greedy modularity algorithm used as a benchmark for this project. Alternative applications of centrality are shown in [11]. It is also worth noting that, while centrality based community detection is intuitively appealing, it can be too slow for many large networks, especially those that are very dense and have often been reported to give poor results when used as a fitness measure.

2.2.2 Heuristic Community Detection

There are two main methods of community detection, Agglomerative and Divisive. Since Girvan and Newman introduced the divisive approach to community detection discussed

above and in[12], this divisive method, especially in conjunction with betweeness centrality, has fallen out of favour as it is seen as far more computationally expensive than the agglomerative (adding nodes to a community over the algorithm runtime). There have been many approaches to Community detection based on hierarchical clustering, however the ones which I benchmark against are as follows:

Label Propagation

Label Propagation algorithms are bottom-up partitioning algorithms inspired by epidemic spreading[53]. In general, before the community detection is performed, a cleaning step is performed on the graph of nodes to remove singletons. After this, the population is initialized where a (usually unique) label is assigned to a small subset of the data points. These labels are then propagated throughout the course of the runtime of this algorithm to the unlabelled points and convergence is reached when each node has the same label as the majority label of its neighbours. By the end of this process most labels will have been eliminated, with only the most significant remaining, This process is described in [47] and summarised below:

1. Initialize the labels at all nodes in the network. For a given node x, $C_x(0) = x$.
2. Set t = 1.
3. Arrange the nodes in the network in a random order and set it to X.
4. For each $x \in X$ chosen in that specific order, let $C_x(t) = f(C_{x_{i1}}(t), ..., C_{x_{im}}(t), C_{x_{i(m+1)}}(t-1), ..., C_{x_{ik}}(t-1))$. Here returns the label occurring with the highest frequency among neighbours. Select a label at random if there are multiple highest frequency labels.
5. If every node has a label that the majority of their neighbours have (or t = the maximum number of user-defined iterations), then stop the algorithm. Else, set t = t + 1 and go to (3).

The key advantages of this approach[34] are in its low runtime, as well as the lack of prior knowledge regarding the structure of the network required, a pre-defined objective function, or prior knowledge about the communities.

Some examples of how this has been used can be found in [29], [52], and [42].

Greedy Modularity

Before discussing the Louvain method of community detection, it is worth discussing Greedy Modularity for community detection. The algorithm I use is the Clauset-Newman-Moore greedy modularity maximisation to find community structure by optimizing community density as outlined in [6] and runs in $O(n \cdot \log^2 n)$ time.

In essence, this method maximises modularity by initializing each node within its community and iteratively joins pairs of communities together until no further increase in modularity is possible. In this way, modularity is maximised at each step until an absolute maximum is reached.

The main Problem with this method is that it is computationally expensive if an ideal number of communities is not given as a parameter before-hand, however it is used as the basis for several more advanced methods, such as the Louvain method of community detection. Additionally, Given that it is simply an application of the idea of modularity to the problem of community detection, it inherits all of the problems of modularity, such as the resolution limit and the modularity maxima as detailed in [35] and summarised above.

Louvain Method

The Louvain method of community detection was proposed in 2008 by Blondel *et al* [4]. This method was inspired by the Greedy Modularity optimization method and as such was designed to optimise for modularity, however louvain also works with CPM as it's objective function. Running intime $O(n \cdot \log n)$, the Louvain approach to modularity is similar to the Clauset method of Greedy Modularity described above, where small communities are found first by optimizing modularity locally on all nodes, and then grouped to form larger communities repeatedly until modularity is maximised.

The Louvain method tackles this very differently however, and it's operation can be be broken down into two phases: Local Moving, and Aggregation. **Local Moving** The First deviation from the Greedy Modularity Optimisation method is that instead of combining communities which maximise the increase in modularity, individual nodes are moved between communities. This can be broken down into the following steps.

1. Assign each node to a different community
2. iterate through each node i, considering neighbours j, and evaluate the increase in modularity that would occur if we moved i into the community of j
3. add node i into the neighbouring community that gives maximal positive gain in modularity

Aggregation The second deviation is that a second phase called Community Aggregation is introduced. In this phase, communities identified in the first phase of modularity maximisation (demonstrated in Greedy Modularity) become the nodes in a new network and each new node connects with edges with weights equivalent to the sum of weights of all edges shared between those 2 communities/nodes. (This includes self-edges as well). The equation for this is as below: $\Delta Q = \left[\frac{\Sigma_{in}+2k_{i,in}}{2m} - \left(\frac{\Sigma_{tot}+k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$ where

- Σ_{in} is the sum of all the weights of the links inside the community i is moving into
- Σ_{tot} is the sum of all the weights of the links inside the community i is moving into
- k_i is the weighted degree of i
- $k_{i,in}$ is the sum of the weights of the links between i and other nodes in the community that i is moving into
- m is the sum of the weights of all links in the network

Once the aggregation phase is complete, we restart from the Local moving phase and repeat until no further modularity gain is possible and the optimal community is found.

This method outperforms the Clauset method, and many other methods of community detection - as will be shown when I evaluate my algorithm against it, in terms of both accuracy and speed. Additionally, the Lovain method can deal with both undirected and directed graphs.

The main downside of the Louvain method is its tendency to find arbitrarily badly connected communities that are internally disconnected[46]. It has also been found to be relatively computationally complex as the Leiden algorithm has a tendency to revisit the same nodes multiple times.

There have been a number of suggested improvements to address these issues in the Louvain algorithm such as Smart local moving (SLM)[49], Random moving[45], and Louvain Pruning[37], however the next major step forward for community detection was the Leiden Algorithm.

Leiden

The Leiden algorithm exists to fix the problem of internally disconnected communities in the Leiden algorithm, as well as improve the computational complexity of the algorithm itself. This was proposed to be achieved by first adding an additional phase to the Louvain algorithm in between the local moving and network aggregation phase, which is the refinement phase and then altering how the first phase operates, and build on the ideas introduced in SLM, with some speed ups inherited from Random moving and louvain pruning. **Refinement Phase** In this third phase, the Leiden algorithm attempts to identify refined partitions from the partitions proposed in the first phase - this is the first algorithm that does not does not follow a greedy approach instead opts to increase randomness by potentially merging communities which allows for a more broad exploration of the search space. [46]

Another avantage of the Leiden algorithm is that it scales much better than the Louvain algorithm for larger networks. This is because it also changes how the first phase operates, only visiting nodes which have changed neighbourhoods instead of iterating over every node again like in louvain[4].

The Leiden algorithm is currently heralded as the crowning achievement of the last decade of algorithmic improvements to the Louvain Algorithm. As such, it will be the main algorithm I benchmark mine against. This is because, not only does it solve the problem of internally disconnected communities but also scales much better over larger datasets.

Issues

A significant issue with hierarchical clustering on the whole, as has been discussed at length already, is the resulting "*dendrogram*" and the widely disparaged *Single Link Effect*, for how in large/complex datasets it is difficult to define appropriate splitting levels, calling into question the meaningfulness of the communities identified. Additionally the need to perform exchaustive search to optimize for modularity or CPM can result in significant loss in terms of execution time when dealing with this NP hard problem.

2.2.3 Evolutionary Computing and GA

In terms of EC approaches to this problem, many attempts have been made to this problem over the years such as those discussed in [38] and [30] however these often suffer from the same pitfalls in terms of needing information in advance - such as a k-cluster algorithm discussed in [10] which requires the k number of clusters to be known in advance.

Perhaps the most interesting and progressive approach taken to community detection has been that which views the field as an *Optimization Problem*, attacking it from an Evolutionary Computing perspective, or using Genetic Algorithms.

Evolutionary Computing, EC, is based on Darwin's 'survival of the fittest principle' in that it applies the idea of natural evolution to real-world search and optimization techniques. As they, in turn, have evolved, evolutionary methods have become incredibly flexible methods capable of solving any problem that can be defined as an optimization problem. All EC methods involve some method of population initialization followed by any number of genetic operations to, for example, improve the fitness of the function or navigate the search space in optimization.

In the last decade or so we have seen methods based on EC explode due to a few key factors that demonstrate its capacity to provide simple and accurate solutions in complex data sets. A large part of this is how, besides the parameters that are specific to GA, these algorithms often require no knowledge of the context of the data. Some examples of these are the number of communities and domain-specific knowledge. Another advantage GA has over its more meta-classical predecessors is that being population-based models, they are naturally parallel and efficient implementations that can be realized to deal with large-size networks.

GA could present an opportunity for improving the efficiency and efficacy of CD given GA complexity is dependent on the size of the population and the number of generations. This means that, given the NP-hard nature of the clustering problem, GA is a natural choice as a solution.

In 2007 GA was first applied to community detection, with effective performance [44] which aimed to optimise for Girvan-Newmans modularity, however this inherited it's problems (such as resolution limit) and additionally, calculating modularity scores for all potential solutions was found to be NP-hard. [38] aimed to fix these problems by introducing GA-Net along with a new measure of community structure they called "Community Score" which aimed to reduce the complexity of the objective optimisation. While this did somewhat achieve this, as the size and complexity of the network increases, the efficacy of this algorithm begins to drop in comparison with the original GA method [38].

Chapter 3

Central Communities Genetic Algorithm (CCGA)

To cope with this scalability problem, I have designed and implemented a Genetic Algorithm (GA)-based approach for social network analysis which relies on finding and identifying central nodes within a network to define communities, which I have named Community Centrality Genetic Algorithm (CCGA).

3.1 Solution Overview

GA has been proven to be a competitive optimization and search technique when compared with more traditional methods and has been applied to many problems across a diverse range of academic and practical domains such neural nets evolution, planning and scheduling, machine learning and pattern recognition [15]. My approach to this problem sought to achieve the following:

1. Reduce complexity of optimizing for modularity - Modularity has proven to be an NP hard problem that has proven to be difficult to obtain an optimal solution due to the exhaustive search method.

2. The second aim was to improve existing GA Approaches. To achieve this I have adapted my own encoding scheme and mutation operator based on the sociological principal that communities in social networks form around central individuals. This should reduce the number of iterations required by our algorithm.

3. Finally we sought to minimize the loss of efficacy, or even improve on it, even with this gained efficiency - These are both addressed by the addition of measuring centrality and only mutating towards more central nodes in the network as a form of Heuristic Mutation.

The algorithm follows a standard GA structure as shown in Algorithm 1.

Algorithm 1 GA structure

Require: $Adj : adjacencymatrix$
Require: $pop \leftarrow initialpopulation$
Require: $G : Graph$
Require: $pop_{size} \leftarrow 300$
Require: $gen_{max} \leftarrow 100$
  $centrality \leftarrow Eigenvectorcentralitymatrix of G$
  $convergence \leftarrow []$
  for ind in pop **do**
    $ind.subsets \leftarrow \text{SUBSETS}(ind)$
    $ind.fitness \leftarrow \text{EVALUATE}(ind.subsets)$
  end for
  $pop \leftarrow \text{SORT}(pop)$
  while $g \leq gen_{max}$ **do**
    $elites \leftarrow pop[: 0.1 \times pop_{size}]$
    loop
      parent1
      parent2
      $parent1; parent2 \leftarrow \text{CROSSOVER}(parent1, parent2)$
      $child1 \leftarrow \text{MUTATE}(parent1)$
      $child2 \leftarrow \text{MUTATE}(parent2)$
     invalidate fitness values
    end loop
      for ind in pop **do**
        $ind.subsets \leftarrow \text{SUBSETS}(ind)$
        $ind.fitness \leftarrow \text{EVALUATE}(ind.subsets)$
      end for
      $pop \leftarrow \text{TOURNAMENTSELECTION}(pop)[: 0.9 \times pop_size] + elites$
      $pop \leftarrow \text{SORT}(pop)$
      $convergence+ = best.fitness$
    end while
    $OUT \leftarrow Ind_{best}$
    $OUT \leftarrow convergence$

The deviation from more traditional GA methods is essentially in how the population is **initialised**, as well as how **mutation** and **crossover** is changed to follow on from this. A breakdown of the genetic representation and operators used for this implementation is discussed in the following subsections.

3.2 Genetic Operators

3.2.1 Selection

Selection is the process of selecting parents which mate and recombine to create offsprings for the next generation. The impact selection has on the GA convergence rate cannot be understated - good parents push the population towards more fit solutions, however if enough randomness is removed the GA is likely to get stuck in local maximas. For this reason i choose roulette wheel selection as it improves both the quality and diversity of the population. My selection process also uses the best 20% of the previous generation as elites to

improve convergence.

3.2.2 Crossover

For this algorithm, a method of Uniform Crossover was utilized, which uniformly exchanges central nodes at a rate of 60%.

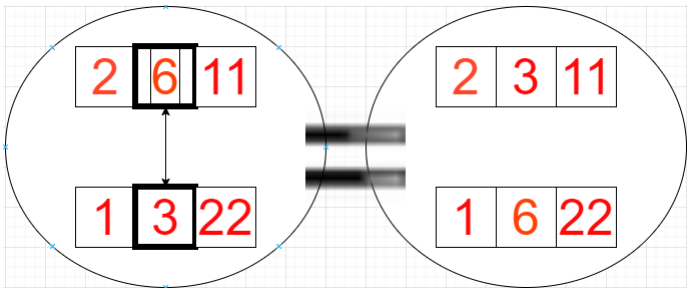


Figure 3.1: Illustration of centre-based uniform crossover

3.2.3 Mutation

Mutation is also very important as it provides minor tweaks to an individual to achieve closer to the desired result, and is typically implemented at a very low rate as, if too high, reduces GA to little more than random walk. My Mutation scheme differs from standard mutation schemes for locus based representation in the same way in which my representation itself differs. Similar to the crossover operator, I mutate the centers themselves. A heuristic measure is used here, borrowing from [35] which suggest that that eigenvector matrices can be used to abbreviate modularity. This allows us to use the eignvector values calculated during initialization to ensure each centre only mutates towards a more central node, thus ensuring that the subsequent community that forms around it is more densely connected (as eigenvector centrality is a measure of the influence a node exerts on a network). This is more practically achieved by finding the neighbouring node with the highest centrality score and mutating the current central node to that node if the central score is greater than that of the current node. This is helpful in reducing time complexity as we only need to search neighbouring nodes of a select few centres instead of the entire network. We also only make a change if the neighbouring node is closer to a local maxima, ie has a higher centrality score in the precomputed eigenvector centrality matrix, than the current centre, which I have implemented as a simplified local search measure in the interests of efficiency. For the reasons detailed here I have chosen a mutation rate of 40%, as, while still low, is higher than we would typically see because of the relatively low number of nodes to mutate (ie only those at the centre). This process is shown in Algorithm 2.

Algorithm 2 Mutation

Require: chrom
Require: $0 \leq MUPB \leq 1$
Require: $G : graph$
  for centre in *chrom.centres* **do**
    $0 \leq random \leq 1$
    if $random \leq MUPB$ **then**
      $neighbours \leftarrow$ G.GETNEIGHBOURS($centre$)
      if not neighbours **or** SIZE(neighbours) < 2 **then**
        n ▷ Random unvisited node
        $centre \leftarrow n$
      else
        n ▷ most central unvisited neighbour
        if $n.centrality \geq centre.centrality$ **then**
          $centre \leftarrow n$
          $visited \leftarrow visited + n$
        else
          n ▷ Random unvisited node
          $centre \leftarrow n$
        end if
      end if
    end if
  end for
  $OUT \leftarrow centres$

3.3 Encoding/Genetic Representation

There were two main encoding schemes considered for this project, the first being standard locus based representation, however this needed to be changed and so I designed my own encoding scheme based off of locus representation. It differs in how each node is not defined by the node it is related to but instead the centre of its community. My individuals are represented as a list of subsets associated with their centers. Each node is represented as an integer corresponding to its location on the network. as shown in Figure 3.2 below:

My method of community detection uses the locus-based adjacency representation as described in [39] and [38], and shown in Figure 3.1 below.

Figure 3.2: Illustration of centre-based representation

This matrix consists of n loci, each representing a node and an allele indicating which centre it is defined by. In Figure 3.2 we can see that node 3 corresponds to node 2 since the genotype for the node is 3. A disadvantage of this approach is that it requires an additional decoding step to rebuild the subsets after each change as the centers have changed and so the communities will have changed. However, similar to decoding locus based representation, this can be achieved in linear time as shown in [16]. Additionally, the number of k-clusters

can be automatically determined in a locus-based system, which is the main advantage of this representation for a GA over others, like medoid-based representation. The actual steps involved here will be discussed at length in the implementation section.

3.3.1 Decoding

The Decoding process is represented as shown in Figure 3.3. In step one all central nodes are divided into their subset communities. After this, step 2,connected nodes are added to these subset communities until either there are no nodes left not in a subset or no change is seen in subsequent iterations. At this point, step 3, remaining nodes placed in subsets along with one of their respective ungrouped neighbors. Step 4 shows communities with common nodes being merged together and this result is then output.

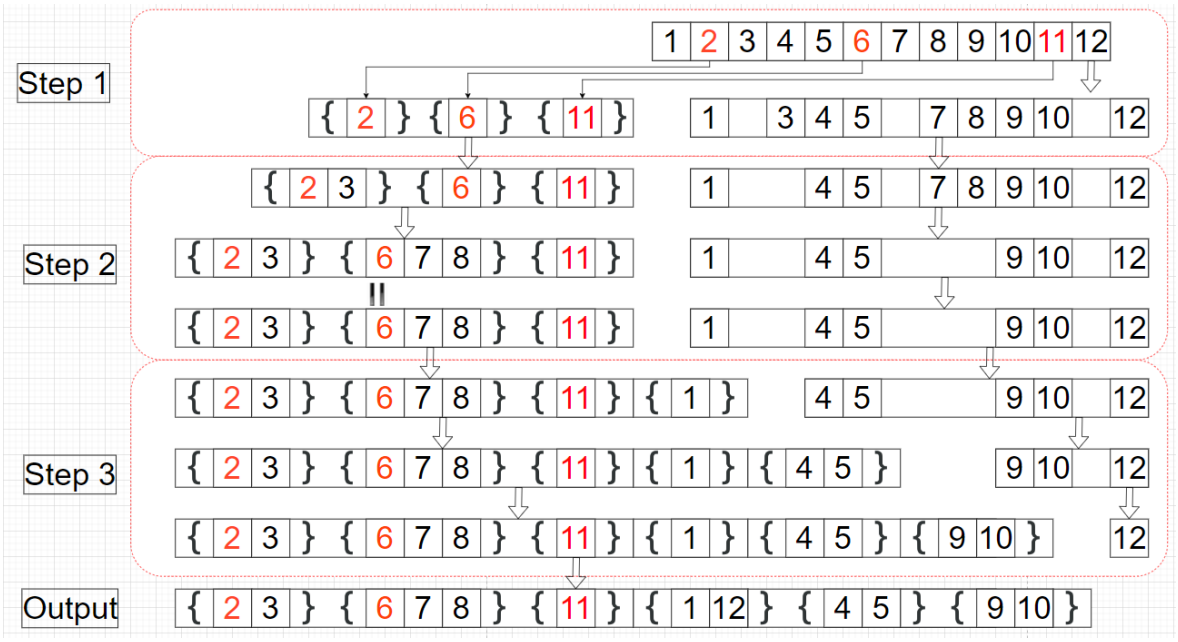


Figure 3.3: Illustration of centre-based subset decoding

3.4 Objective/Fitness Function

Using this representation, the graph will be partitioned into k communities, which will vary for each individual based on their number of centers. We will then optimise the our fitness function for each of these individuals.

For fitness function, a number were considered, from basic centrality measures to Common Potts, I settled on modularity as a measure of fitness as its priority of internal connectivity vs external connectivity fit with the notion of centrality very well. this is because a central node in a community would have a higher proportion of vertices within that community than without. As such, the fitness function for this algorithm will be modularity as described in the background section 2.1.2.

3.5 Initialization

For this algorithm each individual is initialized by first adding all nodes in the graph to a list and then selecting k nodes from this list at random to represent the central nodes. That

is to say, the individual is made up of all nodes in the network with a random sub-grouping of centers. While this leaves the possibility of generating multiple centers of the same value, this is fixed by removing duplicates, essentially turning this into a set of nodes.

Chapter 4

CCGA - Implementation

4.1 Language and Libraries

CCGA was implemented using the python language and relies heavily on the **DEAP** framework to store attributes and information regarding the GA algorithm itself. Other significant libraries include:

- DEAP[20] - for building and storing GA
- networkx[9] - for building and storing network information from data files.
- numpy[23] - for evaluating metrics (ie mean, std, etc)
- pandas[24] - for storing evaluation data
- matplotlib[22] - for plotting convergence curves
- cdlib[19] - for evaluating other algorithms
- leidenalg[4] - for evaluating leiden performance

4.1.1 Networkx for data preparation and storage

NetworkX[9] is a python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. I used it predominatly here for the storage of nodes, as well as the relationships between them and other attributes (such as their centrality within the graph). Initially I had planned to use Neo4j to store static relationships between nodes in the graph, however given the small scale of the datasets we are dealing with here (at most a few thousand nodes), the benefits of Neo4j would have been fairly negligible (storing them all within python memory was not too system intensive and we did not need the fully functional DB provided by Neo4j). Additionally because networkx is a python package we can utilize some of the natively python benefits such as fast prototyping, along with the additional representation options, a more intuitive API, and built in functionily for data type conversion and preparation. This made it relatively easy to convert from synthetic data in the early stages of development (as well as in its creation) to real data for evaluation with a variety of data types. I later consolidated all of these to the gml (Geography Markup Language) format - which is an XML like format that is defined by the Open Geospatial Consortium (OGC) [25]. This is structured as follows:

```
graph
[
```

```
  directed 0
  node
  [
    id 0
    label "Beak"
    club 4
  ]
  node
  [
    id 1
    label "Beescratch"
    club 3
  ]
  ...
  edge
  [
    source 8
    target 3
  ]
  edge
  [
    source 9
    target 5
  ]
}
```

where the club represents the label of the community this node belongs to, and edges are from source to target (although most of these networks are undirected so it is irrelvevant). some datasets also have weighted edges, which is the same as above except the edge variable contains another attribute 'weight'.

These were all loaded in looking only at the node ids (ie names were ignored) as other factors tended to vary significantly between datasets.

4.2 Build

Essentially, at each generation, the centres of each individual is mutated and crossed over which produces the offspring. Because of the nature of our representation this invalidated the fitness values as all affected communities need to be reconstructed. For this reason we invalidate the fitness values and then iterate through these values to rebuild the subsets and recalculate the fitness values. After this the new population is initialised by our tournament selection for 90% with the remaining 10% from the elites set aside earlier. This entirely replaces the existing population. The generation number was capped at 100 as my model managed to converge quickly enough that after 100 very little change was seen.

4.2.1 DEAP for GA design

DEAP[20] is an evolutionary computation framework for the rapid prototyping and testing of ideas, which is designed to make algorithms explicit and data structures transparent. I relied heavily on it for the the devolpment of my Genetic Algorithm as it simplified the process and made keeping track of operators, individuals, and outputs far easier. This is

predominantly through the Creator and toolbox modules which simplified the process of population initialisation significantly by making the individual representation both flexible and explicit, while also making the operator definitions far simpler. Also storing the population as an attribute of the toolbox, and registering all operations within the toolbox made persistence and fitness optimization much less manual:

```
def create(pop_size, G, Adj, CXPB, MUTPB ):
    # create the population
    creator.create("FitnessMax", base.Fitness, weights=(1.0,))
    creator.create("Individual", np.ndarray, fitness=creator.FitnessMax, subset=list, c

    # create the toolbox
    toolbox = base.Toolbox()

    toolbox.register("subsets", find_subsets)
    toolbox.register("evaluate", evaluate, weight='weight', resolution=1.0)

    toolbox.register("mate", cxUniform, indpb=CXPB)
    toolbox.register("mutate", mutation, mutation_rate=MUTPB)
    toolbox.register("select", tools.selRoulette, fit_attr="fitness",)

    toolbox.register("individual", generate_chrom, Adj=Adj, nodes=list(G.nodes))
    toolbox.register("population", tools.initRepeat, list, toolbox.individual, n=pop_si

    toolbox.register("centrality", calculate_centrality, G=G)
    toolbox.register("run", community_detection)

    return toolbox
```

This made implementing the GA algorithm itself far simpler.

4.2.2 Fitness

As mentioned, for my fitness function I used modularity. Fortunately there was a function in the **networkx** package that was able to calculate centrality for communities, so I did not to write this functionality myself. I also played with the idea of using centrality as a component of the fitness function itself, however that would require either multi-objective optimisation (which was beyond the scope of this project) or creating one heuristic from the two values. I attempted to implement this however the did not change at a consistant linear rate and so a consistant heuristic proved difficult. Additionally I did not see much improvement in the results from this as centrality is a component of modularity - especially when given the added execution cost of recalculating centrality values for all communities in a population.

The **DEAP** framework also alows you to state whether you are attempting to maximise or minimise your fitness value and will alter the output accordingly which made trying different fitness methods convenient and consistent. Here we are maximising modularity.

4.2.3 Initialisation and Representation

As mentioned I designed my own initialisation scheme, the process for generating each individual was as follows:

```
def generate_chrom(nodes, Adj):
    chrom = np.array(nodes, dtype=int)
    # shuffle the nodes
    np.random.shuffle(chrom)
    chrom = creator.Individual(chrom)
    chrom.centres = init_centres(chrom)
    chrom.visited = set()
    return chrom
```

As shown in the subsection on DEAP, The individual representation of the individual is a **NumPy** array of integers, with each integer representing the index of a node in the network. Each individual is initialised to contain all nodes in the network. The individual also contains a set of "**subset**"s as an attribute which is the break down of the subsets (or communities) identified for that individual. This is initialised as an empty set. The individual also includes a set of "centers" as an attribute which is a representation of the centers of each community. This is defined as follows:

```
  def init_centres(chrom):
      """
      Finds the node with the highest degree in the given chromsome
      :param chrom:
      :return:
      """
      # generate random number (k) of centres
      k = np.random.randint(1, len(chrom) / 3)
      # get the k nodes within the chrom as centres
      centres = set(np.random.choice(chrom, size=k, replace=False))
      # print("centres:", centres)
      return centres
```

where k random centrers are generated between 1 and 1/3 the size of the chromosome (the number of nodes in the graph). These paramaters were picked because a node can only be a centre if it is connected to at least 2 other nodes (otherwise it is just connected to another node). I also use a set to ensure there are no duplicates.

4.2.4 Mutation

My mutation operator is as depicted in the Design section. This operator went through a number of iterations and was born out of the idea of utilizing a heuristic for local search that would effectively add a mnuemetic hill climbing function to the program. Additionally I felt that mutating every node in the individual was expensive compared to output, where in a social network only a few key central nodes really affect the organisation of communities. This operator also went through a number of iterations both in terms of the centrality measure used as well as it's treatment of nodes that were not seen as central, or were duplicated.

Measures of Centrality

The first challenge presented here was to choose a measure of centrality. The obvious first choice was degree centrality as from a precursory glance at the modularity equation it seemed that the degree of a node was directly proportionate to its modularity value, and so gravitating towards nodes with a higher number of edges would result in a more central node and therefore climb towards a higher modularity value overall. This, however was

too simplistic. Betweeness centrality was also quickly ruled out as it was more capable of determining less central nodes, or boundary nodes, and therefore did not suit our needs.

The first measure implemented properly was to use **closeness centrality** and this was because it was seen as a good measure of the centrality of a node within the community - i.e. how many shortest paths within the community does this node lie on. The problem with this measure, however, came from how expensive this was to check for each node within the graph. While this did see an increase in accuracy of the algorithm, it came at significant expense in terms of execution time and would therefore lend itself far better to a full local search operator.

It was at this point I started looking at precomputing and storing these values before hand for the entire network during population initialisation, however it was too variable between communities and a consistent measure for closeness centrality over an entire network was not possible, at least not without performing it for every possible permuation of communities which was just as expensive as evaluating it for each community in the moment. This brought me to EigenVector centrality which I had earlier dismissed for this same reason, as it was relatively expensive to calculate. I had however read that it could be applied to modularity and thought it may be an effective heuristic to use for generalisation. Additionally, while the upfront cost of the evaluation was fairly high, it did not need to significantly vary between permutations and could be calculated for the entire network up front. For this reason, I choose this as my measure of centrality, utilizing Power Iterations for normalisation, to generate my heuristic. While Power Iterations is not the fastest method, I choose it due to it's simplicity to implement and effectiveness over very large networks given that it was only being calculated once over the entire network.

```
import numpy as np

def power_iteration(A, num_iterations: int):
    """
    Ideally choose a random vector
    To decrease the chance that our vector
    Is orthogonal to the eigenvector
    """

    b_k = np.random.rand(A.shape[1])

    for _ in range(num_iterations):
        # calculate the matrix-by-vector product Ab
        b_k1 = np.dot(A, b_k)

        # calculate the norm
        b_k1_norm = np.linalg.norm(b_k1)

        # re normalize the vector
        b_k = b_k1 / b_k1_norm

    return b_k
```

Weak Centres

"Weak Centres" were defined as nodes that had a low centrality value or lacked neighbouring nodes. Critical to this mutation is comparing each node to its most central neighbour. This meant that if a node was entirely disconnected from the rest of the graph, while it is a "center" to itself, it is not an effective center for building communities within the graph. For this reason, a solution to these disconnected nodes had to be designed. The initial solution was to combine this with another centre and treat it as one community which stemmed from my belief that as a matter of course, the number of centres should reduce throughout the course of the runtime, and also was functionally identical to how communities were found during the community building phase of the algorithm. However I found that this reduced randomness too much and that this should be accounted for simply by specifying a more limited range during initialisation. Additionally this made dealing with these centres far more complicated as they were no longer one node and were a set of multiple - which increased the cost of both finding neighbours and also identifying duplicates. On top of this it prove to be a fairly poor solution as these were often not found within the same communities and so would immediately worsen the quality of that individual.

The, perhaps more obvious and simple solution, was to simply replace it with a different node in the graph. The reason this works is that during the community building stage, nodes that are not connected to a centre are still divided into disconnected communities, and so entirely disconnected communities would naturally find themselves within their own communities and so are not needed to be defined as centres separately.

Duplicated Centres

Another problem that has been alluded to is that of duplicated centres, which is a problem that arises based on the randomness of selecting centres. While the set representation of the centres themselves should mean that any duplicated centres are removed automatically, this effectively results in no mutation occurring for a run in which a duplicate centre is chosen, and that for nodes that belong to the same community, if they eventually gravitate to the same central node, eventually one of these will be removed. Initially this was seen as ideal behaviour, however it reduced the randomness and so a solution was needed beyond simply relying on the functionality of sets. To cope with this I added a "visited" attribute to the individuals that would contain a list of all previous centres, which would be written to each time a new centre was found. Using this, only neighbours that had not yet been visited would be found for each centre for comparison. If this resulted in no neighbours being found, or if none were more central, a random node would be selected from the graph to replace it.

4.2.5 Crossover

The crossover operator was a fairly simple uniform crossover where the only design decisions that had to be made logically followed from the individual representation and mutation strategy.

4.2.6 Community Building and Decoding

As mentioned, the Decoding step is effectively a three step process and the reason for this is similar to the reason for my crossover operators design choices. Steps two and three are inspired by that seen in GA-Net which effectively operates the same way for nodes that are not immediatly obviously connected to a centre, and then these are slowly concatonated

together in the third step. The first step is the main point of difference as it builds these initial communities from the identified centres, iterating over until all connected nodes are contained within a community, before proceeding to group disconnected nodes as well as merging connected communities together.

4.3 Testing Framework

In order to perform testing the previously mentioned libraries were utilized for evaluation. Both cdlib and networkx contain many algorithms within themselves so all I had to implement was how each run would be evaluated. To do this I first recorded the convergence values and best individuals for each run, and then averaged these over all runs. These values were then used to output the resulting convergence curves, network graphs, and statistics tables and I repeated this for each algorithm.

Additionally, where labelled data existed, I used these labels to find the ideal solutions (ie community subsets) and then used this for calculating NMI, accuracy, and community difference. Where no such labels existed, I simply used the algorithm that achieved the best score when evaluated against my measure of fitness as a benchmark to compare my algorithm to - this was usually Leiden, although occasionally was greedy modularity.

4.3.1 CDlib and LeidenAlg for Benchmarking

CDlib[19] is a python library for the extraction, comparing, and evaluation of communoities in complex networks. Most significantly it provides a standardised input/output format for the benchmarking algorithms that I used in my evaluation. This allowed me to perform reliable and consistent evaluations across different algorithms in terms of accuracy over a dataset, execution time, NMI, and partition comparison. It also contained a number of synthetic benchmarking datasets that were useful when starting development. It did not include the leiden algorithm unfortunately and so I had to use a separate library **LeidenAlg**[48], which is the official code of the leiden algoritm published on github by its creator[21]. The structure of this library was slightly different from the others and so had to be evaluated separetely, but metrics were consistent.

4.3.2 Evaluation and Visualisation

For Evaluation I used **NumPy**[23] and **pandas**[24]. **NumPy** is possibly the most popular python package for data science or any kind of scientific computing in python - this is for a number of reasons, such as that it often is faster for looping than vanilla python, however here I used it predominantly for evaluating metrics for comparing benchmark algorithms, as it has built in functionality for calculating mean, standard deviation, and other useful values for results. **Pandas** is also an incredibly popular python library for data science, and specifically data analysis. This is because it offers a convenient data structure whithin which to store data and provides very powerful and convenient tools to manipulate it. I used it here to store evaluation data for later comparison across runs and between benchmarks, as well as eventual visualisation. **MatPlotLib**[22] is once again a very popular library for data visualisation - mostly due to its simplicity and symbiotic relationship with the NumPy package. I used it to show the communities output by the algorithms as well as the resultant convergence curves from CCGA and GA-Net.

```
def calc_metric(metric: list) -> dict:
    return {
```

```
        "mean": round(np.mean(metric), 4),
        "std": round(np.std(metric), 4),
        "min": round(np.min(metric), 4),
        "max": round(np.max(metric), 4),
        "median": round(np.median(metric), 4),
    }

def evaluate_fits(fits):
    # get all NMI scores
    metric_scores = {}
    print(fits)
    for metric in fits[0].keys():
        scores = [fit[metric] for fit in fits]
        metric_scores[metric] = calc_metric(scores)
        # metric_scores.append([fit[metric] for fit in fits])

    return pd.DataFrame(metric_scores, index=["mean", "std", "min", "max", "median"], colu
```

4.4 Run

4.4.1 Models

To improve running speed, all static components of this algorithm have been pickled and are loaded at the start of each session.

4.4.2 Processes

This model is also designed to run on multiple cores. This was to speed up efficiency when running multiple iterations (ie for testing) - as such each iteration is split into its own process and then queued for one of the available processors. As this is GA and utilizes the numpy library this only looks to use CPU cores not GPU cores as using GPU's rarely helps and is often a detriment to the process[2].

Chapter 5

Evaluation

5.1 Evaluation

5.1.1 Baseline datasets

Table 5.1: Datasets

Name	Nodes	Edges
Zachary Karate	34	78
Dolphin	62	159
Network Scientist	1589	3290

The First dataset I benchmarked my algorithm against was the **Zachary Karate Dataset** [51], the true communities of which are depicted in fig 5.6a.

This data set is a social network containing 34 members, or nodes, and 78 connections, or edges, within a university karate club that rose to popularity as a form of community structure after being used by Girvan and Newman in their 2002 paper "Community structure in social and biological networks" [12]. This dataset documents the interactions of its members outside of the club. During the study, a conflict arose between two high-ranking members of the club which lead the club to split in two, half going with each of the two leaders, and a third group giving up on karate altogether. Based on the data, Zachary was able to assign correctly all but one member of the club with each of the communities they joined. This Dataset has gained noteriety in the network science community and as such I will follow the tradition of using it as my first benchmark dataset.

I also used the Dolphin Dataset [27] which is a directed social network of bottle-nose dolphins. This network contains 61 nodes which represent the bottle-nose dolphins (genus Tursiops) of a bottle-nose dolphin community living off Doubtful Sound, New Zealand. An edge indicates a frequent association. The dolphins were observed between 1994 and 2001 and a "ground truth" for their community labels has been determined illustrated in 5.6b.

The final dataset I used (which contained a far more significant number of nodes) was the network scientist dataset which denotes a co-authorship network of 1589 data scientists working on network theory and experiments. This was compiled in 2006 by M. Newman from the bibliographies of two review articles on networks [2] and [33]. This was published in his paper "Finding community structure in networks using the eigenvectors of matrices (2006)" which also introduced the idea of using eigenvector centrality to measure community structure, which, given the use of eigenvector centrality in this algorithm, made the dataset a natural benchmark. A section of the graph containing the largest component of 379 scientists (and 914 edges) can be seen at [26].

5.1.2 Baseline Algorithms

I have benchmarked my Algorithm against the following:

- Label Propogation[32][7];
- Louvain[9][5];
- Greedy Modularity[31][6];
- Leiden[48][46]; and
- GA-net[18][38]

Initially, I was only going to benchmark against the algorithms implemented by Networkx Python library however, as you will also see, The have all been compiled into c and so run orders of magnitude faster than my own. Additionally, extracting convergence curves from these proved very difficult. Given this, I have also implemented GA-Net to provide a more direct level of comparison regarding execution times. This is based on the implementation provided in [38] however I have modified it slightly to be as close to my algorithm, without using the centrality values, as possible.

5.1.3 Parameters

For this set of evaluations, I have used a **mutation rate** of 0.6 and **crossover rate** of 0.4. This is due to the relatively low number of mutations my algorithm will be performing and so a higher chance of mutation is desired. Due to the variability of number of centres being so critical to the algorithm, the variety of individuals in the initial population is critical, meaning that the population size should be disproportionately larger than the number of generations. As such I tested a range of values with a **Population size** of 300 and **Generations** of no greater than 100 yielding the best results in terms of efficacy against efficiency.

I will also **Iterate** over each algorithm 30 times in order to reliably compare the performance of the different algorithms using the wilcoxon rank sum test and a p value of 0.005.

5.1.4 Results

For this evaluation, on each dataset, where a ground truth was known I calculated the NMI score as my principal measure of success. NMI is a measure of similarity proven to be reliable in [8] where the more similar the predicted communities are to the true communities, the higher the NMI Score. I have also compared Modularity (how densely connected the nodes are within the community vs without) and Centrality (how central the central nodes are to the community) values as a secondary measure for when the ground truth was not known, along with differences in community count (the difference in number of communities between the true and predicted values), and execution time.

The results of this for each algorithm are shown below in Table 5.3 and Table 5.2. I have also included the results on the Network Scientist dataset in Table 5.4, however as there is no ground truth value for this, the NMI score was calculated by similarity to the Leiden algorithm.

Table 5.2: Dolphins

ALGORITHM	NMI	COMMUNITY DIFF	MODULARITY	CENTRALITY	TIME
LPA	0.5756	2	0.4986	0.4108	0.0168 ± 0.0001
LOUVAIN	0.6334 ± 0.0569	1 ± 0.1795	0.5208 ± 0.0034	0.2887 ± 0.0277	0.0171 ± 0.0004
GREEDY	0.7227	0	0.4955	0.2829	0.0273 ± 0.0003
LEIDEN	0.7043	1	0.5233	0.2936	0.0005
CCGA	0.6087 ± 0.0385	1 ± 0.2494	0.4981 ± 0.0062	0.2452 ± 0.0424	1.4596 ± 0.0054
GA	0.606 ± 0.0382	4 ± 1.1397	0.4945 ± 0.0066	0.2056 ± 0.061	40.8561 ± 0.0045

Table 5.3: Zachary Karate

ALGORITHM	NMI	COMMUNITY DIFF	MODULARITY	CENTRALITY	TIME
LPA	0.6356	1	0.3095	0.4956	0.0092 ± 0.0001
LOUVAIN	0.6212 ± 0.0569	2 ± 0.3727	0.4409 ± 0.0059	0.3808 ± 0.0483	0.0098 ± 0.0003
GREEDY	0.8004	1	0.411	0.4047	0.0125 ± 0.0001
LEIDEN	0.467	2	0.4449	0.4022	0.0003
CCGA	0.6484 ± 0.1793	1 ± 0.6574	0.4019 ± 0.014	0.3324 ± 0.0735	0.6961 ± 0.7012
GA	0.5847 ± 0.0447	2 ± 0.4583	0.4408 ± 0.0034	0.3632 ± 0.0349	11.58 ± 0.0026

Table 5.4: Network Scientists

ALGORITHM	MODULARITY	CENTRALITY	TIME
LP	0.8762	0.3943	0.2762 ± 0.004
LOUVAIN	0.9548 ± 0.0001	0.3767 ± 0.0008	0.4271 ± 0.0165
GREEDY	0.943	0.3786	0.4899 ± 0.015
LEIDEN	0.9496 ± 0.0018	0.3767 ± 0.0005	0.0121 ± 0.0004
CCGA	0.9432 ± 0.0023	0.3798 ± 0.0018	15.89 ± 0.0042

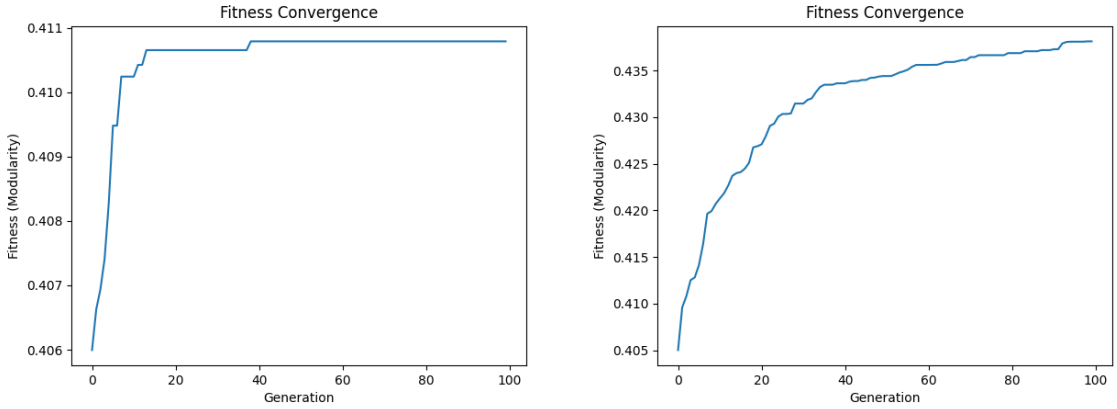
5.2 Discussion

Before venturing into our criteria for success, it is worth looking at the Modularity and Centrality columns as they are critical to the performance of the algorithm itself. In terms of Modularity and Centrality, both the Dolphin Dataset and the Karate Club dataset have optimal values we can compare against, being:

Modularity: 0.5190657015149717, 0.39143756676224206; and **Centrality**: 0.18797193879545337, 0.25529697659764145

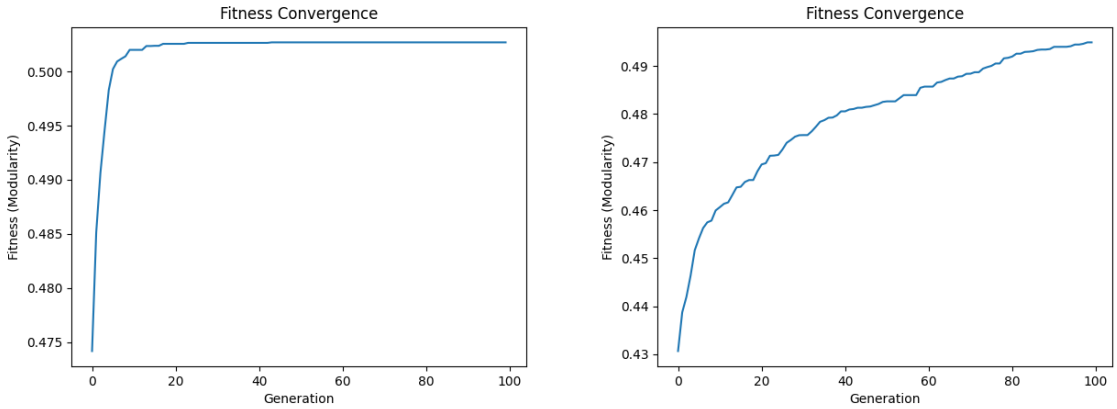
Respectively. For the Karate club dataset, our algorithm actually has the lowest absolute error for both benchmarks, ¡0.1 and 0.3 respectively. For the Dolphin dataset we also perform well, albeit not quit as close to either value, 0.2 off in both cases. Given that we are optimizing for modularity it is a good sign that we are achieving values so close to the optimal values. It is also worth noting that a general increase in average centrality values is associated with high levels of community.

What is immediately observable after this is the difference in execution times with and without the centrality mutation, where using it reduces execution time by almost a third in all. This is made more significant when looking at the convergence curves (below):



(a) Convergence With Centrality (b) Convergence Without Centrality

Figure 5.1: Convergence over Zachary Karate Dataset

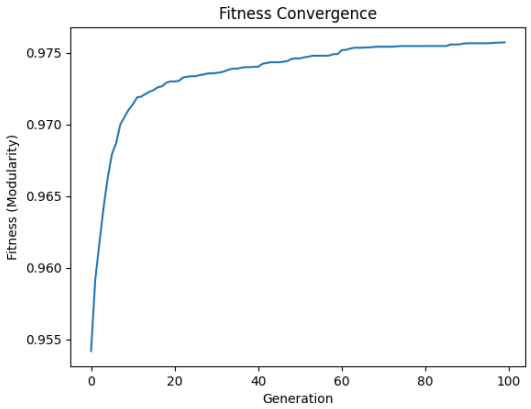


(a) Convergence With Centrality (b) Convergence Without Centrality

Figure 5.2: Convergence over Dolphin Dataset

Which shows how CCGA converges much faster on both datasets than a GA-Net algorithm. I speculate that this is due to the reduced number of potential combinations and the use of centrality as a sort of gradient which ensures that each centre slides“upwards” towards its local maxima quickly. This is also because of our high mutation rate, if we were to reduce it I would imagine the curve to look slightly flatter.

Fig 5.3 shows us the convergence curve of my centrality measure over the network scientist dataset which contains 1500 nodes and relates to the connections between published network scientists. This dataset was too large for the GA-Net approach, however we can see that it performs similarly well here, achieving 98% modularity score.



(a) Convergence With Centrality

Figure 5.3: Convergence over Network Scientists Dataset

While our algorithm performs signifcantly slower (1000 times slower in fact) than the benchmark algorithms I did not write myself, I put this largely down to these other algorithms being compiled in c, whereas ours is written in python and so will be much slower (10-100 times slower typically). I have attempted to somewhat combat this by pickling my algorithm, but this does little given that we are not actually training a model. Regardless, even accounting for this, it is an unavoidable truth that there is a magnitude of 10 times slower that is unaccounted for between my algorithm and the other other benchmark algorithms.

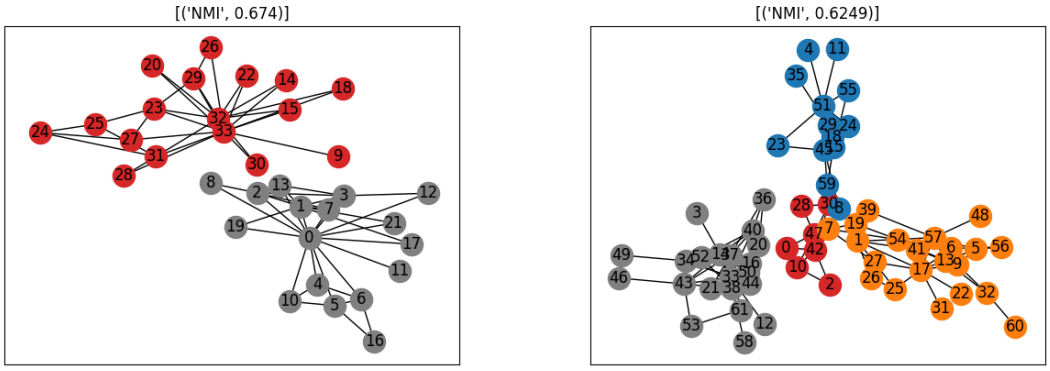
In terms of the evaluation scores themselves, on the Zachary Karate dataset my algorithm performed very well, with the second highest mean NMI score of any dataset (0.67). WThe high degree of spread here is worth noting (0.19) which is again the highest of any algorithm¿ what isnt shown here is that CCGA simultaneously managed to achieve 100% accuracy in the Zachary Karate dataset (twice over the 30 iterations), and also had the lowest minimum value (40%). When combining this with the observations from the convergence curve it is very curious, since we can see that the algorithm seems to be reaching a peak fitness value and then remaining there. This suggests that additional generations or a higher population size would not fix this problem. Additionally, looking at our modularity s.d. we do not see this same variability as is seen in our NMI score. In fact, looking back at iterations we see very similar values for modularity on the Zachary Karate Dataset (0.41) for both high (> 0.8) and low (< 0.5) NMI scores, which suggests this may not be the best measure of community, or may need to be combined with other methods.

Interestingly, this variability is not seen as strongly in the Dolphin dataset, still relatively high but smaller than that of the GA-Net algorithm. Equally, it neither performs so admirably or as poorly as the extremes seen in the Zachary Karate Dataset. In terms of mean

NMI for this dataset, my algorithm marginally outperforms GA-Net and significantly beats out the LP Algorithm - falling just short of Louvain by less than 1%. Both Newmans basic Greedy Modularity algorithm and the leiden algorithm perform significantly better overall.

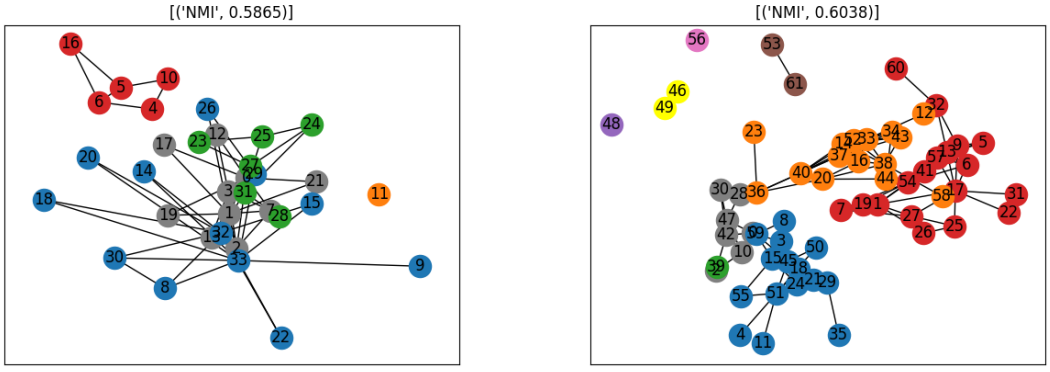
Also looking at the difference in predicted number of communities my algorithm performs incredibly well on the dolphins dataset, always guessing the correct number (4), but the same variability in terms of the Zachary Karate dataset is seen here. One possible reason for this could be due to my initialization strategy for the centres which, as described previously, generates a random number between 1 and n/2 (where n is the number of nodes) - and then through mutation reduces this number to find the optimal number. It is possible that for a dataset with this few communities (2), too many are being initially generated and the algorithm gets stuck in a local minima, unable to optimize fully.

The Community Structure identified by this algorithms (albeit only the final iteration of the 30) is shown in the figures below, along with their mean NMI scores for both CCGA 5.4 and 5.5, along with the true communities represented in 5.6:



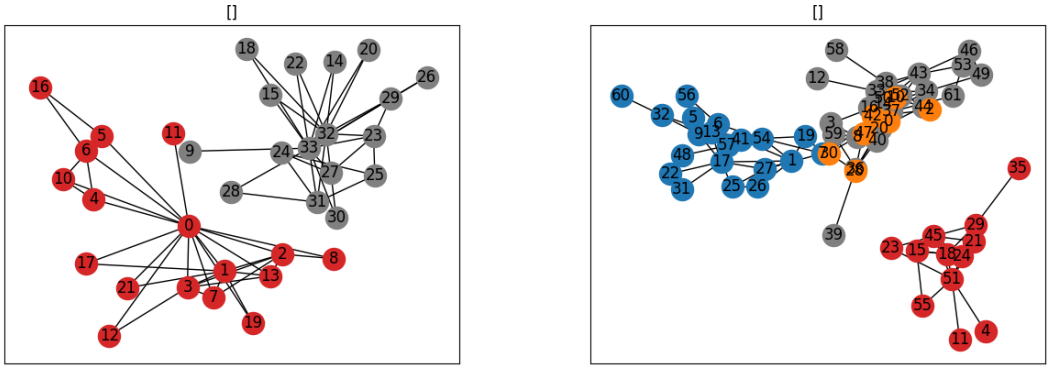
(a) Karate (b) Dolphins

Figure 5.4: CCGA



(a) Karate (b) Dolphins

Figure 5.5: GA-Net



(a) Karate (b) Bottle-nose Dolphin Dataset

Figure 5.6: Ground Truth

Overall our algorithm succeeds in reducing time complexity when compared with a similar implementation of GA in python, however fails to do so when compared with industry standards. Some of these differences can be explained away in terms of differences between languages, however the level to which this has been observed is too significant to ignore and so on this level the algorithm has failed.

In terms of our secondary objective - to improve accuracy, our algorithm has succeeded, being more proficient than many of these industry standards and GA-Net (for a third of the cost). However what we gain here we do lose in determinism unfortunately which is definitely a downside that warrants further investigation.

Chapter 6

Conclusions

6.1 Conclusions

For this project a novel GA approach to community detection in social networks has been designed, prototyped, and evaluated.

The overall aim of this project was to produce a community detection algorithm that is scalable for large datasets in terms of time and space complexity, with minimal loss of accuracy seen. This goal has been largely met through the project objectives which were;

1. not to see an exponential increase in execution time based on network size; and
2. improve the accuracy over other GA and heuristic methods of community detection

In order to meet these we

- Designed and implemented a GA for community detection that successfully identified communities in small social networks
- Designed and implemented new genetic operators by representing individuals by the centers of their communities, therefore only performing operations on these centers and reducing the cost of evaluation, while still evaluating for the entire community
- An evaluation of this improved GA (CCGA) on more significant datasets, with partial evaluation completed on real world datasets.

We have seen CCGA outperform standard GA on synthetic and real world datasets, and even some heuristic methods in terms of accuracy. That being said it is notable that the rate of convergence may be too rapid on smaller datasets, and also is not as fast as some of the more advanced heuristic methods.

6.2 Future Work

CCGA can be extended in a number of ways, the first would be to look at how using the leiden algorithm we could improve the execution time of this algorithm. This is because, as was noted in our experimentation, CCGA leaves much to be desired in terms of execution time, whereas leiden consistently executes the fastest across all data sets.

Another way CCGA could be extended is to determine a new fitness operator. Based on our evaluation, it can be determined that an alternative measure of fitness than modularity would be desirable and so further projects on this topic could investigate these alternative objective functions, such as utilizing a measure of centrality or the discussed Constant Potts

method, which avoids many of the pitfalls of modularity. Multi-Objective Optimisation using NSGAII to combine one of these measures with centrality could also be a worth wile endeavour.

Bibliography

[1] AUSTIN, D. How google finds your needle in the web's haystack, December 2006.

[2] BAETA, F., CORREIA, J., MARTINS, T., AND MACHADO, P. Speed benchmarking of genetic programming frameworks. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Jun 2021), p. 768–775. arXiv:2106.11919 [cs].

[3] BARASH, V., AND GOLDER, S. *Chapter 10 - Twitter: Conversation, Entertainment, and Information, All in One Network!* Morgan Kaufmann, Boston, 2011, p. 143–164.

[4] BLONDEL, V. D., GUILLAUME, J.-L., LAMBIOTTE, R., AND LEFEBVRE, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment 2008*, 10 (Oct 2008), P10008.

[5] BLONDEL, V. D., GUILLAUME, J.-L., LAMBIOTTE, R., AND LEFEBVRE, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment 2008*, 10 (oct 2008), P10008.

[6] CLAUSET, A., NEWMAN, M. E. J., AND MOORE, C. Finding community structure in very large networks. *Physical Review E 70*, 6 (dec 2004).

[7] CORDASCO, G., AND GARGANO, L. Community detection via semi-synchronous label propagation algorithms. pp. 1 – 8.

[8] DANON, L., DÍ AZ-GUILERA, A., DUCH, J., AND ARENAS, A. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment 2005*, 09 (sep 2005), P09008–P09008.

[9] DEVELOPERS, N. $\text{Louvain}_{c}ommunities, Jun2022$.

[10] FIRAT, A., CHATTERJEE, S., AND YILMAZ, M. Genetic clustering of social networks using random walks. *Computational Statistics Data Analysis 51*, 12 (Aug 2007), 6285–6294.

[11] FORTUNATO, S. Community detection in graphs. *Physics Reports 486*, 3-5 (feb 2010), 75–174.

[12] GIRVAN, M., AND NEWMAN, M. E. J. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America 99*, 12 (Jun 2002), 7821–7826.

[13] GIRVAN, M., AND NEWMAN, M. E. J. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. U. S. A. 99*, 12 (June 2002), 7821–7826.

[14] GOLBECK, J. *Chapter 3 - Network Structure and Measures*. Morgan Kaufmann, Boston, 2013, p. 25–44.

[15] GOLDBERG, D. E. *Genetic algorithms*. pearson education India, 2013.

[16] HANDL, J., AND KNOWLES, J. An evolutionary approach to multiobjective clustering. *IEEE Transactions on Evolutionary Computation 11*, 1 (2007), 56–76.

[17] HANSEN, D. L., SHNEIDERMAN, B., SMITH, M. A., AND HIMELBOIM, I. *Chapter 3 - Social network analysis: Measuring, mapping, and modeling collections of connections*, second edition ed. Morgan Kaufmann, 2020, p. 31–51.

[18] HARISWB. hariswb/ga-community-detection, Jul 2022.

[19] HTTPS://CDLIB.READTHEDOCS.IO/EN/LATEST/. cdlib.

[20] HTTPS://DEAP.READTHEDOCS.IO/EN/MASTER/. Deap documentation — deap 1.3.3 documentation.

[21] HTTPS://GITHUB.COM/VTRAAG. Vincent traag.

[22] HTTPS://MATPLOTLIB.ORG/. matplotlib.

[23] HTTPS://NUMPY.ORG/. numpy.

[24] HTTPS://PANDAS.PYDATA.ORG/DOCS/REFERENCE/API/PANDAS.DATAFRAME.HTML. pandas.

[25] HTTPS://WWW.OGC.ORG/. Ogc.

[26] HTTP://WWW-PERSONAL.UMICH.EDU/ MEJN/CENTRALITY/. Community centrality.

[27] LUSSEAU, D., SCHNEIDER, K., BOISSEAU, O. J., HAASE, P., SLOOTEN, E., AND DAWSON, S. M. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology 54*, 4 (Sep 2003), 396–405.

[28] MASON A. PORTER, J.-P. O., AND MUCHA, P. J. Communities in networks.

[29] MURASE, Y., YOSHINO, K., MIZUKAMI, M., AND NAKAMURA, S. Feature inference based on label propagation on wikidata graph for dst. 12.

[30] M'BAREK, M. B., BORGI, A., BEDHIAFI, W., AND HMIDA, S. B. Genetic Algorithm for Community Detection in Biological Networks. *Procedia Computer Science 126* (Jan. 2018), 195–204.

[31] NETWORKX. greedy$_m$*odularity*$_c$*ommunities*$_n$*etworkx*.

[32] NETWORKX. label$_p$*ropagation*$_c$*ommunities*.

[33] NEWMAN, M. E. J. The structure and function of complex networks. *SIAM Review 45*, 2 (2003), 167–256.

[34] NEWMAN, M. E. J. Detecting community structure in networks. *The European Physical Journal B - Condensed Matter 38*, 2 (Mar 2004), 321–330.

[35] NEWMAN, M. E. J. Fast algorithm for detecting community structure in networks. *Physical Review E 69*, 6 (jun 2004).

[36] NEWMAN, M. E. J. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences of the United States of America 103*, 23 (Jun 2006), 8577–8582.

[37] OZAKI, N., TEZUKA, H., AND INABA, M. A simple acceleration method for the louvain algorithm. *International Journal of Computer and Electrical Engineering 8* (2016), 207–218.

[38] PIZZUTI, C. *GA-Net: A Genetic Algorithm for Community Detection in Social Networks*, vol. 5199 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008, p. 1081–1090.

[39] PIZZUTI, C. Evolutionary Computation for Community Detection in Networks: A Review. *IEEE Transactions on Evolutionary Computation 22*, 3 (June 2018), 464–483.

[40] PORTER, M. A., FRIEND, A. J., MUCHA, P. J., AND NEWMAN, M. E. J. Community structure in the u.s. house of representatives. *Chaos: An Interdisciplinary Journal of Nonlinear Science 16*, 4 (dec 2006), 041106.

[41] POWELL, J., AND HOPKINS, M. *19 - Graph analytics techniques*. Chandos Information Professional Series. Chandos Publishing, Jan 2015, p. 167–174.

[42] SPERIOSU, M., SUDAN, N., UPADHYAY, S., AND BALDRIDGE, J. Twitter polarity classification with label propagation over lexical links and the follower graph. In *Proceedings of the First Workshop on Unsupervised Learning in NLP* (USA, Jul 2011), EMNLP '11, Association for Computational Linguistics, p. 53–63.

[43] STREICHER, S. J., WILKEN, S. E., AND SANDROCK, C. *Eigenvector Analysis for the Ranking of Control Loop Importance*, vol. 33 of *Computer Aided Chemical Engineering*. Elsevier, 2014, p. 835–840.

[44] TASGIN, M., HERDAGDELEN, A., AND BINGOL, H. Community detection in complex networks using genetic algorithms, 2007.

[45] TRAAG, V. A., ALDECOA, R., AND DELVENNE, J.-C. Detecting communities using asymptotical surprise. *Physical Review E 92*, 2 (aug 2015).

[46] TRAAG, V. A., WALTMAN, L., AND VAN ECK, N. J. From louvain to leiden: guaranteeing well-connected communities. *Scientific Reports 9*, 1 (mar 2019).

[47] USHA NANDINI RAGHAVAN, R. A., AND KUMARA, S. Near linear time algorithm to detect community structures in large-scale networks.

[48] VTRAAG. leidenalg.

[49] WALTMAN, L., AND VAN ECK, N. J. A smart local moving algorithm for large-scale modularity-based community detection. *The European Physical Journal B 86*, 11 (Nov 2013), 471.

[50] WASSERMAN, S., AND FAUST, K. *Social Network Analysis: Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, 1994.

[51] ZACHARY, W. W. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research 33*, 4 (1977), 452–473.

[52] ZHANG, P., WANG, F., HU, J., AND SORRENTINO, R. Label propagation prediction of drug-drug interactions based on clinical side effects. *Scientific Reports 5*, 11 (Jul 2015), 12339.

[53] ZHU, X., AND GHAHRAMANI, Z. Learning from labeled and unlabeled data with label propagation. Tech. rep., 2002.