Multi-objective Evolutionary Algorithm Framework for Community Detection

Research Report

Brandon Zahn 3186200

Prof. Pablo Moscato, A/Prof. Regina Berretta, Dr. Nasimul Noman
Computer Science Research

March, 2018

Contents

1 Introduction

This research project was undertaken through the Centre for Bioinformatics, Biomarker Discovery and Information-Based Medicine (CIBM) working at the Hunter Medical Research Institute[1] in NSW, Australia. The centre brings together researchers from both Health and Engineering faculties using high advanced computational and mathematical methods to develop patient tailored treatment for a number of genetic diseases.

Brief descriptions of community detection in networks, genetic algorithms and the research objective proceed from this point. An overview of the Multi-Objective Evolutionary Algorithm (MOEA) framework will be provided in Section 1.4 and the objective functions used are presented in Section 1.5. However, more details on the research and development of the framework can be found in the report cited here [5]. In Section 2, the CiteSeer and Cora data sets are introduced together, followed by Section 3 with a description of the tripadvisor data set and the methods used for correlation. Section 4 presents an analysis and discussion of the results. Modifications that were made to the framework are described in Section 5 and finally Section 6 brings a conclusion along with future work suggestions for the advancement of the framework.

1.1 Community detection in networks

Community detection in networks, or graphs, consist of vertices and edges. An edge typically connects a pair of vertices which indicates some sort of relation with one another, depending on what the network vertices represent.[3] Most real-world networks display some sort of community structure, meaning that their vertices are organised into groups, called communities or clusters. Identifying communities may offer insight on how the network is organised. It allows for focus on regions having some degree of autonomy within the graph. It helps to classify the vertices, based on their role with respect to the communities they belong to. Community detection in networks is still in the stage of major ongoing development so there is no universal definition of the things that researchers should be looking for. Consequently, there are no solid guidelines on how to assess the performance of different algorithms in this field.

Community detection can be done in several ways using different approaches. Link based algorithms that analyse the structure of edges to cluster vertices are common. Content based analysis also exists, where additional information about the vertex is used to help accurately determine communities. Ideally, when both link and content based approaches are applied simultaneously, the best features of both approaches can be combined to improve the performance of community detection. This is a fairly new approach and often uses a weighted objective in order to maintain the use of a single-objective optimisation framework. Since a truly multi-objective framework has not yet matured, it is the focus of this research.

[1] https://hmri.org.au/

1.2 Genetic algorithms

Genetic algorithms are a type of evolutionary algorithm that are often applied as a heuristic in optimisation problems.[12] There are many different types but all have the same fundamental directives which consist of the following steps:

1. Initialise a random population of solutions
2. Select solutions from the current generation to be parents for the next generation
3. Breed new solutions using some crossover and mutation methods from old solutions
4. Select a new generation from the old and new solutions
5. Repeat steps 2-4 until a terminating condition is met

When new individuals are bred from previous generations, the genes are passed on from old to new individuals. So a child population would inherit genes from one parent and some from the other when a generation passes. These genes are the properties of the solutions, which in the case of community detection, would usually be some kind of community representation. When choosing which solutions form the next generation, the process can vary greatly depending on whether there is a single-objective or multi-objective approach being applied. In the case of single-objective optimisation, solutions can be simply ranked by their objective function value. Multi-objective optimisation, however, uses the concept of domination. Generally, if there are too many solutions of a given rank in both cases crowding distance is used, which ensures that solutions are spread across the solution space and the program does not get stuck in a local optima. At the end of the algorithm in the single-objective case, the best solution which has been seen so far is returned. In the case of multi-objective optimisation this is not possible except in the case that a single solution dominates every other solution found. The set of non-dominated solutions found is called the pareto Front. Optimally, the non-dominated solutions will be spread well across the multi-dimensional euclidean space of the objective function values.

1.3 Research objective

Validating the use of multi-objective optimisation for community detection in large data sets is an important step in reinforcing the strengths of using multiple objectives over traditional single-objective optimisation. Consequently, the objective of this research was to validate the ability for a proposed MOEA framework to perform community detection using two objective functions. The framework has only been exposed to limited testing with small data sets, such as the karate club data set.[2] Thus, it was necessary to subject it to tests with large, real-world networks.

[2]http://www-personal.umich.edu/~mejn/netdata/

Three data sets were selected to test the framework for this research project, the CiteSeer and Cora data sets[3], and the tripadvisor data set[4]. The CiteSeer and Cora data sets are document classification data sets that describe a directed network of scientific publication citations. All publications are classified into one of several classes. The tripadvisor data set contains reviews of various hotels that were trawled from the tripadvisor website.

1.4 MOEA Framework Overview

The MOEA framework is based on a genetic algorithm called *"Non-dominated Sorting Genetic Algorithm II"* (NSGA-II).[1] Algorithm 1 shows how the framework has been arranged. It starts by initialising a random population of solutions, referred to as individuals. It then evaluates and sorts the individuals. For each generation the framework is run, a child population is created using a crossover function. The child individuals are mutated to introduce diversity into the solution space, before merging the two populations together to give a mixed population. The mixed population is sorted into fronts using a fast non dominating sort (FastNDS). The individuals in each front are all dominated by the same number of other individuals. In each front, the crowding distance of the objective function values is also assigned. Individuals from the mixed population are then selected to form the parent population for the next generation.

Describing the edges in the network is done by using an adjacency matrix Adj and a similarity matrix Sim for the edge-weightings. For a graph G with vertex set V, Adj is a square $|V| \times |V|$ matrix A such that its elements A_{ij} is one when there is an edge from vertex i to vertex j, and zero when there is no edge. The diagonal elements of the matrix are all zero, since edges from a vertex to itself are ignored.

Figure 1.1 shows an example of an unweighted graph G_{ex} described by matrix Adj_{ex} in equation 1.1. G_{ex} represents an undirected graph, where there are ones in $Adj_{ex_{ij}}$ and $Adj_{ex_{ji}}$, representing the same edge. In the case of a directed graph, the direction of the edge is indicated by element Adj_{ij} meaning that the edge direction is from $V(i)$ to $V(j)$.

$$Adj_{ex} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{1.1}$$

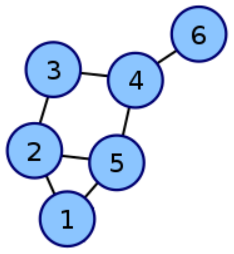


Figure 1.1: Labelled graph G_{ex} with $|V| = 6$. The numbers in each vertex represent the vertex number and the row in the matrix Adj_{ex}

[3]https://linqs.soe.ucsc.edu/data
[4]http://times.cs.uiuc.edu/~wang296/Data/

Algorithm 1: The MOEA framework

```
Input: $N_{gens}$, $N_{popsize}$, $Adj$, $Sim$
Output: Pareto front of non-dominated solutions
Instantiate $Population\ Parent(N_{popsize})$
Instantiate $Population\ Child(N_{popsize})$
Instantiate $Population\ Mixed(2N_{popsize})$
$Initialise(Parent) \longleftarrow$ Generates a random set of solutions
$Evaluate(Parent)$
$FastNDS(Parent)$
$CrowdingDistance(Parent)$
while $t < N_{gens} \longleftarrow t_{init} = 0$ do
    $Child = Selection(Parent)$
    $Mutate(Child)$
    $Evaluate(Child)$
    $Crossover(Child, Parent) \longleftarrow$ Combines the two populations
    $Fronts = FastNDS(Mixed)$
    $CrowdingDistance(Fronts)$
    $Clear(Parent) \longleftarrow$ Remove the past parent population
    foreach $f \longleftarrow Fronts$ do
        if $Parent.size + f.size < N_{maxpopsize}$ then
            $Parent.add(f)$
        else
            Remember $f_{next}$
            break
    while $Parent.size < N_{maxpopsize}$ do
        $Parent.add(f_{next}.next)$
    output $Parent$
    Increment $t$
```

The *FastNDS* algorithm this is a sorting method that ranks individuals in a population into fronts. The algorithm is a two-dimensional loop that checks the dominance of two individuals. Each individual maintains a count of the number of individuals that dominate it and a list of the individuals that it dominates. The *Selection* algorithm chooses which individuals in the parent population will participate in crossover to produce a child population by randomly selecting two individuals from the parent population and putting them through a tournament, where the winner is assigned as the fist parent and the second as the other parent. The *Mutation* algorithm randomly alters the child population by altering the community representations of specific vertices. There is a substantial random factor in this decision. The *Crossover* algorithm is one of the key components of this genetic algorithm. It is where the genes of the parents are mixed together to create the genes of the child.

1.5 Objective functions

There are two objective functions; modularity and weighted modularity. Modularity is a community detection metric commonly used to evaluate the quality of clusters. It analyses the structure of a detected partition's edges and compares it with the result of a random graph. From this the modularity of a partition is defined. Finding a partition structure that produces a higher modularity value implies that a better partition is obtained, as compared to a random graph.[6] The modularity objective function is maximisation. In a graph G with an adjacency matrix Adj_{ij}, the modularity Q of a partition P is calculated as

$$Q(P) = \sum_{c=1}^{C} \left(\frac{l_c}{m} - \frac{d_c}{2m} \right) \tag{1.2}$$

where C is the number of communities in the partition P, l_c shows the number of edges within community c, d_c is the total degree of vertices within community c and m shows the number of edges in G. The weighted modularity objective function is also a maximisation function and is calculated by the edge weightings. It also takes into account the internal and total degree of vertices. It is defined as

$$Q_{weighted}(P) = \sum_{c=1}^{C} \left(\frac{L_c}{M} - \frac{D_c}{2M} \right) \tag{1.3}$$

where

$$M = \sum_{i,j \in E} Sim_{ij}$$

with Sim_{ij} coming from the similarity matrix,

$$L_c = \sum_{i,j \in c} Sim_{ij}$$

for the internal degree of community c and

$$D_c = \sum_{i \in c} \sum_{\forall j} Sim_{ij}$$

for the total degree. The way in which the framework has been implemented has led to the inverting of both the objective functions during execution time. This means that while both objective functions are intended to be maximisation functions, it runs with minimisation functions, so the results obtained will show negative values for both Q and $Q_{weighted}$. Simply negating the results will give the maximised fitness scores.

2 The CiteSeer and Cora Data Sets

The Cora dataset consists of 2708 scientific publications, each classified into one of seven classes. This network consists of 5429 directed links describing the citations made by each publication. The contents of each publication in the data set is described by a zero or one-valued word vector indicating the absence or presence of the corresponding word from a dictionary which consists of 1433 unique words, created by finding regularly used and significant words in the publication.

The CiteSeer data set consists of 3312 scientific publications classified into one of six classes. This network consists of 4732 links. The citations and the content of each publication in this data set is described in the same way as the Cora data set. The dictionary consists of 3703 unique words. Information about these two networks is presented in Table 2.1. Figure 2.1 shows how these two data sets are structured, each publication is referred to in a cites file and a content file, where the cites file stores the links and the content file stores the vocabulary array.

Since both of the data sets describe links and content, interpreting the data to be used in the MOEA framework is not too complicated. The citations are used to define directed edges in the network and the number of common vocabulary terms as an edge weight. This is how these data sets are most commonly interpreted when performing community detection.

Table 2.1: Network details where |V| and |E| refer to the number of vertices and edges in the graph, respectively

Network	**|V|**	**|E|**	**Vocabulary size**	**Class count**
CiteSeer	3312	4732	3703	6
Cora	2708	5429	1433	7

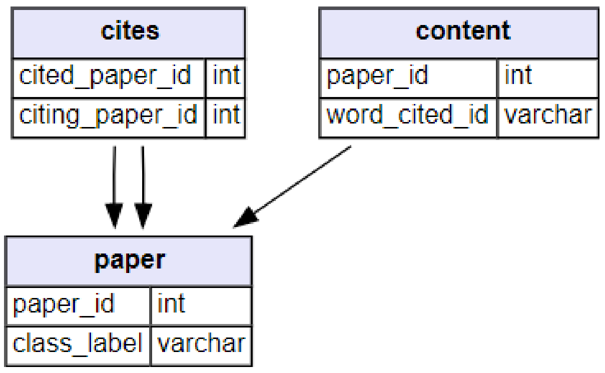


Figure 2.1: The structure of the CiteSeer and Cora data sets

2.1 Making correlations between publications

A correlation program has been made to create the input file for the MOEA framework. The list below describes the steps in the correlation process:

1. Define structures for Cora papers and CiteSeer papers. These structures have an ID, vocabulary array, paper class, cites array and a cited by array as member variables.

2. Process the Cora cites and content files

3. Calculate and write Cora Adj matrix by setting element Adj_{ij} where $publication_i$ cites $publication_j$

4. Calculate and write Cora Sim matrix by setting element Sim_{ij} to the number of common vocabulary between $publication_i$ and $publication_j$

5. Process the CiteSeer cites and content files

6. Calculate and write CiteSeer Adj and Sim matrices in the same way as in steps 3 - 4

2.2 Using ground truth to evaluate performance

All of the publications from these data sets are in predefined classes which can be used as a ground truth to evaluate the performance of the MOEA framework by comparing the detected communities to the original classes. A metric known as the Normal Mutual Information (NMI) is used to measure the partition performance. [11] $C = \{C_1, \ldots, C_k\}$, where C_k contains the set of vertices that are in the k^{th} community is used to represent the community structure of the predefined classes. The community structure given by the MOEA framework is represented by $C' = \{C'_1, \ldots, C'_k\}$. The mutual information (MI) between the two is defined as

$$MI(C, C') = \sum_{C_i, C'_j} p(C_i, C'_j) log \frac{p(C_i, C'_j)}{p(C_i)p(C'_j)} \tag{2.1}$$

and the NMI is defined by

$$NMI(C, C') = \frac{MI(C, C')}{max(H(C), H(C'))} \tag{2.2}$$

where $H(C)$ and $H(C')$ are the entropies of the partitions C and C', $p(C, C')$ is the joint probability function of C and C', and $p(C)$ and $p(C')$ are the marginal probability distribution functions of C and C' respectively. The higher the NMI is, the closer the partition is to the ground truth.

3 The tripadvisor Data Set

The tripadvisor data set contains some 37,000 hotel reviews about 1850 hotels from around the world that were crawled from the tripadvisor website. Each review is comprised of certain words, or aspects, that could be used in making correlations between hotels.[10][9] The data set consists of the following files:

1. *1850 hotel review files*; one file for each hotel that contains all of that hotel's reviews. Every review is comprised of an author name, a content field that shows the review, a review date, eight ratings ranging from one to five and eight lists of aspect words, one for each rateable category. The eight rating categories are overall, value, rooms, location, cleanliness, check-in and front desk, service and business service. An aspect is made up of an ID number, the word itself and an occurrence count, representing the number of times it has been found in the review. Figure 3.1 shows a typical review and Figure 3.2 shows a review with incompatible ratings, which are represented by -1. These reviews were taken from hotel with ID 72572.

2. *Featured words list*; seven categories of featured words that list words to be considered more significant for each rating category. Figure 3.3 shows an excerpt of the featured words file, the first 10 words from each category. The Overall aspect category does not contain featured words.

3. *Locations and prices list*; prices and locations for all hotels. Figure 3.4 shows an excerpt of the price and locations for 10 hotels. Note that the bottom two hotels have an Unknown value in the price and location fields.

```
<Author>everywhereman2
<Content>Old seattle getaway This was Old World Excellence at it’s best.THIS is the place to stay at when visiting the historical
     area of Seattle. Your right on the water front near the ferry’s and great sea food restraunts,and still with’in walking
     distance for great blues and jazz music. The staff for this hotel are excellent,they make you feel right at home. The
     breakfast was great.We did’nt have to travel far to have a good cup of JOE and a light meal to start our adventurous day off
      into one of the most beautifull city’s in america. This hotel is in an area that makes it easy to get to any place you want
      to go and still find your way back, I highly recomend this hotel for your next visit to seattle.
<Date>Jan 6, 2009
<Rating>5       5       5       5       5       5       5       5
<Aspects>
9       65(visit):2     93(easy):1      138(highly):1   272(world):1    762(seattle):3  1089(getaway):1 1254(excellence):1
     1420(recomend):1        2311(historical):1
11      8(day):1        41(city):1      68(travel):1    157(light):1    165(start):1    213(meal):1     800(cup):1      1724(
     america):1 3392(adventurous):1     3896(beautifull):1      3919(joe):1
0
12      7(walk):1       16(food):1      31(front):1     57(water):1     102(near):1     115(distance):1 398(music):1    694(blue)
     :1     758(sea):1      1010(ferry):1   1563(jazz):1    4062(restraunts):1
0
4       0(staff):1      21(excellent):1 34(feel):1      141(home):1
1       6(breakfast):1
0
```

Figure 3.1: The layout of a typical review from the review file for hotel ID 72572

```
<Author>RW53
<Content>Location! Location?       view from room of nearby freeway
<Date>Dec 26, 2008
<Rating>3       4       3       2       4       3       -1      -1
<Aspects>
0
0
3       17(view):1      232(nearby):1   2318(freeway):1
1       1(location):2
0
0
0
0
```

Figure 3.2: A review with negative ratings from the review file for hotel ID 72572

```
<Value>         experience      cheap   quality         cost    expectation     honeymoon       vacation        accommodate
     discount    atmosphere
<Rooms> bathroom        bed     size    tower   window  suite   sleep   decorate        air     decor
<Location>      location        shop    station locate  bus     airport outside taxi    site    facility
<Cleanliness>clean      dirty   nonsmoking      valet   smoke   linen   dirty   smell   tidy    maintain
<Check_in/front_desk>staff      reception       wait    concierge       reservation     receptionist    checkout        office
     management      inform
<Service>       breakfast       service restaurant      food    park    coffee  cafe    drink   dinner  buffet
<Business_service>      business        center  computer        management      wifi    company route   massage facility
     website
```

Figure 3.3: The layout of the featured words file, containing the first 10 featured words for each category

```
Hotel                           Price   Location
hotel_1071829_parsed            255     Amsterdam_Noord_Holland
hotel_1126079_parsed            154     Amsterdam_Noord_Holland
hotel_189387_parsed             388     Amsterdam_Noord_Holland
hotel_189389_parsed             868     Amsterdam_Noord_Holland
hotel_189397_parsed             154     Amsterdam_Noord_Holland
hotel_190614_parsed             578     Amsterdam_Noord_Holland
hotel_190667_parsed             798     Amsterdam_Noord_Holland
hotel_284087_parsed             Unknown Florence_Tuscany
hotel_99762_parsed              370     Unknown
```

Figure 3.4: The layout of the name, price and location file, containing the first 10 hotels in the file

3.1 Making correlations between hotels

In order to perform community detection, a network graph G describing each hotel's relationship with all the other hotels needs to be made. Since this data set is data-rich and plentiful ways to interpret it exist, there could be more optimal methods for correlation. Since the MOEA framework needs to work with two objective functions, a correlation must be made for each of them, but from different data sources. The edges of G are determined by the euclidean distance of the hotels' normalised category ratings and are written into the adjacency matrix Adj. This is further described in Section 3.2. The edge-weightings are determined by the distance-weighted cosine of the normalised aspect usage counts and are written into the similarity matrix Sim, which is described in Section 3.3.

A correlation program has been made to create the input file for the MOEA framework. Algorithm 2 shows the main process of the program. The list below describes the steps of the correlation process:

1. A Hotel structure is defined, which will store a hotel's ID, location, price, an array of eight variables for the eight category ratings, a vector matrix of $[8][m]$ variables that stores each aspect's usage count, where m is the number of aspects in a category, and finally a review counter that counts the number of reviews in each hotel file.

2. The program then gets settings from a configuration file, including a threshold, a featured word weighting and the name of the file that has all the names of the hotel files written in it. The threshold is used to create the edges between hotels when performing depth-first search.

3. Review data from all the hotel files are read into the program, one at a time. It starts with finding the first review in the file. Once found, the eight category ratings from the review are checked for incompatible ratings. Only compatible ratings are added to the ratings array variable of it's Hotel structure.

4. The review's aspects' counts are then processed. If any featured words are present, the usage count of that word is multiplied by the featured word weighting. All the aspects' usage counts from the review are added to the aspect vector matrix in the Hotel structure. If a given aspect is already present in the hotel's aspect array, the usage count of that word is added to the usage count of the aspect in the structure.

5. If there is a next review, the program moves on to the next review and increments a review counter.

6. Once all reviews have been processed, the eight concatenated ratings, $ratingsTotal[n]$ where $0 \leq n \leq 7$, are normalised by dividing each rating by the number of times a compatible rating was added to this value, $ratingsCounter[n]$, such that

$$ratingsNormalised[n] = \frac{ratingsTotal[n]}{ratingsCounter[n]} \tag{3.1}$$

7. Each aspect's usage count is stored in the Hotel structure as $aspects[n][m]$, where $0 \leq n \leq 7$ and m is the amount of aspects in the category. They are also normalised by dividing each usage count by the number of reviews read for a hotel, stored as $reviewCounter$, such that

$$aspectsNormalised[n][m] = \frac{aspects[n][m]}{reviewCounter} \tag{3.2}$$

8. The program then starts to read the next hotel file in the same way as the preceding.

9. Once all hotel files have been processed, the name location price file is processed.

10. The adjacency matrix *Adj* and similarity matrix *Sim* is then calculated using the hotel data.

11. The adjacency matrix *Adj* and similarity matrix *Sim* is written to a file called input.dat.

Algorithm 2: tripadvisor hotel correlator main process

```
Input: configuration file, 1850 hotel data files, featured words file, location price file
Output: NSGA2 input file
Define Hotel structure ⟵ ID, location, price, ratings[8], aspects[8][n], review count
Declare a vector of Hotels with size 1850 ⟵ hotelVector
Get configurations ⟵ distance threshold $d_{thr}$, featured aspect weighting
Get featured words
foreach hotel file do
    while not end of file do
        Find next review
        foreach i ⟵ category do
            if $review.ratings[i] \neq -1$ then
                Add review.ratings[i] to hotel.ratings[i]
                Increment ratingsCounter[i]
        foreach i ⟵ category do
            Get review.category[i].aspectCount
            Set $j_{max}$ to aspectCount
            foreach j ⟵ aspect do
                if $review.aspect[i][j] = featured$ then
                    Multiply review.aspect[i][j].usageCount by featured aspect
                     weighting
                if $review.aspect[i][j].ID + 1 > hotel.aspects[i].size$ then
                    Add $n = ID - size$ empty elements to hotel.aspects[i]
                Add review.aspect[i][j].usageCount to
                 hotel.aspects[i][ review.aspect[i][j].ID ].usageCount
        Increment reviewCounter
    Normalise category ratings
Get location and price
Calculate adjacency Adj and similarity Sim matrices
Write Adj and Sim to file (NSGA2 input file)
```

3.2 Determining edges with euclidean distance

As previously discussed, every hotel has eight ratings that could be used to find similar hotels. If only one rating category was available, finding similar hotels would be straight forward; find all hotels with a similar rating value. In the case of eight categorical ratings, finding similarity requires more dimensions, one for each category. Describing the eight dimensions in euclidean space allows a distance metric, or distance function, to describe the similarity of two hotels.[2] The distance metric d on $\mathbb{R}^n$ is defined by

$$d(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}|| = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{3.3}$$

where $\mathbf{x}$ and $\mathbf{y}$ are some vectors describing $hotel_i$ and $hotel_j$ ratings respectively, in $n = 8$ dimensional space. As the euclidean distance d between $\mathbf{x}$ and $\mathbf{y}$ were to decrease, the similarity between $hotel_i$ and $hotel_j$ should increase. Since an edge with no weighting can only take a binary value, a distinction needs to be made, whether or not $hotel_i$ and $hotel_j$ are similar enough that an edge should be created between them. The distance d at which it is considered an adequate distance for an edge to be created depends mostly on goal, to create a connected graph.

3.2.1 Creating a connected graph with depth-first search

For the purposes of performing community detection on this data set, all vertices in the graph should have some relation to every other vertex such that from a randomly selected vertex, a path could be created to every other vertex by following the edges. A depth-first search (DFS) is a recursive algorithm for traversing tree or graph data structures that can be used to determine whether some graph, G, is a connected graph.[7] As the DFS algorithm recursively traverses the graph by using the edges described by the adjacency matrix Adj, it marks each vertex it has visited with a boolean variable to true. If the number of boolean variables set to true is equal to the number of vertices in the graph, then the graph is connected. If this is not the case, the graph has to be modified such that some more edges are created.

This is done by using a distance threshold d_{thr}. If the distance d between $hotel_i$ and $hotel_j$ is less than or equal to d_{thr}, an edge is created. By incrementing d_{thr} by some constant value, the number of created edges should increase. Each time d_{thr} is incremented, adj is recalculated using the new d_{thr} value and another DFS is performed. d_{thr} is initialised to the value retrieved from the configuration file and is incremented until a connected graph emerges. Algorithm 3 shows this procedure.

3.2.2 Reducing graph density with Kruskal's algorithm

Using Algorithm 3 to create a connected graph results in a very dense graph, some 1.5 million out of a possible 1.7 million being created until a connected graph emerges. In an effort to reduce the density of the graph, a modified version of Kruskal's algorithm is used.

Kruskal's algorithm is a minimum-spanning-tree algorithm which finds an edge of the least possible weight that connects any two trees in a graph. It finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimised. [8]

Algorithm 3: Connected graph procedure

```
Input: $d_{thr}, normalisedRatings$
Data: Adjacency matrix $Adj$, vector of hotels $hotelVector$
Output: Connected graph $G_{connected}$

$connected = false$
while $connected = false$ do
    foreach $i$ do
        for $j$ do
            if $i$ != $j$ then
                Get $hotel_i.ratings$ from $hotelVector[i]$
                Get $hotel_j.ratings$ from $hotelVector[j]$
                Calculate $d$ between $hotel_i$ and $hotel_j$
                if $d <= d_{thr}$ then
                    $Adj[i][j] = 1$
                else
                    $Adj[i][j] = 0$
    Create a graph $G_{temp} = (V, E)$ where $|V| = 1850$
    Set $G_{temp}.E$ as described by $Adj$
    Perform recursive DFS on $G_{temp}$ with root $V(0)$
    Get the number of vertices visited in the DFS $vertexVisitedCount$
    if $vertexVisitedCount = |V|$ then
        $connected = true$
    else
        Increment $d_{thr}$ by 0.005
$G_{connected} = G_{temp}$
```

The modification created edges until a minimum-spanning-tree emerged. This decreased the number of edges in the original graph, while keeping it connected. The two inputs are the raw euclidean distance values d found in Algorithm 3 and Adj from $G_{connected}$. The distance d is used to give weightings to the edges of $G_{connected}$. These weighted edges are then used to create a graph, A, to run Kruskal's algorithm on. Algorithm 4 shows the modified version. Notice the lack of $UNION(u, v)$ in the for loop in the original algorithm.

Algorithm 4: Modified Kruskal's algorithm

```
Input: Euclidean distances for edge weights $d$, $Adj$
Output: A

A = ∅
foreach $v \in G.V$ do
    MAKE_SET(v)
foreach (u,v) in G.E ordered by weight(u,v), increasing do
    if $FIND\_SET(u) \neq FIND\_SET(v)$ then
        $A = A \cup \{(u,v)\}$
        UNION(u,v)
    else
        UNION(u,v) ⟵ Modified
```

3.3 Determining edge weightings with Dw-Cosine

Correlating $hotel_i$ and $hotel_j$ based on the normalised aspect usage counts is achieved by using Distance-weighted Cosine (Dw-cosine). Dw-cosine is mostly used for text classification and similarity descriptions.[4] Cosine measures the angle between two vectors and is calculated as a dot-product of two normalised vectors. Given two n dimensional vectors $\mathbf{v}$ and $\mathbf{w}$, the Cosine similarity between them is calculated as

$$Cosine(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \bullet \mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{n} v_i \times w_i}{\sqrt{\sum_{i=1}^{n} v_i^2}\ \sqrt{\sum_{i=1}^{n} w_i^2}} \tag{3.4}$$

In the context of pure mathematics, the Cosine similarity is perfect. However, in text classification and mining applications, it may not provide suitable results. The Cosine of two text segments will describe the angle between them dimensionally, but it may not accurately describe the commonality of text since text classification relies on finding common terms between two segments. The Cosine similarity can also be biased by the features of higher values, but it doesn't care much about how many features two vectors share. To account for this bias, the weighted hamming distance (WHD) is used. The WHD is defined as

$$WHD(\mathbf{v}, \mathbf{w}) = \frac{\sum_{i=1}^{n} v_i(1 - sign(v_i w_i))}{\sum_{i=1}^{n} v_i} + \frac{\sum_{i=1}^{n} w_i(1 - sign(w_i v_i))}{\sum_{i=1}^{n} w_i} \tag{3.5}$$

It is combined with the Cosine to give Dw-cosine, defined as

$$Dw\text{-}cosine = \frac{Cosine}{Dist^2 + Cosine} \tag{3.6}$$

where $Cosine$ is as defined in Equation 3.4 and $Dist$ is the HWD, as defined in Equation 3.5.

Equation 3.6 is used to find the similarity of $hotel_i$ and $hotel_j$ aspects. A Dw-cosine value is found for each of the eight aspect categories and only the highest Dw-cosine value is selected to be written to the Sim matrix. Since the Sim matrix is identical along the principal diagonal, Sim_{ij} and Sim_{ji} are equal so calculations only need to be run once for both. All the aspects in each category vector in the $hotelVector$ have been placed in ascending order by aspect ID in Algorithm 2, so when calculating the Dw-cosine between $hotel_i$ and $hotel_j$, each aspect from $hotel_i$ matches up with the same ID as the aspects in $hotel_j$. Algorithm 5 shows this procedure.

Algorithm 5: Distance-weighted cosine procedure

```
Input: hotelVector
Output: $Sim$
$dwCosine = 0$
foreach $i$ do
    foreach $j$ do
        if $i \neq j$ & $i < j$ then
            Get $hotel_i.aspects$ from $hotelVector[i]$
            Get $hotel_j.aspects$ from $hotelVector[j]$
            foreach $k \longleftarrow category$ do
                if $hotel_i.aspects[k].size \neq hotel_j.aspects[k].size$ then
                    Add empty elements to the smaller until equal in size
                foreach $l \longleftarrow aspect$ do
                    Normalise $hotel_i.aspects[k][l].usageCount$
                    Normalise $hotel_j.aspects[k][l].usageCount$
                Calculate $result$ = Dw-cosine of $hotel_i.aspects[k]$ and
                 $hotel_j.aspects[k]$)
                if $result > dwCosine$ then
                    $dwCosine = result$
            $Sim_{ij} = Sim_{ji} = dwCosine$
```

4 Results Analysis

The framework and all other correlation programs were written in C++. Due to the amount of time required to run more than a hundred generations using the complete data sets on a standard computer, the MOEA framework executable was run on the university's Research Compute Grid (RCG), which is part of the High Performance Computing facility[1]. By using the RCG, computation time was reduced from a week to about 18 hours. The parameters that were used to run the tests are shown in Table 4.1.

Table 4.1: Test parameters used in the tests

Test	Random seed	N_{pop}	N_{gens}	$pmut_{comm}$	$pcross_{comm}$	eta_c	eta_m
CiteSeer	2312	200	2000	0.9	0.9	0.1	0.1
Cora	7865	200	2000	0.9	0.9	0.1	0.1

4.1 CiteSeer data set

The framework was executed for 2000 generations with the CiteSeer data set. The resulting pareto front is shown in Figure 4.2. It can be seen that the solutions do not advance much after the 1000th generation, which is provided for reference. This indicates that the solutions are converging to their respective optimal values. To analyse the performance of the framework, the clusters of three individuals from the pareto front were selected, Ind_A from the top edge, Ind_B from the centre and Ind_C from the bottom edge.

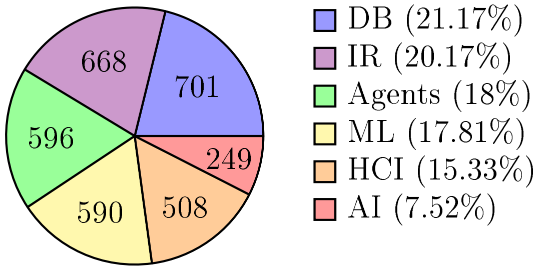


Figure 4.1: The predefined class distributions for the CiteSeer data set

To interpret the clustering of the individuals, a comparison to the predefined classes was made. Table 4.2 lists the cluster IDs for Ind_A, Ind_B and Ind_C. It can be seen that there are many small clusters that have less than 1% of the total vertices, making the community structure quite fragmented, with some clusters only having a single vertex. The number of clusters seems to decrease for individuals lower in the pareto front. Those higher in the front seem to have a lot more clusters, including small ones, making up the solution. Some can have up to 200 different clusters, with up to 20% of vertices being in small clusters.

[1]https://www.newcastle.edu.au/research-and-innovation/resources/research-computing-services/high-performance-computing-hpc

Table 4.2: A list of the clusters by ID and size in percentage relative to the vertex count. The *Small* ID is a concatenation of all clusters that have a size percentage of less than 1%, with the number of clusters comprising it being indicated to the left and to the right the combined cluster size

Ind_A		Ind_B		Ind_C	
NMI	0.076	**NMI**	0.057	**NMI**	0.016
ID	**Size** (%)	**ID**	**Size** (%)	**ID**	**Size** (%)
2	9.84	31	23.49	1	30.43
3	8.73	30	20.17	2	22.01
5	5.65	34	18.66	3	20.44
4	5.37	37	11.08	5	12.89
7	5.34	39	9.09	4	12.26
8	4.65	40	5.77	7	1.3
10	4.5	36	3.89	8 Small	0.66
12	3.56	38	3.41		
11	3.11	46	1.15		
14	3.02	21 Small	3.29		
1	2.99				
13	2.93				
16	2.48				
9	2.32				
18	2.26				
15	1.57				
19	1.54				
24	1.33				
23	1.33				
21	1.33				
20	1.09				
17	1.06				
26	1				
153 Small	24				

Using the predefined classes shown in Figure 4.1 as a ground truth, a similarity score using Equation 2.2 was used to determine the quality of partitions. Similar research has been done with several different community detection algorithms where these data sets were used to help determine the performance.[11] Table 4.3 shows some of the NMI scores for link and content based algorithms, where the LCF algorithm scored the lowest and PCL-DC scored the highest. The score from Ind_A of the CiteSeer front and Ind_A of the Cora front are used to represent the MOEA framework scores. The CiteSeer data set has lower scores across all algorithms, including the MOEA framework.

Table 4.3: Link and content based NMI score from different community detection algorithms and the MOEA framework

Data set	PCL-DC	LCF	MOEA
CiteSeer	0.2921	0.0934	0.076
Cora	0.5123	0.1227	0.151

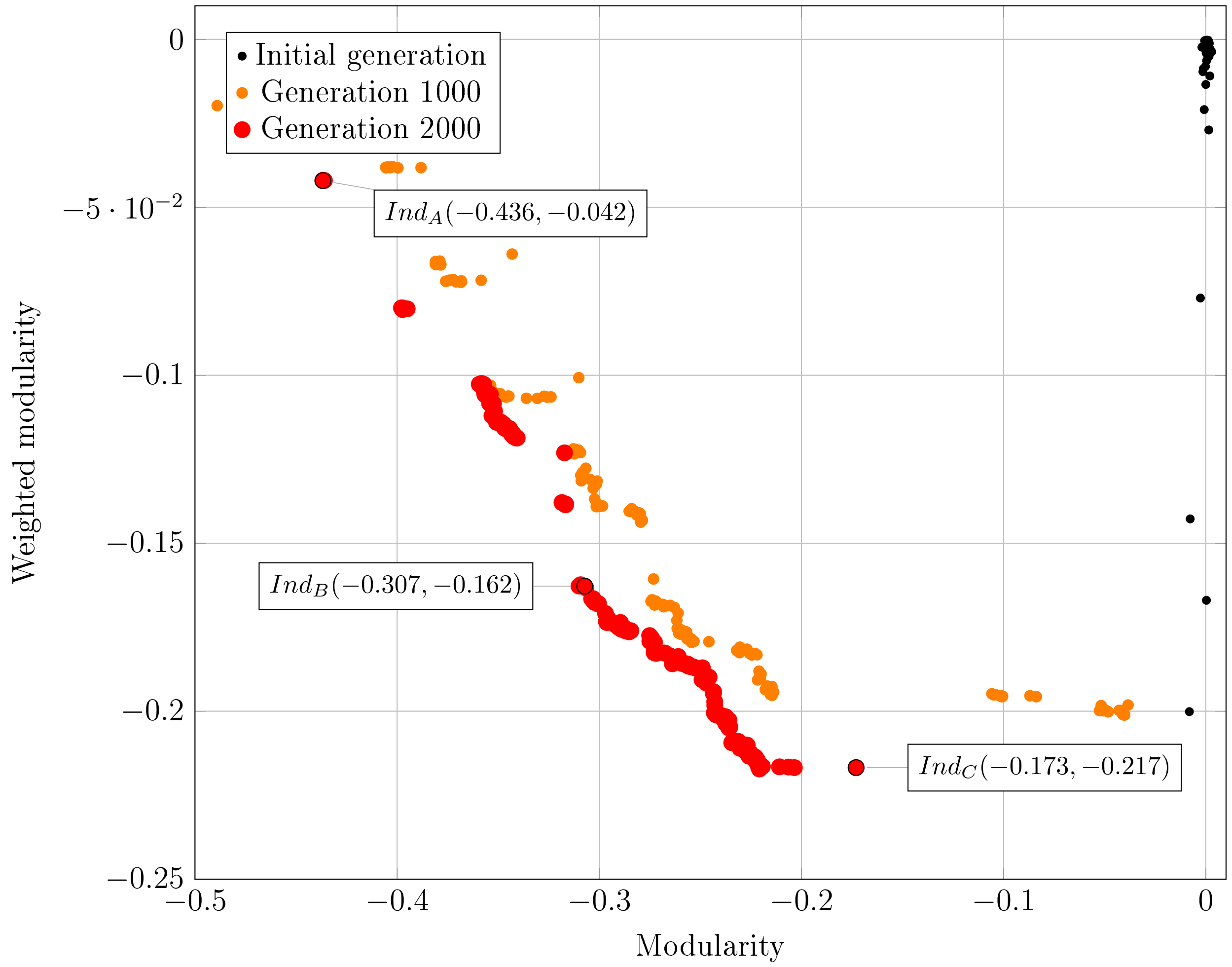


Figure 4.2: The pareto front of the CiteSeer data set solutions after 2000 generations. Generation 1000 is also included for reference

The pareto front is quite fragmented and there are areas where no solutions exist. Some local optima are possibly causing the grouping of all the individuals. Another observation is that the NMI score decreases with the higher weighted modularity scores, but the number of clusters decrease. This could be caused by most predefined classes being shared amongst most clusters, creating a higher state of entropy.

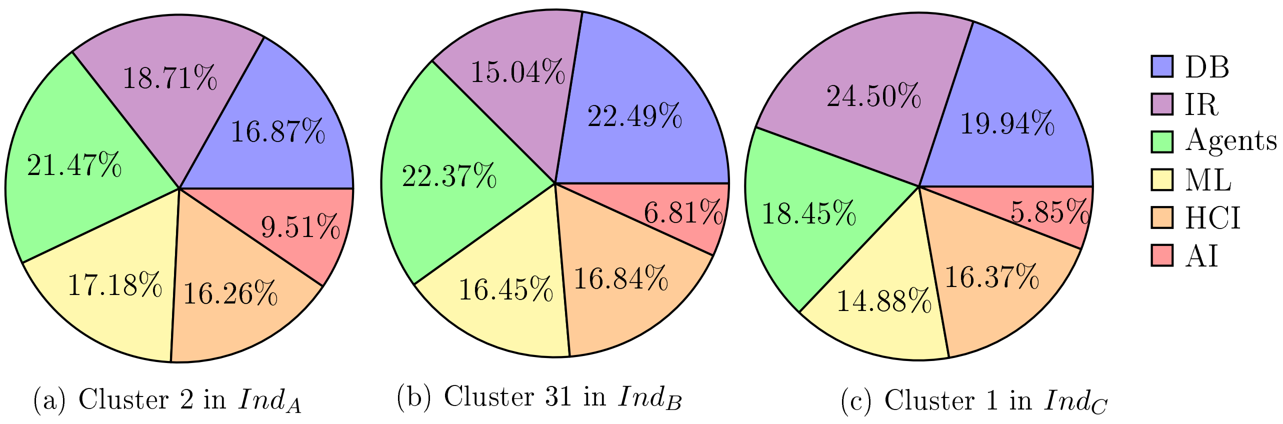


Figure 4.3: Class distributions for CiteSeer clusters

4.2 Cora data set

The Cora data set was also run for 2000 generations with the parameters listed in Table 4.1. The similarity score was calculated again using the NMI and was found to be 0.151 for Ind_A which is the highest score of the chosen individuals. Table 4.4 shows a list of all the clusters for the three chosen individuals, the same way as the CiteSeer results have been presented. The pareto front is not as split up as the CiteSeer front and seems to be much more dominated by the weighted modularity objective function.

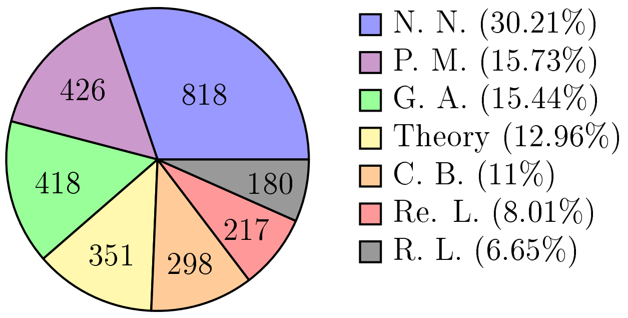
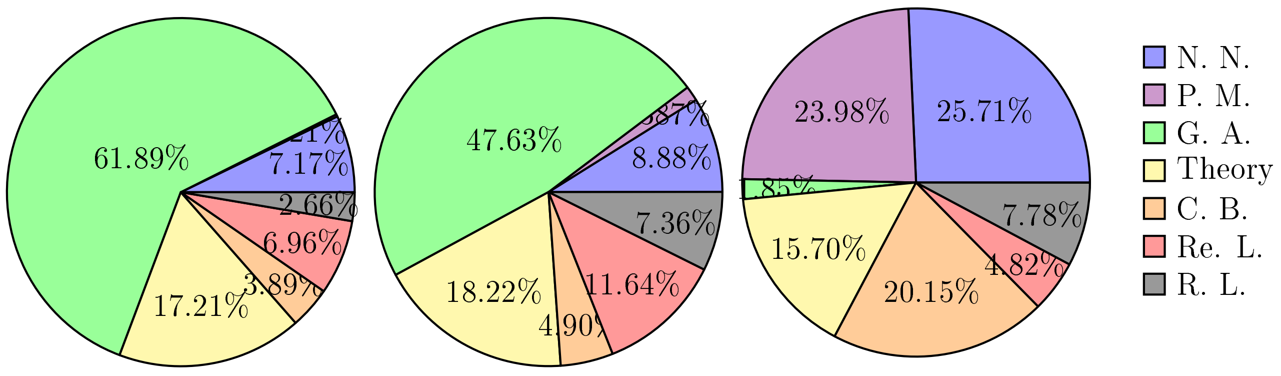


Figure 4.4: The predefined class distributions for the Cora data set

Since the ground truth was found using the NMI, the size of the clusters doesn't affect the score. The biggest impact is the class distribution of the clusters. It can be seen in Figure 4.4 that the highest NMI score was achieved by Ind_A. After investigating the class distribution of the largest cluster in Ind_A, the majority of nodes come from the Genetic Algorithms class, about 61%, which is shown in Figure 4.5a. As the modularity score is minimised in the pareto front, the class distribution becomes more spread out across all classes, as evident in Figures 4.5b and 4.5c. This is accounting for the decrease in NMI score and shows that it has an ample affect, creating greater entropy in the partitions. It seems that the weighted modularity fitness score decreases the similarity score overall, even in the CiteSeer results.

Table 4.4: A list of the clusters by ID and size in percentage relative to the vertex count

Ind $_A$		***Ind*** $_B$		***Ind*** $_C$	
NMI	0.151	**NMI**	0.138	**NMI**	0.125
ID	**Size** (%)	**ID**	**Size** (%)	**ID**	**Size** (%)
1	18.02	147	24.11	1	29.87
3	9.97	149	15.25	3	21.38
2	8.68	151	10.34	2	17.87
5	6.87	153	6.57	5	8.31
4	5.47	155	4.32	7	3.8
7	5.06	158	3.25	8	3.73
10	4.69	161	2.99	9	3.43
9	4.39	160	2.77	11	2.58
12	3.25	157	2.73	12	2.47
13	3.06	166	2.18	14	1.96
8	2.95	163	1.92	13	1.59
14	1.92	165	1.33	24 Small	2.99
15	1.59	172	1.07		
16	1.55	168	1.03		
17	1.37	171	1		
18	1.33	132 Small	19.13		
20	1.14				
22	1.07				
24	1				
117 Small	17.61				



(a) Cluster 1 in Ind_A (b) Cluster 147 in Ind_B (c) Cluster 1 in Ind_C

Figure 4.5: Class distributions for Cora clusters

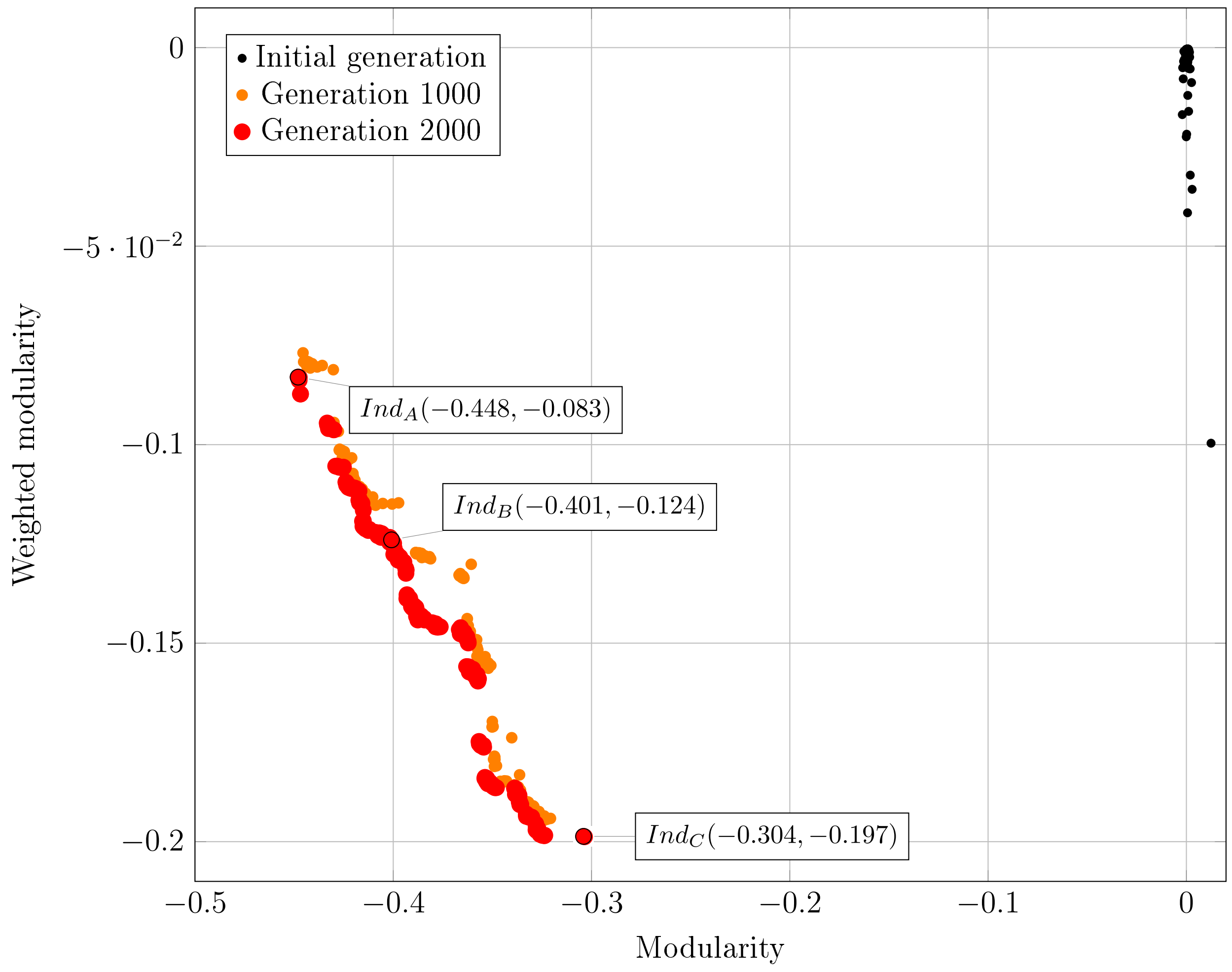


Figure 4.6: The pareto front of the Cora data set solutions after 2000 generations. Generation 1000 is also included for reference

4.3 tripadvisor data set

The graph created for the tripadvisor data set had 527,100 edges. It was run for 2000 generations and seems to have been dominated by the weighted modularity fitness function, as seen in the resulting pareto front, shown in Figure 4.7. After reviewing the content of three individuals in the same way as before, the cluster count was seen to only reach a maximum of three in Ind_B and only one cluster containing all the vertices in Ind_C. The clusters are shown in Table 4.5.

Table 4.5: A list of the clusters by ID and size in percentage relative to the vertex count

Ind$_A$		***Ind$_B$***		***Ind$_C$***	
ID	**Size** (%)	**ID**	**Size** (%)	**ID**	**Size** (%)
1	59.30	5	84.27	2	100
2	40.70	4	15.62		
		6	0.11		

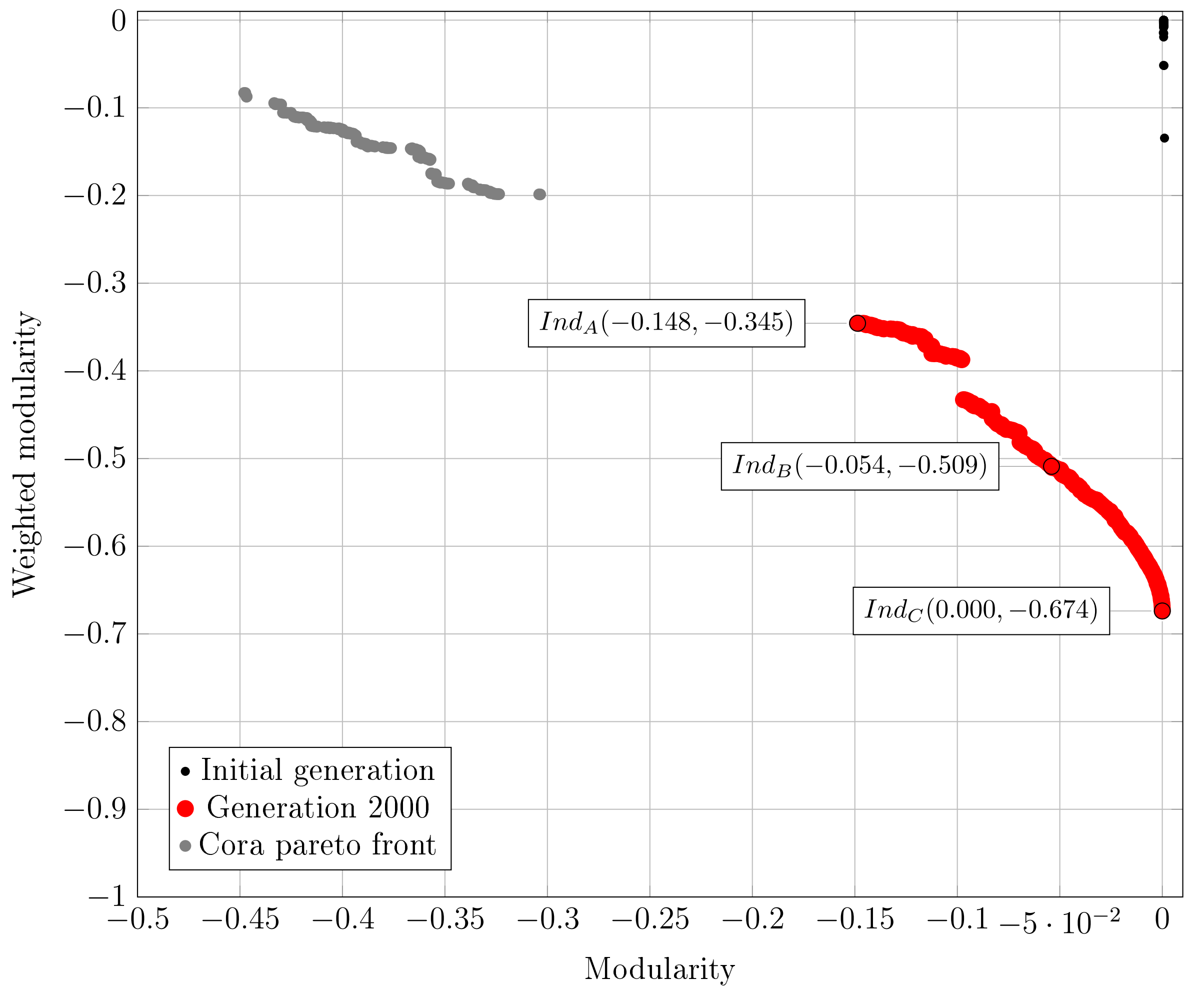


Figure 4.7: The pareto front of the tripadvisor data set solutions after 2000 generations

5 Framework Modifications

Two modifications were made, a refinement of the community representation and fixing a memory leak that was occurring as generations advanced.

5.1 Refinement of community representation

The way that communities were represented was modified slightly to increase the effectiveness of the crossover function. Figure 6 shows the procedure that was written to improve the representation. The pareto front of the Cora data set after applying this refinement is shown in Figure 5.1 and the cluster list is shown in Table 5.1.

The goal of the refinement was to change the order of community ID tags that was being assigned to vertices. All the vertices in the largest cluster were to get the lowest ID. If two clusters had the same cluster size, the one that had the lowest vertex ID was prioritised. This would continue until all clusters' community tags were reassigned.

Table 5.1: A list of the clusters by ID and size in percentage relative to the vertex count

Ind_A		***Ind_B***		***Ind_C***	
NMI	0.045	**NMI**	0.046	**NMI**	0.03
ID	**Size** (%)	**ID**	**Size** (%)	**ID**	**Size** (%)
1	37.17	1	48.89	1	60.12
2	24.04	2	24.11	2	16.80
3	7.20	3	6.46	3	6.43
4	6.43	4	3.66	4	3.10
5	4.87	5	3.25	5	2.22
6	3.36	6	1.99	6	2.03
7	2.95	7	1.96	7	1.96
8	2.25	8	1.96	8	1.92
9	1.98	9	1.96	9	1.85
10	1.98	10	1.92	10	1.77
11	1.92	11	1.74	19 Small	1.81
12	1.88	15 Small	2.10		
13	1.77				
13 Small	2.22				

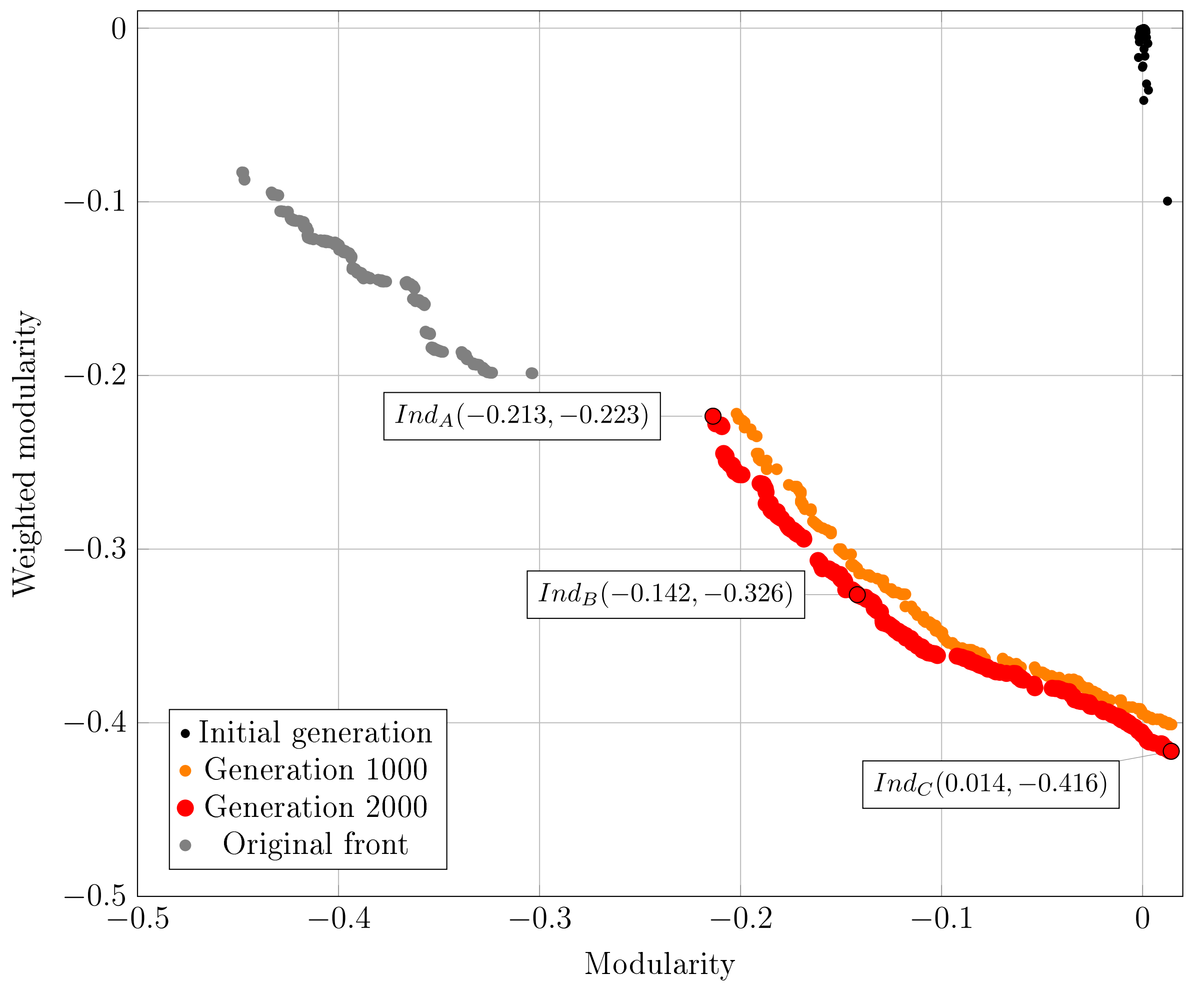
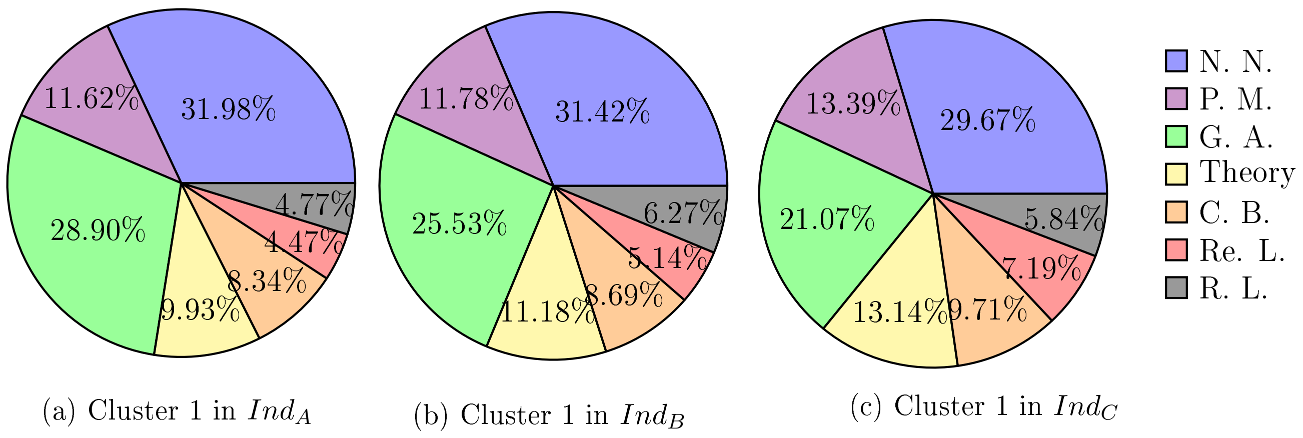


Figure 5.1: The pareto front of the Cora data set solutions after refinement of community representation



(a) Cluster 1 in Ind_A (b) Cluster 1 in Ind_B (c) Cluster 1 in Ind_C

Figure 5.2: Class distributions for Cora clusters after community representation refinement

Algorithm 6: Community representation refinement procedure

```
Input: $commSize \longleftarrow$ A list of tags that give the community size for each vertex,
       $vertexSet \longleftarrow$ A list of the original community IDs for each vertex
Output: $refinedVertexSet \longleftarrow$ A refined list of community IDs
$highest = 1 \longleftarrow$ Highest community size
foreach $i \longleftarrow$ Vertices in graph do
    $commSize[i] = 1$
    $refinedVertexSet[i] = 0$
    foreach $j \longleftarrow$ Vertices in graph do
        if $vertexSet[i] = vertexSet[j]$ & $i \neq j$ then
            Increment $commSize[i]$
    if $commSize[i] > highest$ then
        $highest = commSize[i]$
$commCounter = 1 \longleftarrow$ Number of communities
for $size > 0 \longleftarrow size_{init} = highest$, Decrementing do
    foreach $i$ do
        if $commSize[i] = size$ then
            if $refinedVertexSet[i] = 0$ then
                $refinedVertexSet[i] = commCounter$
                foreach $j \longleftarrow j_{init} = i + 1$ do
                    if $vertexSet[j] = vertexSet[i]$ then
                        $refinedVertexSet[j] = commCounter$
                    Increment $commCounter$
```

5.2 Improving dynamic memory allocation

The framework had a memory leak where community structures were dynamically allocated but not deallocated when advancing generations. The size of the memory leak is then proportional to the number of individuals in a population. A procedure was written to handle this leakage by searching for any community structures in the parent population that were not going to be used or passed on to the child population, and deleting them.

6 Conclusion

In conclusion, the results of the CiteSeer and Cora tests have shown that the content based objective function, the weighted modularity, seems to cluster vertices with a similar distribution as that of the original predefined classes. This is most evident in the Cora data set, seen in Figure 4.3c, how cluster 1 in Ind_A is mostly comprised of one class and has a high NMI score, relative to Ind_B and Ind_C. This is due to a modularity score of 0.448. As this same front is modified by refining the community representation, it can be seen that a lower modularity score is correlated with a lower NMI score.

Future work to further the framework can be summarised by the following list:

1. The content fitness function, the weighted modularity, needs reanalysis. It can be seen from the results that it diminishes the similarity score in the CiteSeer and Cora data sets
2. Investigate an alternative of how the communities are represented
3. The dynamic memory allocation of the community structures needs to be investigated for improved memory management. The current implementation of removing unused community structures has a high computation time and can be reduced if the structure has a destructor function implemented
4. Replace the community, individual, and population structures with more complete classes. Doing so will ensure that any memory dynamically allocated should be automatically handled by its own destructor, provided it is present

Bibliography

[1] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (April 2002)

[2] Deza, M.M., Deza, E.: Encyclopedia of distances. Springer-Verlag Berlin Heidelberg p. 94 (2009)

[3] Fortunato, S.: Community detection in networks: A user guide (2016)

[4] Li, B., Han, L.: Distance weighted cosine similarity measure for text classification. In: Yin, H., Tang, K., Gao, Y., Klawonn, F., Lee, M., Weise, T., Li, B., Yao, X. (eds.) Intelligent Data Engineering and Automated Learning – IDEAL 2013. pp. 611–618. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[5] Meek, E.: Multi-objective genetic algorithm for community detection. Work Integrated Learning Report, University of Newcastle, Australia (November 2016)

[6] Naeni, L.M.: Memetic algorithms for community detection and clustering problems. Ph.D. thesis, University of Newcastle, Australia (September 2016)

[7] Tarjan, R.: Depth-first search and linear graph algorithms. SIAM J. Comput. 1(2), 146–160 (June 1972)

[8] Thomas, C., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. p. 631. MIT (2009)

[9] Wang, H., Lu, Y., Zhai, C.: Latent aspect rating analysis on review text data: A rating regression approach. KDD'10 pp. 783–792 (2010)

[10] Wang, H., Lu, Y., Zhai, C.: Latent aspect rating analysis without aspect keyword supervision. KDD'11 pp. 618–626 (2011)

[11] Yang, T., Jin, R., Chi, Y., Zhu, S.: Combining link and content for community detection: A discriminative approach. KDD'09 1(2), 927–935 (June 2009)

[12] Zahn, B., Ucherdzhiev, I., Szeles, J., Botzheim, J., Kubota, N.: Optimisation of a proportional-summation-difference controller for a line-tracing robot using bacterial memetic algorithm. International Conference on Intelligent Robotics and Applications pp. 362–372 (2016)