Evolution program for network model

Xenos Alexandros Orestis

supervised by

Pierluigi Frisco

Master of Science
Heriot-Watt University
Edinburgh
2011
Declaration

I, Xenos Alexandros Orestis, confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form are properly acknowledged at any point of their use. A list of the references employed is included.

Edinburgh 18 August 2011

Signature
Acknowledgments

I would like to thank my project supervisor, Pierluigi Frisco for his constant guidance and support.

I would also like to thank David Corne for his valuable advice.

Finally I would like to thank my family and my friends for always supporting me.
Abstract

Recently Pierluigi Frisco proposed a new network model. In this model nodes are represented as words over an alphabet and edges between nodes are added according to some distance measures. A program that enables the user to create networks that follow this model has also been implemented. In order to achieve networks with certain topologies, the user has to tune various properties which are provided as input to the algorithm.

In this research an evolutionary algorithm has been developed, which automates the properties' tuning procedure. This procedure constitutes a multiobjective optimization problem. In general, evolutionary algorithms have been widely used for solving optimization problems and show significantly good performance in multiobjective optimization problems.

Among other parameters, the implemented algorithm requires from the user three network characteristics that need to be reached. These characteristics are the average degree, the average path length and the average cluster coefficient. These values form the objectives that the algorithm tries to optimize. Once the algorithm terminates, the user receives as output, those network's properties that were found to be optimal for the certain input.

The program was tested by running simulations of the Escherichia coli protein-protein interaction network. The test – runs were conducted on a computer cluster and the results improved previous attempts to replicate this network.
## Contents

Chapter 1 : Introduction ........................................................................................................8
  1.1 Aims of Research........................................................................................................8
  1.2 Requirements...........................................................................................................8
  1.3 Project Plan...............................................................................................................9
  1.4 Professional, Legal and Ethical Issues.................................................................9

Chapter 2 : Literature Review .............................................................................................10
  2.1 Graph Theory..........................................................................................................10
    2.1.1 Basic Concepts..............................................................................................10
    2.1.2 Basic Graph Quantities..............................................................................11
    2.1.3 Graph Models.............................................................................................12
      2.1.3.1 Random Graph Model (Erdős–Rényi model)..................................12
      2.1.3.2 Small World Model (Watts and Strogatz model)..........................13
      2.1.3.3 Barabasi – Albert Model...............................................................14
      2.1.3.4 Structured Nodes model (SN – model)........................................14
  2.2 Evolutionary Algorithms .........................................................................................18
    2.2.1 History........................................................................................................18
    2.2.2 Components of an Evolutionary Algorithm .............................................18
      2.2.2.1 Representation..............................................................................18
      2.2.2.2 Fitness function..........................................................................19
      2.2.2.3 Population..................................................................................19
      2.2.2.4 Parent Selection Mechanism.....................................................19
      2.2.2.5 Variation Operators ......................................................................19
      2.2.2.6 Survivor Selection Mechanism..................................................20
  2.3 Multi – Objective optimization...............................................................................20
    2.3.1 Basic Concepts.............................................................................................20
    2.3.2 Evolutionary Algorithms and multi – objective optimization.................22
      2.3.2.1 Aggregating functions approach..............................................22
      2.3.2.2 Population – based approach..................................................22
      2.3.2.3 Pareto – based approach.......................................................22

Chapter 3 : Program Description ......................................................................................26
  3.1 Overview..................................................................................................................26
Figures

Figure 1 : Gantt Chart .....................................................................................................................9
Figure 2 : A simple directed graph................................................................................................10
Figure 3 : Power law degree distribution......................................................................................11
Figure 4 : Random Graph with 12 nodes and 24 edges.................................................................12
Figure 5 : Degree distribution in random graphs........................................................................13
Figure 6 : Small word network......................................................................................................13
Figure 7 : Scale – free network’s degree distribution in a log – log plot.................................14
Figure 8 : Evolutionary algorithm – flow chart........................................................................20
Figure 9 : Optimization of two conflicting parameters.............................................................21
Figure 10 : Graphical representation of the Pareto – optimal surface for a two objective problem........................................................................................................21
Figure 11 : NSGA flow chart.........................................................................................................24
Figure 12 : “The Pareto Archived Evolution Strategy.................................................................25
Figure 13 : Program’s overview flow chart..................................................................................26
Figure 14 : Gaussian distribution (μ: mean value, σ: standard deviation).................................31
Figure 15 : Program’s analytical flow chart ...............................................................................34
Figure 16 : Class Diagram............................................................................................................58
Chapter 1

Introduction

The Structured Nodes model (SN – model) is a new network model that was proposed by Pierluigi Frisco [11]. The creation of networks according to this model can be done by using a program which receives as input a property file (containing the network properties). The user is able to study the generated network from a set of output files that the program returns. In order to achieve certain topological measures of a network, the user has to tune up the appropriate properties in the input property file. This tuning process is done manually. However, it is difficult to predict the topological properties of a network given the property file. A more detailed description of the SN – model and the algorithm that implements it is made in the literature review chapter.

1.1 Aims of research

The aim of this project was the development of an evolutionary algorithm which would function as an extension of the program that implements the SN – model. The role of the implemented algorithm is to produce optimal values of certain network properties so that the resulting network will have the specified by the user characteristics. Apart from implementation this research aimed at finding those evolution parameters that provide optimal solutions (concerning the match between the created network's topological properties and the target values that the user has defined). Some of the evolution parameters are the size of the population, the number of generation and probabilities of mutation and crossover. All of them are explained in more details in chapter 3. In order to find the more effective combination of the evolution properties, many test-runs have been made. These were conducted on the Escherichia coli protein-protein interaction network and the results showed that despite the fact that there wasn't a very close fit to the target values, the average values (over 100 runs) of the optimal networks had a 20% variance from the target values. The analysis of the collected data is made in chapter 4.

1.2 Requirements

As specified in the startup of this project the functional requirements of the application are:

- The user should be able to specify the network's characteristics (average degree, average path length and average clustering coefficient) that the program will try to satisfy.
- The user should be able to change some of the parameters that affect the evolution procedure.
• The user should be informed of the set of properties that optimized the specified network's characteristics.

The developed application meets all the aforementioned requirements.

1.3 Project Plan

The plan of the project can be seen in the Gantt Chart below:

![Gantt Chart](image)

**Figure 1:** Gantt Chart

Most of the tasks have been completed in time. However a problem with the fitness function of the evolution program was discovered during Testing and Evaluation. The problematic fitness function was replaced by a completely different approach, both described in more details in chapter 3. Unfortunately this change had lead the project slightly out of plan as the test-runs had to be conducted again with the new fitness function.

1.4 Professional, Legal and Ethical Issues

This project doesn't raise any professional, legal or ethical issues. There is no need for data protection or any other security mechanism. Moreover the evaluation of the software doesn't associate with people using it. However it is important to mention that the application that has been developed makes use of an existing program that implements the SN – model. This program has been created by the supervisor of my dissertation project, Mr Pierluigi Frisco. The SN – model program is a software distributed under the GNU General Public License [13].
Chapter 2
Literature Review

As already described, this dissertation project aims at the development of an evolutionary algorithm which will be responsible for the tuning of the input parameters of the SN – model program in order to have specific networks as output. The literature which relates to this project can be divided into three parts. The first part includes concepts that refer to networks. The second part reviews evolutionary algorithms while the third is addresses issues relevant to multi – objective optimization.

2.1 Graph Theory

2.1.1 Basic Concepts

A graph is a mathematical object which represents the relations between two sets [3]. These are the set of vertices (nodes) and the set of edges.

Type of the graph

A graph can be:
- Directed – edges can be traversed in one direction
- Undirected – there isn't a direction constraint in the edges of the graph
- Weighted – each edge has a value (cost)
- Oriented – a special case of directed graphs in which no loops or multiple edges are allowed [3].

Figure 2 : A simple directed graph [1]
2.1.2 Basic Graph Quantities

Each graph has a number of vertices. This number is referred as the *order* of the graph. Moreover the number of edges that a graph has is termed as *size* of the graph. If a graph doesn't have any edges then the graph is *empty*. In contrast if all possible edges exist then the graph is *complete*. The number of all possible edges of a graph with \( n \) nodes can be easily calculated [3]:

\[
e_{\text{max}} = \frac{n(n-1)}{2}
\]

\( e_{\text{max}} \): number of possible edges  
\( n \): number of nodes

Each node in a graph has a *degree* which relates to the number of edges that this node has. In the case of a directed graph this property is divided to the *in – degree*, which refers to the number of the edges that end to the node, and the *out – degree*, which refers to the number of edges that start from the node.

A graph is called *connected* when a path (edge or sequence of edges) for each pair of nodes exists. Otherwise the graph is *unconnected* [5].

The *distance* between two nodes is the minimum number of edges that need to be crossed in order to reach one node from another [3].

The *diameter* of a graph is a value that indicates the maximum possible distance between any two nodes [3]. The diameter can be also defined in average value.

The *degree distribution* of a graph is a function that describes how the degrees of the nodes are distributed. In other words this function relates the number of nodes with a certain degree to the degree itself [3].

![Power law degree distribution](image)

Figure 3 : Power law degree distribution

In a graph each node has neighboring nodes with which it is connected. For every pair of neighboring nodes a triangle is potentially formed. The number of possible triangles is analogous to the number of neighboring nodes and can be calculated:

\[
\text{possible triangles} = n_{i} (n_{i} - 1) / 2
\]

\( n_{i} \): number of neighboring nodes
The number of the actually existing triangles divided by the number of possible triangles is called cluster coefficient [5]. The sum of the cluster coefficients of all nodes divided by the number of the nodes is termed as cluster coefficient of the graph [5].

2.1.3 Graph Models

Models are widely used across all scientific disciplines. They are used to describe in a simplified way real complex concepts. In the field of graphs there are models that represent real networks. The aim of these models is to abstractly describe the structure and understand the dynamics of complex networks. A notion to some of these models is made below.

2.1.3.1 Random Graph Model (Erdős–Rényi model)

This is a simple model for generating graphs which was proposed by Paul Erdős and Alfred Rényi. It proposes the construction of a network with specified order (certain number of nodes) in which edges between nodes are added randomly. Each edge is added according to a defined probability $p$ [3].

![Random Graph with 12 nodes and 24 edges](image)

**Figure 4**: Random Graph with 12 nodes and 24 edges [19]

The number of edges in the resulting network is proportionate to the number of nodes and the probability.

$$E = p \times \left(\frac{n(n-1)}{2}\right)$$

$E$ : number of edges  
$p$ : probability  
$n$ : number of nodes

The degree distribution in the generated networks follows a Poisson distribution with mean value $2 \times E / n$, where $E$ is the number of edges and $n$ the number of nodes.
A disadvantage of this model is that it cannot be used to create networks with high clustering coefficient. Moreover it is not suitable for creating networks with scale free structure [17]. The scale – free term indicates that in a network the degree of each node can be very different from the others (degrees cover a big range of values).

![Graph](image)

**Figure 5**: Degree distribution in random graphs [22]

2.1.3.2 Small World Model (Watts and Strogatz model)

This model was proposed by Watts and Strogatz and according to that networks with small path length and high clustering coefficient can be created [3]. The model suggests that nodes which are arranged in a grid need to be directly connected to their neighbors to a specified layer. Moreover, random edges between non – neighboring nodes are created either by rewiring existing edges or by adding new.

![Diagram](image)

**Figure 6**: Small word network [18]

In this model the degree distribution is relevant to the layer that neighboring nodes will connect. For instance if the nodes interconnect only with their closest neighbor (layer one) we will have many nodes with degree of two while some others will have slightly higher degree. However it is important to mention that the degree
distribution also depends on the number of randomly added shortcuts (not in the case of rewiring). If no shortcuts are added the degree distribution will be zero for any value except for the value that indicates the number of directly connected nodes.

2.1.3.3 Barabasi – Albert Model

This model was proposed by Barabasi and Albert and can create scale-free networks. This characteristic is very common in real networks. The model constructs the network in time-steps. In each step new nodes are added making the network bigger. Starting from a set of unconnected nodes (at least two), new nodes connect to existing ones by preferential attachment [3]. In other words nodes with high degree are more likely to establish additional connections. The probability of a node with certain degree to have a new edge is [17]:

$$P(k_i) = \frac{k_i}{\sum k_i}$$

where $k_i$ : the degree of node $i$

The degree distribution of the network which are created with this model is a power law distribution.

![Figure 7: Scale-free network's degree distribution in a log-log plot][22]

2.1.3.4 Structured Nodes model (SN – model)

This model was proposed by Pierluigi Frisco [11]. In this model, sequence of characters, called structures, represent the nodes of the network and undirected edges link nodes if some distance measures between the structures are satisfied. The generated networks have a power law degree distribution, low average path length and the clustering coefficient is not affected by the size of the network [11]. Unlike to the Barabasi – Albert model it doesn't use the preferential attachment. The approach which is followed is inspired by the biological networks where no heuristics apply [11].
A network is created by starting from a set of initial nodes (at least one) with specified structures. The development of the network is done in logical steps which are equal to the number of nodes that the network will have. In each step a new node is created and edges with the rest of the nodes are added only if they are similar (the comparison is based on our specification). A new node is created by modifying the structure of a randomly selected existing node.

This model is implemented by a program. The description of the network's creation from the program will be helpful in understanding the model.

The program uses a property file in order to receive input from the user. This input relates with the user's preferences for the network that is going to be created. These properties which are changeable are demonstrated in the property file that is shown in the end of this section. During the description of the program's function I will refer to many of them but there are also others that won't be mentioned.

A network has nodes. In this model the nodes are represented as sequence of characters. The possible characters that a node structure may have come from the alphabet. This is a set of characters that are allowed to be present in the structure. The alphabet can be specified in the property file. For instance:

Alphabet = A,B,C,D

A possible node structure can be AACBD or ABBA. The program requires a number of initial nodes with specified structure. This can be set in the initial node property where the user defines one or more nodes separated by commas. More than one nodes can be set if the Barabasi – Albert model is implemented. An example of the initial node property:

Initial_node = ABCDABCDABCD

As already mentioned the network is built up by creating a new node by changing the structure of an existing randomly selected node. This modification can be done by mutation (changing one character with another one that exists in the alphabet), addition of a new character, deletion of a character or duplication of a character. Considering a node structure ABAA and alphabet {A,B} the effects of each of these operations can be seen below.

- mutate : ABAB, the character in the last position was changed
- add : ABAAB, a new character was added in the end
- delete : AAA , the second character was deleted
- duplicate : ABBAA, the second character has been duplicated

The modification of the nodes structure is done by using only one of the aforementioned operations. The way of deciding which operation will be used each time is based on four related to the operations probabilities. The value of each probability should be between 0 and 1 but the sum of the values of all probabilities shouldn't be over 1. The probabilities can be set to desirable values from the property file. The following example indicates that nodes' structures will be modified only by mutation (probability to mutate= 1).

Prob_to_mutate = 1
Prob_to_add = 0
Prob_to_delete = 0
Prob_to_duplicate = 0

At this point it is important to mention that the new node will be added to the network only if it's structure doesn't already exist.

By the time the new node is created an edge is added between the new and each existing nodes according to their similarity. The similarity is evaluated according to a set of input properties. The evaluation can be done either by considering the Hamming distance [12] of the character sequence or by considering the percentage of each alphabet's character in the structure of the node (Discrepancy PARIKH) or both of them [12]. In the first case the unit distance and the max distance properties are also considered. Unit distance indicates the number of characters that are going to be compared. If unit distance is equal to 1 then the characters of the nodes are compared one by one. If it is equal to 2 the characters are compared in pairs. The max distance property indicates the boundary value of adding an edge or not. For instance if the Hamming distance of the two strings (node's structures) is 2 and the max distance is 1 the edge won't be added. In the case that nodes' structures have different length instead of comparing the Hamming distance by ignoring the extra characters, the comparison can be made on the percentage of each character in the structure. The property discrepancy PARIKH is considered. It indicates the allowed deviation of each character percentage between the nodes structures [12]. If this value is not exceeded an edge is added. Finally, if no edge is added to the new node, the node is discarded.

The process of creating a new node and adding edges according to the rules is repeated until a specified number of nodes is reached [11]. If the program cannot reach the specified number of nodes, it will stop after a number of attempts set by the user. Assuming the alphabet {A,B,C}, the structure of the initial node ABCABC, and the Hamming and unit distances equal to 1, using the aforementioned variation operators (mutation, deletion, addition, duplication) the network growth is done according to the following procedure. From the initial node a new node is created by mutation and has the structure AACABC. Between the initial and the new node an edge is added as their Hamming distance is equal to 1 (only one character different). The next node will be created by randomly selecting a “parent” from the existing set of nodes. For instance a new structure may be AAACABC by adding an A at the start of the structure AACABC. In this case as the length of the new node is greater than the others, the exceeding characters would not be considered when the comparison takes place. The AAACABC node doesn't have any edges with the existing nodes as its distance (from the others) is greater than 1. The structure ABCAB can be obtained by deleting the last character of the ABCABC node with which an edge will be added (the hamming distance equals to 1). Finally a new node structure may be created by duplication. For instance the structure ABCABBC is a result of duplicating the second “B” of the ABCABC structure. Once again an edge is added between those two nodes. If the unit distance property is greater than 1 then the structures of the nodes are compared as sequence of characters and not one by one. In this case a file match may be used. This is a simple text file which indicates which sequences of characters should be considered equal. An match file may have the following content (assuming the alphabet is {A,B} and unit distance 2).

AB =
BA =
AA = BB
BB = AA

The file match indicates that above sets of characters should be considered equal. As a result the AABB and BBAA node structures will be counted as identical [11]. An instance of the property file, containing network's properties is shown below [12]:

```plaintext
Base_dir = xxx_
Format_output = TEXT_3
Counter = 1
Random_seed = 1
Network_size = INCREMENTAL
Initial_node = ABCDABCDABCD
Num_BA_initial_nodes =
Final_remove_min_nodes = 0
Running_remove_min_nodes = 0
Running_remove_max_nodes = 0
Num_new_edges_for_each_new_node = 1
Num_runs_each_network = 99
Frequency_save = 0
Type_mutation = RANDOM
Mutation_fix_number = 1
Prob_to_mutate = 1
Prob_to_add = 0
Prob_to_delete = 0
Prob_to_duplicate = 0
Alphabet = A,B,C,D
Chosen_node = RANDOM
Max_num_attempts = 1000
Type_distance = HAMMING
Direction = HAMMING
3_node_motif_details = TRUE
File_3_node_motifs = outMotifDirec.log
N_node_motifs =
Adjacency_seperator =
Node_names_file =
Discrepancy_Parikh =
File_matches =
Unit_distance = 2
Max_distance = 1
File_edges = outputEdges.log
File_motifs = outputMotifs.log
File_path_length = outputLength.log
File_clustCoeff = outputClustCoeff.log
File_network = network.log
Verbose = FALSE
```

When the network has been created the program creates several output files. These contain information about the topology of the generated network. An excerpt of the outputLength log file is given below:

- Number of nodes = 100
- Number of edges = 455
- Average degree = 4.55
- Average path length = 2
- Average clustering coefficient = 0.28892682996398167
2.2 Evolutionary Algorithms (EA's)

Within the area of evolutionary computing, evolutionary algorithms are techniques that use nature's evolution mechanism in order to provide solutions to optimization problems. As in nature where the fittest organisms survive and consequently reproduce, solutions in an evolutionary algorithm evolve through generations according to their fitness.

2.2.1 History

The birth of evolutionary computing dates back to the 1940's when Turing suggested "genetical or evolutionary search" [7]. However, it was not until the 1970's when this field started to gain the attention of the research community. Nowadays, evolutionary computing consists of four distinct fields that were developed during the last decades. These are genetic algorithms, evolutionary programming, evolution strategies and genetic programming [7]. When the concept of genetic algorithms was first introduced they were used for modeling adaptive processes [2], but currently they are widely used as an optimization technique. Evolutionary programming first appeared in the field of artificial intelligence while evolution strategies were used to solve hard optimization problems [2].

In the history of evolutionary computing many different algorithms have been developed which suggest various approaches to the evolution procedure. They mainly differ in the variation operators and the way of evaluating the solutions(fitness function) [7]. However evolutionary computation shouldn't be considered as a set of fixed algorithms that can deal with every problem but as a general methodology for dealing with optimization problems [2].

2.2.2 Components of an Evolutionary Algorithm

Every EA consists of some parts that need to be specified according to the problem that the algorithm addresses to. These parts are described below [7].

2.2.2.1 Representation

It relates to the description of candidate solutions (individuals) of the problem in such way that evolution can take place. Solutions that are described in the original problem setting are called phenotypes while those which are described in the algorithm's environment are called genotypes. The procedure of linking phenotypes to genotypes is named representation and it should be invertible. For instance, we define the problem of visiting a number of cities with the constraint of each city to be visited only once. The phenotype is a sequence of cities names which indicates a solution. In contrast the genotype might be a sequence of integers where each number would represent a city.
2.2.2.2 Fitness function

It is a way of evaluating the solutions of the problem. The fitness function, also called evaluation function, draws the rules which the candidate solutions must comply with. The more these rules are satisfied, the fitter the solution is. For example, if the fitness function targeted to maximize the square value of an integer, the fitness of the number 5 would be 25 while the fitness of the number 10 would be 100. In this case the second individual is fitter than the first. It is worth mentioning that the fitness function may suggest either the maximization or the minimization of a score.

2.2.2.3 Population

The population is a set of individuals (genotypes). The term population size indicates the number of individuals that the population has. In most applications the size remains static during the evolution procedure. The term population diversity refers to the unique solutions that exist in a population. Currently there isn't a certain way of measuring that but it is common to count the different fitness values within the population.

2.2.2.4 Parent Selection Mechanism

It refers to the manner of selecting individuals from the existing population which will be subjected to variation and produce offspring. The term offspring (children) means the creation of new individuals that inherit most characteristics from the selected parents and only differ to those that the variation operators have changed. Usually the selection mechanism is based on probabilities, with the fitter individuals having higher probability to be selected and the less fit lower.

2.2.2.5 Variation Operators

Variation operators are methods for creating new individuals. They are divided into two categories according to the number of parameters they receive as input.

**Mutation**

This operator has single input (one individual) and creates a slightly different offspring. The modification that this operation causes depends on the design of the algorithm. However it is common to randomly swap two values of the input genotype.

**Crossover**

This operator receives two individuals as input and creates an offspring by combining parts of each parent. The result of the operation is a new individual which has inherited characteristics from both parents. The selection of the parts to be combined is usually made at random. Crossover can also be done with more than two parents but the fact that this doesn't happen to nature had prevented the widespread use of it.
2.2.2.6 Survivor Selection Mechanism

By the time the offspring have been created, in order to keep the population size same, some individuals have to be replaced by the new ones. This procedure is called survivor selection or replacement. The pattern which is used is usually based on individual's fitness. The worse solutions are replaced by the better. However there are cases where other factors may influence this process. For instance the replacement mechanism may also consider the age of an individual. As a result the newly created individuals will always override the older.

![Evolutionary Algorithm Flow Chart](image)

**Figure 8**: Evolutionary algorithm – flow chart [7]

2.3 Multi – Objective optimization

2.3.1 Basic Concepts

The solution to some problems requires the optimization of a single parameter. For instance a financial problem may just require the maximization of a company's profit. These problems are termed as single – objective optimization problems. However many real world problems require the concurrent optimization of many different parameters which in some cases are also conflicting [16]. For instance someone would like to maximize the comfort of the car he intents to buy, while in the same time he wants to minimize the cost of this car [6]. The best combination of these two objectives would be a solution to this problem. These problems are termed as multi – objective optimization problems.
The solutions to a multi-objective optimization problem can be divided into two categories. These are the dominated and the non-dominated solutions. The second category is also termed as Pareto-optimal [16]. The dominated solutions, as implied by their name, are those for which better solutions, in respect to all the objectives, exist.

Figure 9: Optimization of two conflicting parameters [6]

Figure 10: Graphical representation of the Pareto-optimal surface for a two objective problem [23]
2.3.2 Evolutionary Algorithms and multi – objective optimization

Evolutionary algorithms have been considered suitable for solving multi – objective optimization problems from their early stages of development [9]. However the creation of an evolutionary algorithm for that purpose hadn't taken place till the mid 80's [21][20]. Currently there are three basic approaches in solving multi – objective optimization problems with evolutionary algorithms. They mainly differ in the selection mechanism that is used and are [4]:

- Aggregating functions approach
- Population – based approach
- Pareto – based approach

Each approach will be shortly described in the following pages.

2.3.2.1 Aggregating functions approach

This a simple approach in solving multi – objective optimization problems. The technique that is used in this approach suggests the use of one single function, linear or not, that incorporates all the problem's objectives. The function assigns a fitness score to each individual in the population taking into consideration the performance of the individual in all objectives. It is important to mention that each objective may also have a different weight in the function. However in most cases all objectives have equal weights. The evaluated individual has a score which is the sum of the single scores for each objective [4].

2.3.2.2 Population – based approach

In this approach the population (set of solutions) is divided to sub – populations which optimize each one objective separately. Consequently the population is divided into \( k \) sub – populations where \( k \) indicates the number of objectives [4]. A widely known implementation of the population based approach is the Vector Evaluated Genetic Algorithm (VEGA) created by Schaffer [20], [4]. In this algorithm the population was separated into parts of one – objective optimization and then all parts were recombined in order to proceed with the variation operation [4]. However, VEGA had many drawbacks due to the fact that it didn't consider the Pareto – optimal set. For instance if a solution is good in optimizing all the objectives together but it can't be considered optimal for at least one of them separately, it will not be accepted [4]. Alterations to this algorithm have been made in order to overcome these issues and generally it is still in use.

2.3.2.3 Pareto – based approach

In this approach the mechanism of selecting individuals from the population and evaluating them is using the Pareto – optimality concept. The non – dominated solutions have greater probability of reproduction than the dominated ones [9]. Among them the probability is the same. When non – dominated solutions have been evaluated they are removed from the population. Many algorithms have been
developed according to this approach and the fitness assignment as well as the individuals that are kept inside the population vary greatly. Some of the algorithms that follow this approach are mentioned below [4]:

- Multi – Objective Genetic Algorithm (MOGA)
- Non – dominated Sorting Genetic Algorithm (NSGA)
- Niched Pareto Genetic Algorithm (NPGA)
- Strength Pareto Evolutionary Algorithm (SPEA)
- Pareto Archived Evolution Strategy (PAES)

**Multi – Objective Genetic Algorithm**

This algorithm was proposed by Fonseca and Fleming and according to it the non – dominated solutions share common ranking and fitness value. Consequently these solutions have equal probability for reproduction [4], [10].

**Non – dominated Sorting Genetic Algorithm (NSGA)**

This technique was proposed by Srinivas and Deb and targets to the classification of the individuals into non – dominated solutions sets [24]. In other words the population is first evaluated, the non – dominated solutions are assigned a rank value and then are ignored. The process iterates until the all the population is divided into sets of non – dominated solutions with different ranks [4].
Figure 11: NSGA flow chart [24]
Pareto Archived Evolution Strategy (PAES)

This method was developed by Knowles and Corne. According to it, each parent produces a slightly modified child (as a result of mutation). The algorithm uses an archive where the non-dominated solutions are stored and are used as reference point in order to evaluate the next generations [4], [15].

Figure 12: The Pareto Archived Evolution Strategy [15]
Chapter 3

Program Description

3.1 Overview

The implemented evolutionary algorithm aims at providing optimal solutions for a set of three network properties which are provided by the user. These are the network's average degree, average path length and average clustering coefficient. The program makes use of the SN – model algorithm in order to create networks. The evolution program has a property file in which the user can define various parameters, relevant to its execution. The property file is fully described in section 3.3. At this point it is important to mention that the SN – model algorithm also has a property file which contains properties of the SN-model simulation. These two property files are completely different and in order to avoid confusion the SN – model algorithm's property file will be referred as network property file. The program's functionality can be summarized in the following four operations:

- Chromosomes creation (by running SN – model algorithm with a different network property file for each chromosome)
- Chromosomes evaluation
- Chromosome's variation operations
- Population's evolution

The logical order of these operations can be seen in the following flow chart. An analytical description of each function is given in section 3.

![Program's overview flow chart](image-url)

**Figure 13**: Program's overview flow chart
3.2 Evolution Program's Components

This section describes how the components of evolutionary algorithms have been used in the implemented program.

3.2.1 Representation

The genotypes of the evolution program are network property files. Each file, given as input to the SN – model algorithm creates a network with specific characteristics. The network property file (described in section 2.1.3.4) contains network properties on some of which the variation operations are made.

3.2.2 Fitness Function

3.2.2.1 The implemented Fitness Function

Each network property file (chromosome) is provided as input to the SN – model algorithm. If the network property file indicates more than one random seeds, then the SN – model algorithm creates one network for each seed. For the evolution program, networks are run for five random seeds (1 to 5). Among other output files SN – model algorithm creates a file (outputLength.log.avg) in which the network's average degree, average path length and average clustering coefficient are available. These values are used in order to find the fitness of the chromosome. In other words, for each random seed and for each of the three values that the program tries to match, the distance from its target (as specified by the user) is calculated. Then the average (from all random seeds) distance of each value is added to a vector which constitutes the distances of the network from the the target values. For example, assuming that from a network property file the created network has the following values :

<table>
<thead>
<tr>
<th>Random Seed</th>
<th>Average Degree</th>
<th>Average Path Length</th>
<th>Average Clustering Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4,2</td>
<td>3</td>
<td>0,2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>0,22</td>
</tr>
<tr>
<td>3</td>
<td>3,6</td>
<td>3</td>
<td>0,25</td>
</tr>
<tr>
<td>4</td>
<td>4,1</td>
<td>3</td>
<td>0,18</td>
</tr>
<tr>
<td>5</td>
<td>4,3</td>
<td>4</td>
<td>0,19</td>
</tr>
</tbody>
</table>

The target values as defined by the user :

<table>
<thead>
<tr>
<th>Average Degree</th>
<th>Average Path Length</th>
<th>Average Clustering Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>0,2</td>
</tr>
</tbody>
</table>
Then the absolute distance for each seed and each value would be:

<table>
<thead>
<tr>
<th>Random Seed</th>
<th>Average Degree Distance</th>
<th>Average Path Length Distance</th>
<th>Average Clustering Coefficient Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0.02</td>
</tr>
<tr>
<td>3</td>
<td>0.4</td>
<td>0</td>
<td>0.05</td>
</tr>
<tr>
<td>4</td>
<td>0.1</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>5</td>
<td>0.3</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td>Average</td>
<td>0.2</td>
<td>0.4</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Finally the chromosome will have a vector containing the three average distances \{0.2,0.4,0.02\}. Needless to say, the lower these distance values are, the better the chromosome is.

In order to find how fit a chromosome is, a Pareto – based approach has been used. In other words, the chromosome's "distances" vector is compared with all other chromosomes' vectors, to test whether it is Pareto – dominated. In the end of all comparisons each chromosome has a score which indicates the number of chromosomes that have dominated it. The best chromosomes will have a score of '0' (non – dominated). A chromosome may Pareto – dominate another if it is not worst in any of the three values and it is better in at least one value. In this case "better" implies to have a lower value and "worst" to have a higher value. This is because the vector contains distances from the target values and the goal is to minimize these distances. A Pareto – dominance example is provided below:

Assuming three chromosomes A,B,C with vectors:

A : 0.2 , 0.4 , 0.1  
B : 0.2 , 0.3 , 0.1  
C : 0.3 , 0.4 , 0.2

Chromosome B dominates A and C  
Chromosome A dominates C but not B  
Chromosome C doesn't dominate neither A nor B

The score of each chromosome would be:

B : 0 (non – dominated)  
A: 1 (dominated by B)  
C: 2 (dominated by A and B)

3.2.2.2 Another approach to the Fitness Function

As already described in the previous section the chromosome's distance vector contains the average distance values from the five random seeds. This simplified approach provided acceptable results, which are analyzed in the next chapter. However there were other approaches that could have been implemented. For
instance, instead of calculating the average distance values, the worst distances could have been used. Another approach might be that in order to decide if a chromosome A is Pareto – dominated by a chromosome B we would have to check if the distance values of all random seeds of a chromosome A, are Pareto – dominated by all distance values of all random seeds of a chromosome B. Unfortunately this approach increases the complexity of the fitness score application as from the one comparison (that it is made in the currently implemented version) the algorithm would have to make twenty five comparisons. The implementation of a different fitness function wouldn't be an extremely difficult task. However every new or additional fitness function needs to be tested. The duration of the program's test – runs and the generally limited time availability made the implementation of the aforementioned approaches impossible.

3.2.2.3 A rejected Fitness Function

Apart from the implemented and the suggested fitness functions the algorithm first used a simple aggregate function (scalar). According to this approach the fitness of a chromosome was a score. The program first calculated the average degree, the average path length and the average clustering coefficient of a network (not their distances). Then the score of each chromosome was the square root of the sum of the squared differences between the average and the target values (Euclidean distance). After conducting test – runs with this fitness function, it was proved that despite the fact that many chromosomes had very good fitness scores, the resulting networks didn't have the desirable characteristics. The problem was located in the calculation of the chromosomes' fitness. The fact that the average value of each property was calculated lead to the evolution of solutions with very different values in each random seed. For instance, assuming that we try to match only the degree of a network we set target value 5. Then we calculate the average value of a chromosome's degree. If in the 5 random seeds the chromosome has values 8,2,7,2,6 then the average equals to 5, which means that the solution is optimal. Needless to say this "optimal" solution is actually a very bad solution. The fluctuations of a network's average degree between different seeds is not very common but it may be possible if the probability to duplicate (network's property file) has a high value (above 0.2). This was the reason for the creation of networks with such different values among the five random seeds. In most cases the program evolved solutions with high probability to duplicate and the resulting chromosomes weren't as fit as they were considered. A solution to this problem is to first calculate the distances from the target values for each seed and then find the average distance of each property. This slight change was implemented for the Pareto – based approach.

3.2.3 Population

The population is a set of chromosomes (network properties files) whose size is defined by the user. In order to sustain population's diversity every chromosome that is added is unique. The check of a chromosome’s uniqueness is made on the network's property file (see section 2.1.3.4). As the network's properties that are used for the variation operations are the Probability_to_mutate, the Probability_to_delete, the Probability_to_add, the Probability_to_duplicate and the Initial_node, two chromosomes are equal if all the aforementioned properties are
the same. Equal chromosomes are not allowed in the population. As a result each chromosome will differ in at least one property from all the others.

3.2.4 Parent Selection Mechanism

This mechanism specifies the way the parents to be used for reproduction are selected. By the time the chromosomes have been given a fitness score (which implies the number of chromosomes that Pareto – dominate them), the best of them share equal probability to be selected as parents. Needless to say the best chromosomes are the ones with '0' score (non dominated). However it should be mentioned that in case there is only one non dominated solution, the program will also consider as best chromosomes, the ones with score '1'. This happens because the set of potential parents should contain at least two chromosomes. The reason for this is that the crossover operation needs two parents.

3.2.5 Variation Operators

In this evolutionary algorithm two variation operators are used. These are mutation and crossover. They way they are used in the program is explained is explained in the next two sections.

3.2.5.1 Mutation

When a chromosome is to be mutated, two different options exist. Mutation may be conducted in the probabilities of the network (Probability_to_mutate, the Probability_to_delete, the Probability_to_add, the Probability_to_duplicate) or in the initial node structure.
Mutating the probabilities of the network in the network's property file means changing the value of one(out of four) randomly selected probability. The change in the value follows a Gaussian distribution.
Figure 14: Gaussian distribution ($\mu$: mean value, $\sigma$: standard deviation) [8]

However, it is important to mention that due to the fact that the sum of all probabilities should be 1, a change in the value of one probability implies that changes to the values of the other probabilities need to be done. As a result, when a probability changes, the difference from its previous value (either positive or negative) is equally distributed to the other three probabilities. In this way, when mutating one probability, a change to the value of the selected for mutation probability occurs, but additionally, the remaining three probabilities also change.

Mutating the Initial_node property includes two main options, changing node's structure length and changing node's structure character sequence. Changing node's structure length includes two sub-options: increase the length of the structure or decrease the length of the structure. The increment of node's structure is done by randomly selecting one character from the alphabet (property in network's properties file) and adding it in the end of the structure. In contrast, decrement of node's structure is achieved by deleting a randomly selected character from the existing structure. The second main option of the initial node mutation is achieved by swapping positions of two randomly selected characters from the node's structure. However, as the structure of a node is a sequence of a certain set of letters, characters are repeated within the structure. When swapping two randomly selected characters, it is possible to result in the same structure. The algorithm handles this possibility and changes the sequence characters until a structure different from the previous is obtained.
3.2.5.2 Crossover

The crossover operation needs two chromosomes to be used as parents. When the parents have been selected their offspring (the new chromosome) inherits the initial node's structure from the one parent and the values of Probability_to_mutate, Probability_to_delete, Probability_to_add, Probability_to_duplicate from the second. For instance, if the two chromosomes that are selected for reproduction have the following properties:

Parent A

Initial_node = ABCDABCDABCD
Prob_to_mutate = 1
Prob_to_add = 0
Prob_to_delete = 0
Prob_to_duplicate = 0

Parent B

Initial_node = ABCD
Prob_to_mutate = 0.3
Prob_to_add = 0.3
Prob_to_delete = 0.4
Prob_to_duplicate = 0

Then a child could be:

Child

Initial_node = ABCDABCDABCD
Prob_to_mutate = 0.3
Prob_to_add = 0.3
Prob_to_delete = 0.4
Prob_to_duplicate = 0

Once again it is possible for the algorithm to be unable to create unique chromosomes as the two parents may share the same node structure and different probabilities or vice versa. The created child would be identical to one of its parents. For that reason if the created child is not unique (not only compared to its parents but to the whole population) the algorithm selects different parents and repeats the crossover operation. However it was proven that the algorithm may be completely unable to create unique children by conducting the crossover operation for a certain set of potential parents. A simple scenario that confirms this problematic situation (infinite loop) is when all candidate parents have the same initial node structure. In order to overcome this problem the algorithm has a limit in trying to create a unique offspring with the crossover operation. This limit is a property (Max_Num_Crossover_Attempts) in the program's property file.
3.2.6 Survivor Selection Mechanism

Once each chromosome in the population has been rated according to how many other chromosomes Pareto-dominated it, the best of them (non-dominated) are archived. Only if just one non-dominated solution exists, chromosomes which have been Pareto-dominated once are also added to the archive. The chromosomes that haven't been added to the archive are all replaced by the new ones.
Figure 15: Program's analytical flow chart
3.3 Property File Definition

The property file is read at the startup of the program and it contains properties in a "key-value" format. All properties are checked for their values and if errors exist the program terminates informing the user properly. The file is a simple text document and it is named “EvolutionProgramProperties.txt”. A sample property file can be seen below:

Random_Seed=5
Probability_To_Mutate=0.8
Probability_To_Crossover=0.2
Probability_to_Mutate_Probabilities=0.8
Probability_to_Mutate_Initial_Node=0.2
Population_Size=40
Target_Degree=6.04
Target_Path_Length=3.78
Target_Clust_Coeff=0.22
Number_of_Generations=40
Individuals_for_Results=5
Probability_Deviation=1.0
Number_of_Nodes=230
Min_Node_Structure_Length=2
Max_Node_Structure_Length=80
Max_Num_Crossover_Attempts=10
Cluster=true
Computer_Names=linux69,linux70,linux71,linux73,linux74
Cluster_Directory=/u1/msc/aox1/Desktop/
Username=aox1

The meaning of each property and their permissible content is described below:

**Random_Seed**

It is an integer, greater than 0 number which specifies the number of random seeds. It refers to the network property file where the random seed property specifies the number of different seeds that the simulation of a network will perform.

**Probability_To_Mutate**

It is a double value between 0 and 1.0 both included. It refers to the variation operation of mutation. It specifies the probability of a chromosome to be mutated (rather than conducting the crossover operation). The Probability_To_Mutate and Probability_To_Crossover (see next) properties should sum up to 1.

**Probability_To_Crossover**

It is again a double value between 0 and 1.0 both included. It refers to the variation operation of crossover. It specifies the probability of conducting the crossover operation. As already mentioned Probability_To_Mutate and Probability_To_Crossover should sum up to 1.
Probability_to_Mutate_Probabilities

A double value between 0 and 1 both included. It is a probability that is used only if the variation operation of mutation is to be conducted. As described in the previous section the mutation operation is divided in two sub-operations, mutating probabilities and mutating initial node's structure. The value in this property specifies the probability of mutating probabilities (rather than mutating initial node). Once again the values provided in Probability_to_Mutate_Probabilities and Probability_to_Mutate_Initial_Node properties should sum up to 1.

Probability_to_Mutate_Initial_Node

A double value between 0 and 1 both included. Again this probability will be used only if the variation operation of mutation is to be conducted. The value in this property specifies the probability of mutating the structure of the initial node and it should sum up to 1 with the Probability_to_Mutate_Probabilities.

Population_Size

An integer number greater than 2 which specifies the size of the population. In other words this number indicates how many chromosomes will exist in the population in each generation.

Target_Degree

In this property the user is able to define the desirable degree of the network. It is a double value and it cannot be negative. This value constitutes the one of the three objectives that the algorithm tries to optimize.

Target_Path_Length

At this point the user can define the value for the target path length of the network. The permitted content is a double non-negative value.

Target_Clust_Coeff

This property is the third target network characteristic that the user is able to specify and refers to the desirable clustering coefficient of the network. The value of this property is again a double non-negative number.

Number_of_Generations

An integer number which specifies the number of generations that the population will be evolved. This number should be equal or greater than 1.

Individuals_for_Results

The value of this property is an integer positive and non-zero number which indicates the number of individuals (chromosomes) that would be provided as results. However the algorithm will only return non-dominated solutions and so if the individuals for results property has a value greater than the resulting non-dominated solutions the algorithm will return less chromosomes than those
specified in this property.

**Probability_Deviation**

As mentioned in the previous section when mutating a probability, the algorithm gives a new value by following a Gaussian distribution centered in the probability's initial value. If the user sets this property equal to 1 then the change in the probability's value will follow a normal Gaussian distribution. In contrast if the value of the property is less than 1, the probability of making a great change in the value of the probability decreases. In other words the lower the value of Probability_Deviation is, the narrower the Gaussian curve becomes. Needless to say the value of this property is a double number between 0 and 1.

**Number_of_Nodes**

A positive non zero integer number which specifies the number of nodes that the networks will have. It refers to the network property file.

**Min_Node_Structure_Length**

As the mutation of initial node's structure includes the option of decreasing the its length, there was need to prevent the decrement of the structure if its length was below 3. This is because there are other operations, such as changing the sequence of the characters, that require at least two letters in the structure. Instead of preventing the decrement only for structures with length less than two, this property was added in which the user can set the minimum length of the node's structure. If this limit is reached, then instead of decreasing the structure the algorithm increases it by one. The value of this property is an integer number greater than 1.

**Max_Node_Structure_Length**

In the same way as previously this property prevents the increase of a node's structure length if the specified by the user limit is exceeded. In case the limit is exceeded the algorithm proceeds with node's structure decrement. However there wasn't a technical reason for adding this limit. It was included in the property file just for providing more options to the user.

**Max_Num_Crossover_Attempts**

As explained in the previous section, the crossover operation may be impossible to be conducted. This relates to the unique chromosomes restriction that has been applied in the population. This property defines the maximum number of attempts that the algorithm will perform in order to conduct the crossover operation and achieve a unique chromosome. If this limit is exceeded then the algorithm performs the mutation operation. The value of this property is an integer number greater than one.

**Cluster**

This property specifies whether the program should operate in single or cluster mode. Acceptable values are true or false. However if the cluster property is set to true the user has also to define the computer names (next property) for the cluster.
**Computer Names**

If the cluster property is true the user should specify the names of the computers that will be used in the computer cluster. The names should be provided in a comma separated list. It is important to provide valid and reachable (connected in a network) computer names otherwise the program will terminate.

**Cluster Directory**

This property specifies the path of the directory where the instances of the SN – model algorithm and the network property file will be stored. Each computer participating in the cluster will have its own directory in the specified path. For example, if the path is `/u1/msc/aox1/Desktop/` and the computer names are `linux42`, `linux41`, `linux40` then the algorithm will build the following folders: `/u1/msc/aox1/Desktop/linux42`, `/u1/msc/aox1/Desktop/linux41`, `/u1/msc/aox1/Desktop/linux 40`. It is important to provide a valid path otherwise the program will exit.

**Username**

This property refers to the username that will be used to in order connect via ssh to remote computers when the cluster mode is on. If the Cluster property is true then the user has to provide a string value to this property.

### 3.4 Program's Modes

The first and main program's mode is the “single mode” which means that the algorithm is running on one computer and is using one thread. However the duration of the test – runs made the implementation of different modes absolutely necessary. For instance a test -run for a network of 230 nodes, with forty chromosomes in each population and for forty generations, lasted almost fifteen hours. However this run was made on an ordinary computer(2.2 GHz). The duration of evolution program is strongly affected by the duration of SN – model algorithm. For that reason the different modes that have been implemented focus on distributing the load of SN – model invocations. There are two other modes, the “multiple – cores” mode and the “cluster” mode. Both are described analytically in the following sections.

#### 3.4.1 Multiple – cores mode

For this mode the algorithm was modified in order to benefit from multiple cores computers. The main idea of this mode is to simultaneously run instances of SN – model algorithm in more than one cores. For instance if a computer has two cores, for the creation of chromosomes the program invokes SN – model algorithm twice. This is achieved by running a shell script which contains commands for executing in parallel SN – model algorithm, one for each core. However, in order to run SN – model algorithm simultaneously, it is required to have a different directory for each core. Each directory includes a copy of SN – model algorithm and a network property file. Moreover inside each directory another shell script exists (bash file) which contains the the command for executing SN – model algorithm in the specific directory. The whole process
starts from writing a chromosome's network's properties file in one's core directory. So, assuming we have two cores, the algorithm writes the network's properties files of two different chromosomes in each core's directory. Then the shell script containing commands to run shell scripts in both directories is executed. By the time the execution of SN–model algorithm has finished, the evolution program collects the required data from all directories (in the example of two cores, there would be two directories). Then the output of SN–model algorithm is deleted and the network properties files of the next chromosomes are written as before in order to repeat the procedure.

The way this mode is implemented enables running SN–model algorithm in parallel in more than two cores. However the implementation was only tested in computers with two cores (as there wasn't access to computers with more than two cores). The performance of the algorithm was slightly better than running it on one core. This may happened because many processes run on a computer and it is not possible to exploit all the computational power for one only task.

With such poor performance the multiple cores mode was not integrated in the final version of the program (where the cluster mode option is available). The multiple cores mode may be useful in computers with more than two cores but the fact that it is not tested in such computers lead to the decision of not integrating it to the evolution program.

### 3.4.2 Cluster mode

The main idea of this mode is to run SN–model algorithm in more than one computers. In order to achieve this the existence of a computer network is required. Moreover as the master computer connects to the other computers via ssh (secure shell) a password–less ssh connection has to be established.

On startup of the application if the property “Cluster” in the property file is set “true” the algorithm reads the provided computer names that are going to participate in the computer cluster and builds one directory (in a specified by the user location) for each computer. The directory has the name of the remote host and contains a copy of SN–model algorithm (which is copied by the project's directory). Each directory also contains a shell script which has a command that if run, establishes an ssh connection from the master computer to the remote host. For instance if a host is named linux41, inside the related directory there would be a file (Ssh.bash) containing the following command:

```bash
ssh linux41 -l username ; java directory_path/modelNetwork.Main ,
```

which establishes an ssh connection to the remote host linux41 and invokes the main method of the SN–model algorithm.

Once directories for all host have been built, the algorithm proceeds normally until the creation of chromosomes is to be done. At this point the network properties files of the chromosomes that need to be run by SN–model algorithm are written to the directories of the hosts. Then the algorithm executes a shell script which in turn executes all the shells scripts contained in hosts' directories (Ssh.bash). This results in almost concurrent invocation of SN–model algorithm in all used remote hosts. The evolution program waits until hosts have finished the execution in the order that they have been called. Once this happens the data (values of properties that need to be matched) of the chromosomes is collected and the output of SN–model algorithm is deleted in all directories. It is important to mention that there is no benefit from using more
computers than the size of the population. This is because the algorithm will distribute the network’s properties files of chromosomes in the current population. The program will wait until one generation has finished and will not proceed into creating other chromosomes. So if for example the population’s size is twenty chromosomes and we use twenty five computers the program will invoke twenty instances of SN – model algorithm in twenty computers and the rest five will remain unused. Once the twenty first finish, the chromosomes will be evaluated, the new generation will be created from the best chromosomes and then the program will run SN – model algorithm for the new generation (leaving again five computers unused). The same scenario may hold even if we use less computers than the size of the population. Assuming we use ten computers and the size of the population is fifteen. The program will firstly use ten computers but for the remaining five chromosomes of the generation, the program will wait until the first ten computers have finished and in the “second round” it will only use five computers. The reason that this happens is that it is required to evaluate one generation in order to proceed to the next.

In general the performance of the algorithm in the cluster mode is by far better than in the single mode. The more the computers in use the faster the algorithm runs. However, the speed of proceeding from one generation to another is dependent from the slower computer in the cluster. For example if one computer needs more time to run SN – model algorithm than the others, the algorithm will wait until the slowest has also finished. Finally it is important to mention that only one instance of SN – model algorithm is invoked in each computer. In other words, the execution of the program is using only one core of each computer.

3.5 Problems

During the implementation of the evolution program many problems were encountered. Most of them have been solved but there are some others that haven’t. For both cases a short description is provided in the two following sections.

Solved Problems

It is true that many problems have been encountered. Among other problems that were handled efficiently there were two that are worth mentioning. The first one had to do with the precision of numbers with lots of decimal places. These numbers were created when the algorithm changed the value of one probability (with the Java’s “nextGaussian” method) and then tried to distribute the difference of the probability from its initial values, to the rest three probabilities. This was done in order to maintain the sum of the four probabilities equal to one. Unfortunately the division of the probability's difference by three (in order to distribute it equally) was creating numbers with many decimal places. Moreover the difference itself had in most cases many decimal places (due to the use of nextCaussian method). The “less than required” precision of these numbers was obvious when summing the four probabilities. The sum wasn’t equal to one and so the SN – model algorithm wasn’t able to run (as the sum of the probabilities is required to be equal to one). This problem was solved by manipulating numbers as Strings. The implementation of this feature (LongNum class) was made by Pierluigi Frisco.
The second problem had to do with the output of SN – model algorithm. In more details, the algorithm produces various files, one of which is used in order to read network's values (degree, path length and clustering coefficient). The name of the file contains the number of nodes of the created network, which is a constant value, set on program's startup. When the network's values are to be read, the evolution program waits until a certain file (with number of nodes in file's name) is available. However, there are cases that SN – model algorithm creates networks with less than specified nodes. This is an expected behavior of the algorithm as there is a possibility of being unable to create the network with the specified nodes. Unfortunately this results in creating output files with different than expected name and consequently put the evolution program in an infinite loop (keeps waiting for a file which will never be created). This problem was solved by searching for all possible output files and if a file with different than expected name is found (a network with less nodes), the chromosome that is represented by the file is discarded. In this way the evolution program doesn't go into an infinite loop.

Unresolved Problems

The first unresolved problem is that the algorithm is not platform independent. This is because the program uses shell scripts which are not platform independent. For example, “bash” files cannot be executed on Windows OS.

The second unresolved problem relates to the “cluster” mode of the program. It was observed that in some cases where the algorithm waits until the output of the SN – model algorithm is present in each one of the computer directories, it may be unable to find the required files despite the fact that the files actually exist. If the user opens the directory where the algorithm is waiting for the output of SN – model algorithm, then the evolution program “realizes” that the files exist and proceeds to the collection of data. This problem may just cause delay but may also make the algorithm wait forever (if nobody opens the “problematic” directory). This error didn't appear frequently. An explanation of this strange behavior may be found in the fact that the error only appeared when the user who run the evolution program was simultaneously logged in all computers that were participating in the cluster. The simultaneous log in was required because the computers in the lab where the test runs where conducted, automatically shut down (at night) if no user is logged in and no process is running. Unfortunately when a computer finishes the run of the SN – model algorithm, it is waits (idle) for the next invocation of the algorithm. It may also remain unused for some time (as described in “Cluster mode” section), increasing this way the probability of get shut down. All these led to logging in all computers that were used in the test runs. Currently there is no solution to be proposed for this problem.
Chapter 4

Results Analysis

In order to evaluate the output of the evolution program many test runs have been conducted. These were done for a certain network, the Escherichia coli protein-protein interaction network. This network was reproduced using the SN – model algorithm (with different sets of input parameters) in [11]. The E. coli protein-protein interaction network has 230 nodes, an average degree of 6.04, an average path length of 3.78 and an average clustering coefficient of 0.22. Considering these properties as target values the evolution program was run for various sets of input parameters. However it is important to mention that as the output of evolutionary algorithms is stochastic in the following analysis of the results the evolution properties are evaluated according to the general trend of each set. It is true that in some cases sets of evolution properties provided unexpectedly good results and vice versa. All test – runs have been conducted on a computer cluster, using different number of computers according to the size of the population. The two factors that affect the duration of the program's execution are the number of generations and the population's size.

<table>
<thead>
<tr>
<th>Size of Population</th>
<th>Generations</th>
<th>Used Computers</th>
<th>Average Duration (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>20</td>
<td>20</td>
<td>27</td>
</tr>
<tr>
<td>20</td>
<td>30</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>20</td>
<td>40</td>
<td>20</td>
<td>52</td>
</tr>
<tr>
<td>30</td>
<td>20</td>
<td>30</td>
<td>28</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>30</td>
<td>38</td>
</tr>
<tr>
<td>30</td>
<td>40</td>
<td>30</td>
<td>53</td>
</tr>
<tr>
<td>40</td>
<td>20</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>40</td>
<td>30</td>
<td>20</td>
<td>69</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
<td>20</td>
<td>102</td>
</tr>
</tbody>
</table>

The computers that were used were running Linux OS and had 2 cores of 2.4GHz. The use of only twenty computers for populations of size equal to forty increased the duration of these test – runs. However the use of forty computers (instead of twenty) wasn't possible.
4.1 Results without using a match file

Without the use of a match file (see section 2.1.3.4), thirty six test runs have been conducted. The evolution properties that were differentiated are the number of generations, the size of the population, and the probabilities to mutate and crossover. The probability to mutate probabilities and the probability to mutate the node's structure were set to 0.8 and 0.2 respectively. For all tests a common network property file had been used and it is provided below:

Counter=1
File_clustCoeff=outputClustCoeff.log
Base_dir=xxx_
Num_BA_initial_nodes=
File_dot_network=
File_edges_incoming=
Running_remove_max_nodes=0
File_edges=outputEdges.log
Network_size=INCREMENTAL
Discrepancy_Parikh=
File_motifs=outputMotifs.log
Node_names_file=
3_node_motif_details=TRUE
Initial_node=ABCDABCDABCD
Make_undirected=
Num_new_edges_for_each_new_node=1
Format_output=TEXT_3
File_matches=
File_path_length=outputLength.log
Parallel=
N_node_motifs=
Direction=HAMMING
Prob_to_delete=0.0
Prob_to_mutate=1.0
Prob_to_add=0.0
Prob_to_duplicate=0.0
Max_distance=1
Type_mutation=RANDOM
Unit_distance=2
Random_seed=1,2,3,4,5
Max_num_attempts=1000
Frequency_save=0
Chosen_node=RANDOM
File_network=network.log
Type_distance=HAMMING
Num_runs_each_network=229
Running_remove_min_nodes=0
File_3_node_motifs=outMotifDirec.log
Verbose=FALSE
Mutation_fix_number=1
Final_remove_min_nodes=0
Alphabet=A,B,C,D
The following results refer to the best non dominated solution of each test run. As best non dominated solution is considered the one that has the lowest sum of the three distances from the target values.

<table>
<thead>
<tr>
<th>Population Size</th>
<th>Probability to Mutate</th>
<th>Probability to Crossover</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.8</td>
<td>0.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generations</th>
<th>Degree Distance</th>
<th>Path length Distance</th>
<th>Clustering Coefficient Distance</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.482608695652174</td>
<td>0.77999999999999998</td>
<td>0.03531571186518015</td>
<td>1.297924407517354</td>
</tr>
<tr>
<td>30</td>
<td>0.350086956521739</td>
<td>0.868</td>
<td>0.05468935445132674</td>
<td>1.272776310973066</td>
</tr>
<tr>
<td>40</td>
<td>0.345217391304348</td>
<td>0.6679999999999999</td>
<td>0.05363984043912362</td>
<td>1.066857231743472</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generations</th>
<th>Degree Distance</th>
<th>Path length Distance</th>
<th>Clustering Coefficient Distance</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.255826086956522</td>
<td>0.868</td>
<td>0.06500725023496398</td>
<td>1.18883337191486</td>
</tr>
<tr>
<td>30</td>
<td>0.541391304347826</td>
<td>0.77999999999999998</td>
<td>0.03562136144986206</td>
<td>1.357012665797688</td>
</tr>
<tr>
<td>40</td>
<td>0.454434782608696</td>
<td>0.5559999999999999</td>
<td>0.0311601285546703</td>
<td>1.041594911163366</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generations</th>
<th>Degree Distance</th>
<th>Path length Distance</th>
<th>Clustering Coefficient Distance</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.598782608695652</td>
<td>0.6679999999999999</td>
<td>0.05107177266683978</td>
<td>1.31785438136405</td>
</tr>
<tr>
<td>30</td>
<td>0.638086956521739</td>
<td>0.444</td>
<td>0.04162352621929014</td>
<td>1.123710482741029</td>
</tr>
<tr>
<td>40</td>
<td>0.540347826086956</td>
<td>0.444</td>
<td>0.05781903840656453</td>
<td>1.042166864493521</td>
</tr>
<tr>
<td>Generations</td>
<td>Degree Distance</td>
<td>Path length Distance</td>
<td>Clustering Coefficient Distance</td>
<td>Score</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------</td>
<td>----------------------</td>
<td>---------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>20</td>
<td>0.519826086956522</td>
<td>0.6679999999999999</td>
<td>0.04683046758755149</td>
<td>1.23465655454073</td>
</tr>
<tr>
<td>30</td>
<td>0.42295652173913</td>
<td>0.756</td>
<td>0.04727040829280253</td>
<td>1.226226930031933</td>
</tr>
<tr>
<td>40</td>
<td>0.341913043478261</td>
<td>0.6679999999999999</td>
<td>0.05432632535898803</td>
<td>1.064239368837249</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generations</th>
<th>Degree Distance</th>
<th>Path length Distance</th>
<th>Clustering Coefficient Distance</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.452173913043478</td>
<td>0.7799999999999998</td>
<td>0.028753960481079772</td>
<td>1.260927873524558</td>
</tr>
<tr>
<td>30</td>
<td>0.280347826086956</td>
<td>0.6679999999999999</td>
<td>0.047650649446467006</td>
<td>0.995998475533423</td>
</tr>
<tr>
<td>40</td>
<td>0.502434782608696</td>
<td>0.7799999999999998</td>
<td>0.07324852349641087</td>
<td>1.355683306105106</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generations</th>
<th>Degree Distance</th>
<th>Path length Distance</th>
<th>Clustering Coefficient Distance</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.544</td>
<td>0.6679999999999999</td>
<td>0.06149314260586265</td>
<td>1.273493142605863</td>
</tr>
<tr>
<td>30</td>
<td>0.48504347826087</td>
<td>0.444</td>
<td>0.04397739872505719</td>
<td>0.973020876985927</td>
</tr>
<tr>
<td>40</td>
<td>0.339652173913043</td>
<td>0.6679999999999999</td>
<td>0.025670211050285476</td>
<td>1.033322384963329</td>
</tr>
<tr>
<td>Population Size</td>
<td>Probability to Mutate</td>
<td>Probability to Crossover</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------</td>
<td>--------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.9</td>
<td>0.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generations</th>
<th>Degree Distance</th>
<th>Path length Distance</th>
<th>Clustering Coefficient Distance</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.599478260869565</td>
<td>0.6679999999999999</td>
<td>0.04746415825637233</td>
<td>1.314942419125938</td>
</tr>
<tr>
<td>30</td>
<td>0.477391304347826</td>
<td>0.5559999999999999</td>
<td>0.0572633701417979</td>
<td>1.090654674489624</td>
</tr>
<tr>
<td>40</td>
<td>0.430260869565217</td>
<td>0.5559999999999999</td>
<td>0.03704039017668398</td>
<td>1.023301259741901</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Population Size</th>
<th>Probability to Mutate</th>
<th>Probability to Crossover</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>0.9</td>
<td>0.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generations</th>
<th>Degree Distance</th>
<th>Path length Distance</th>
<th>Clustering Coefficient Distance</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.544869565217391</td>
<td>0.6679999999999999</td>
<td>0.04754798510824299</td>
<td>1.260417550325634</td>
</tr>
<tr>
<td>30</td>
<td>0.259304347826086</td>
<td>0.5559999999999999</td>
<td>0.04709443964782161</td>
<td>0.862398787473908</td>
</tr>
<tr>
<td>40</td>
<td>0.213043478260869</td>
<td>0.6679999999999999</td>
<td>0.05521123371691503</td>
<td>0.936254711977784</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Population Size</th>
<th>Probability to Mutate</th>
<th>Probability to Crossover</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>0.9</td>
<td>0.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generations</th>
<th>Degree Distance</th>
<th>Path length Distance</th>
<th>Clustering Coefficient Distance</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.366434782608696</td>
<td>0.756</td>
<td>0.051082607395149734</td>
<td>1.173517390003846</td>
</tr>
<tr>
<td>30</td>
<td>0.304347826086957</td>
<td>0.7799999999999998</td>
<td>0.10085937463634298</td>
<td>1.185207200723299</td>
</tr>
<tr>
<td>40</td>
<td>0.495826086956521</td>
<td>0.6679999999999999</td>
<td>0.03286625876028967</td>
<td>1.196692345716811</td>
</tr>
<tr>
<td>Population Size</td>
<td>Probability to Mutate</td>
<td>Probability to Crossover</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------</td>
<td>--------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.1</td>
<td>0.9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generations</th>
<th>Degree Distance</th>
<th>Path length Distance</th>
<th>Clustering Coefficient Distance</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>0.54103478260869</td>
<td>0.7799999999999999</td>
<td>0.05704263576305375</td>
<td>1.378086114023923</td>
</tr>
<tr>
<td>30</td>
<td>0.710956521739130</td>
<td>0.6679999999999999</td>
<td>0.07695235040304996</td>
<td>1.45590872142180</td>
</tr>
<tr>
<td>40</td>
<td>0.743652173913043</td>
<td>0.7799999999999999</td>
<td>0.041354458842936856</td>
<td>1.565006632755980</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generations</th>
<th>Degree Distance</th>
<th>Path length Distance</th>
<th>Clustering Coefficient Distance</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>0.52121739130437</td>
<td>0.7799999999999999</td>
<td>0.03700593194331581</td>
<td>1.338223323247663</td>
</tr>
<tr>
<td>30</td>
<td>0.527304347826086</td>
<td>0.5559999999999999</td>
<td>0.09066502360202149</td>
<td>1.173969371428108</td>
</tr>
<tr>
<td>40</td>
<td>0.309043478260869</td>
<td>0.5559999999999999</td>
<td>0.03556316205116246</td>
<td>0.900606640312032</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generations</th>
<th>Degree Distance</th>
<th>Path length Distance</th>
<th>Clustering Coefficient Distance</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>0.34278260869562</td>
<td>0.756</td>
<td>0.03712303352759137</td>
<td>1.135905642223243</td>
</tr>
<tr>
<td>30</td>
<td>0.642956521739130</td>
<td>0.6679999999999999</td>
<td>0.046029376759869706</td>
<td>1.356985898498999</td>
</tr>
<tr>
<td>40</td>
<td>0.607652173913043</td>
<td>0.444</td>
<td>0.05174046706331774</td>
<td>1.103392640976361</td>
</tr>
</tbody>
</table>
The solutions that were provided by running the evolution program with these sets of parameters are quite similar. This fact implies that there cannot be drawn safe conclusions. However there is a trend of better results in test runs with greater populations and more generations. Another worth mentioning observation is that the algorithm seems to create fitter chromosomes when the probability of mutation is much higher than the probability of the crossover operation.

Finally due to the tendency of the algorithm to find better solutions in runs with many generation a test run of 100 generations was conducted. The size of the population was 20 chromosomes, the probability of mutation was set to 0.9 and the probability of crossover 0.1. The best chromosome wasn't much fitter than those in run with 40 generations. However it was observed that the whole set of non-dominated solutions was very close to the best non dominated solution. In contrast , when running the algorithm for 40 (or less) generations the set of non dominated solutions contained chromosomes that were very close to one target value and had great distance from the two others. The fact that they were very close to one value made it very difficult to be dominated by another chromosome.

4.2 Results with the use of a match file

After conducting a series of test – runs without using a match file, another run was conducted but in this case with the existence of a file match. It is important to mention that a different initial input network property file has been used. The different properties of this file are listed below :

```
Alphabet=A,T,C
Initial_node=ATCATCTCATCACT
Prob_to_add=0.0
Prob_to_delete=0.0
Prob_to_mutate=0.4
Prob_to_duplicate=0.6
```

The file match that was used was the same with that in [11] and contained :

```
AA =
AT =
AC = CA
TA =
TT =
TC = CT
CA = AC
CT = TC
CC =
```

The results obtained from this test run were much better, with the fitter chromosome having values for the five random seeds :

48
<table>
<thead>
<tr>
<th>Random Seed</th>
<th>Average degree</th>
<th>Average path length</th>
<th>Average clustering coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.826086956521739</td>
<td>4</td>
<td>0.285991848572551</td>
</tr>
<tr>
<td>2</td>
<td>6.078260869565217</td>
<td>4</td>
<td>0.263409594127956</td>
</tr>
<tr>
<td>3</td>
<td>5.834782608695652</td>
<td>4</td>
<td>0.262513599527718</td>
</tr>
<tr>
<td>4</td>
<td>6.208695652173913</td>
<td>4</td>
<td>0.310927200664651</td>
</tr>
<tr>
<td>5</td>
<td>5.643478260869565</td>
<td>4</td>
<td>0.293775566909393</td>
</tr>
</tbody>
</table>

However, the certain solution has the listed topological properties for the five random seeds. The average values over 100 tests aren't that close to the target values. The network property file of the provided solution included the values below:

Initial_node=CCCTCAAATTT  
Prob_to_add=0.0  
Prob_to_delete=0.016  
Prob_to_mutate=0.977  
Prob_to_duplicate=0.007

It is important to mention that the despite the fact that the average (over 100 runs) of the values weren't that close to the target values, the 83% of the networks generated by this input network property file fall in between the 20% of the the target values. This percentage is by far better, compared to the one reached when tuning the network's properties manually (34%), as done in [11].
Conclusion

The implementation of this evolutionary algorithm was a challenging and engaging task. Unfortunately, the discovery of the problematic fitness function caused many delays in the project and made it impossible to run tests, with other sets of input parameters, or even with different target values. If more test runs had been conducted, the conclusions on the best set of input parameters would have been more clear. However the fact that the algorithm provided solutions of which the 83% (over 100 runs) of the generated networks have a 20% variance from the target values, indicates that the use of this program can be beneficial to the network's properties tuning process.

Future Works

A first suggestion for future work, relates to the need of conducting test runs for different sets of parameters (eg. more generations). Moreover it would be useful to test the performance of the algorithm on different fitness functions. This may include the implementation and testing of a simple aggregate function (a Single Objective approach). Another approach for checking if a chromosome Pareto dominates another may also consider comparing the values of all random seeds one by one instead of using the average (as described in section 3.2.2.2). Finally, the number of network properties that the program tries to match can be increased.
References


[4]. C. Coello Coello, G. Lamont, Applications of Multi-Objective Evolutionary Algorithms (Advances in Natural Computation), Lamont, 2005

[5]. D.W. Corne, Web Intelligence lecture notes, Heriot – Watt University, Edinburgh


[9]. C. Fonseca and P. Fleming, An Overview of Evolutionary Algorithms in Multiobjective Optimization, University of Sheffield, April 7, 1995


[23]. The University of Sheffield, http://www.shef.ac.uk/acse/staff/peter_fleming/intromo.html , April 2011

Appendix A

Classes Definition

The Java project that implements the evolutionary algorithm has two different packages. The first package is the SN – model algorithm which is used in order to create networks. The second package is the evolution program which is consisted of nine classes. Each class is responsible for different tasks. The classes are:

- EvolutionProgramMain
- Individual
- Population
- Evolution
- Evo_IO_Manager
- EvoConstants
- DominanceComparator
- FitnessComparator
- LongNum

The basic tasks of each class are described below. Moreover a list of the methods that each class contains will be provided.

**EvolutionProgramMain**

This class included the Main method of the program. From this point the program starts and at this point it ends. The main method calls the constructor of the initial population (Population class) as well as the constructor of the Evolution class. Another operation of the main method is that it calculates the duration of the program run by storing the System time on startup and subtracting it from the System time just before the program's termination.

<table>
<thead>
<tr>
<th>EvolutionProgramMain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main ( String [ ] args )</td>
</tr>
</tbody>
</table>

**Individual**

This class is responsible for all operations that relate to one chromosome. The objects of this class represent the chromosomes of the evolutionary algorithm. It contains many different methods ranging from calculation of individual's distances from target values ,to the mutation operation.
Individual

- avg_degree : double [ ]
- avg_path_length : double [ ]
- avg_clust_coeff : double [ ]
- avg_degree_distance: double [ ]
- avg_path_length_distance: double [ ]
- avg_clust_coeff_distance: double [ ]
- avg_distances : double [ ]
- dominated : int
- fitness : double
- props : Properties
- evoProperties : Properties

+ Individual(prop : Properties, evoProperties : Properties)
+ Individual(evoProperties : Properties)
+ addIndividualStats(degree : double,length : double,clustCoeff : double,rand_seed : int): void
+ getIndividualProperties() : Properties
+ setIndividualProperties(properties : Properties) : void
+ getEvoProperties() : Properties
+ setEvolutionProperties(evo : Properties) : void
+ setSomeProperties(parent : Individual) : void
+ calculateFitness() : void
+ getFitness() : double
+ calculateDistances() : void
+ setDominated() : void
+ resetDominance(): void
+ getDominated () : int
+ getAvgDegreeDistance() : [ ] double
+ getAvgPathDistance() : [ ] double
+ getAvgCcDistance() : [ ] double
+ getAvgDistances() : [ ] double
+ mutate(indi : Individual) : Individual
+ mutateProbabilities(indi : Individual) : void
+ mutateInitialNode(indi : Individual) : void
+ increaseNodesStructureLength([ ]letters : String, initialNode : String) : String
+ decreaseNodesStructureLength(initialNode : String) : String
+ distributeProbabilities(indi : Individual, probToBeChanged : String): void
+ roundThreeDecimals(s : Float) : Float

Population

This class contains an array list of individuals. In other words it is a collection of individuals. The methods that are included in this class are related to the manipulation of a whole set of individuals. Instances of this class represent the population of chromosomes in the evolutionary algorithm.
Population

- Population : ArrayList<Individual>

+ Population()
+ addToPopulation(indi : Individual) : void
+ removeFromPopulation(indi : Individual) : void
+ getPopulationSize() : int
+ sortByFitness() : void
+ sortByDominance() : void
+ getIndividualByIndex(index : int) : Individual
+ clearPopulation() : void
+ initializePopulation( evolutionProperties : Properties) : void
+ sortByFitnessReport(best : int) : String
+ runModelNetwork(indi : Individual) : Individual
+ runModelNetworkSimple(indi : Individual, indiNo : int) : void
+ uniqueIndividual( indi : Individual, pop : Population) : boolean
+ readData(indi : individual, host : String, directory : String) : void
+ compareIndividuals( a : Individual, b : Individual) : boolean

Evolution

The constructor of this class is only called once and the instance of the class represents the evolution procedure. Apart from its constructor it contains one method in which the evolution starts and lasts for as many generations as specified. In this method many instances of the population class are created.

Evolution

- networkPopulation : Population
- bestOfGeneration : Population
- intermediatePop : Population
- evolutionProperties : Properties
- children : Individual[]

+ Evolution(pop : Population, evoProp : Properties)
+ startEvolution() : void

Evo_IO_Manager

This class contains methods that handle the input and output of the algorithm. Its main operation is to check the properties that are provided as input from the property file and if errors exist, inform the user properly. Another important task that is executed in methods of this class is the results production.
### Evo_IO_Manager

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkProperties(evo : Properties , prop : Properties) : void</td>
<td></td>
</tr>
<tr>
<td>checkResultsDirectory() : void</td>
<td></td>
</tr>
<tr>
<td>loadEvolutionProperties(evoProps : String) : Properties</td>
<td></td>
</tr>
<tr>
<td>checkEvolutionProperties(evoProps : Properties) : boolean</td>
<td></td>
</tr>
<tr>
<td>parseDoubleValue(value : String) : double</td>
<td></td>
</tr>
<tr>
<td>parseIntValue(value : String) : int</td>
<td></td>
</tr>
<tr>
<td>writeToFile(fileName : String , report : String) : void</td>
<td></td>
</tr>
<tr>
<td>copyFiles(fileName : String , baseDir : String, randSeed : int , indiNo : int) : void</td>
<td></td>
</tr>
<tr>
<td>produceResults(pop : Population , bestIndiNo : int) : void</td>
<td></td>
</tr>
</tbody>
</table>

### EvoConstants

This class contains string constants. The values of these constants derive from the property file. They are used as constants inside the program in order to facilitate any refactoring task that the program may need.

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVO_PROPERTIES_FILE</td>
<td></td>
</tr>
<tr>
<td>EVO_RANDOM_SEED</td>
<td></td>
</tr>
<tr>
<td>PROBABILITY_TO_MUTATE</td>
<td></td>
</tr>
<tr>
<td>PROBABILITY_TO_CROSSOVER</td>
<td></td>
</tr>
<tr>
<td>PROBABILITY_TO_MUTATE_PROBABILITIES</td>
<td></td>
</tr>
<tr>
<td>PROBABILITY_TO_MUTATE_INITIAL_NODE</td>
<td></td>
</tr>
<tr>
<td>POPULATION_SIZE</td>
<td></td>
</tr>
<tr>
<td>TARGET_DEGREE</td>
<td></td>
</tr>
<tr>
<td>TARGET_PATH_LENGTH</td>
<td></td>
</tr>
<tr>
<td>TARGET_CLUST_COEFF</td>
<td></td>
</tr>
<tr>
<td>NUMBER_OF_GENERATIONS</td>
<td></td>
</tr>
<tr>
<td>INDIVIDUALS_FOR_RESULTS</td>
<td></td>
</tr>
<tr>
<td>PROBABILITY_DEVIATION</td>
<td></td>
</tr>
<tr>
<td>NUMBER_OF_NODES</td>
<td></td>
</tr>
<tr>
<td>MIN_NODE_STRUCTURE_LENGTH</td>
<td></td>
</tr>
<tr>
<td>MAX_NODE_STRUCTURE_LENGTH</td>
<td></td>
</tr>
<tr>
<td>FITNESS_REPORT</td>
<td></td>
</tr>
<tr>
<td>MAX_NUM_CROSSOVER_ATTEMPTS</td>
<td></td>
</tr>
<tr>
<td>CLUSTER</td>
<td></td>
</tr>
<tr>
<td>COMPUTER_NAMES</td>
<td></td>
</tr>
<tr>
<td>CLUSTER_DIRECTORY</td>
<td></td>
</tr>
<tr>
<td>USERNAME</td>
<td></td>
</tr>
</tbody>
</table>

### DominanceComparator

This class is used for sorting the population of chromosomes according to their dominance score. Despite the fact that the dominance score is an integer number, this class was implemented as more than one was of sorting the population are needed. The class implements the Comparator Interface and has
the required \textit{compare} method. The sorting of the chromosomes is in ascending order.

\begin{tabular}{|c|}
\hline
\textbf{DominanceComparator} \\
+\text{compare}(\text{indi1} : \text{Individual}, \text{indi2} : \text{Individual}) : \text{int} \\
\hline
\end{tabular}

\textbf{FitnessComparator} \\
In the same way as DominanceComparator class this one sorts the chromosomes according to their final fitness score. It is used only to sort the population of non-dominated solutions that are to be provided as results. In more details, when the evolution procedure has terminated, there is a set of non-dominated solutions. Instead of providing all of them as results, the algorithm calculates the sum of the distances from the three target values for each non-dominated solution. As a result each chromosome has a single score. The lower the score is, the better the chromosome is considered. This class sorts the final non-dominated chromosomes in ascending order so that the best “n” (as specified by the user) can be provided as results.

\begin{tabular}{|c|}
\hline
\textbf{FitnessComparator} \\
+\text{compare}(\text{indi1} : \text{Individual}, \text{indi2} : \text{Individual}) : \text{int} \\
\hline
\end{tabular}

\textbf{LongNum} \\
This class was implemented by Pierluigi Frisco (supervisor of this project) in order to overcome an accuracy problem with numbers with lots of decimal places. The use of Long or Double or even BigDecimal types of the Java language was problematic. This class manipulates numbers as strings and instances of LongNum were used for storing network's probabilities values.
Figure 16: Class Diagram