Optimal Route Selection in Computer Network using Genetic Algorithm

HUYAM YOUNIS ABDALLA GISMALLA

B.Sc. of Science (Honours) in Mathematics and Computer Science, University of Gezira (2014)

A Dissertation
Submitted to University of Gezira in Partial Fulfilment of the Requirements for the Degree of Master of Science

in

Computer Science

Department of Computer Science

Faculty of Mathematical and Computer Sciences

April, 2018
Optimal Route Selection in Computer Network using Genetic Algorithm

HUYAM YOUNIS ABD ALLAH GISSM ALLAH

Supervision Committee:

<table>
<thead>
<tr>
<th>Name</th>
<th>Position</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dr. Awadallah Mohammed Ahmed Ali</td>
<td>Main Supervisor</td>
<td>................</td>
</tr>
<tr>
<td>Dr. Mohammed Albarra Hassan</td>
<td>Co-supervisor</td>
<td>................</td>
</tr>
</tbody>
</table>

Date: April, 2018
Optimal Route Selection in Computer Network using Genetic Algorithm

HUYAM YOUNIS ABDALLA GISMALLA

Examination Committee:

<table>
<thead>
<tr>
<th>Name</th>
<th>Position</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dr. Awadallah Mohammed Ahmed Ali</td>
<td>Chairperson</td>
<td>...............</td>
</tr>
<tr>
<td>Dr. Mohammed Babiker Ali</td>
<td>External examiner</td>
<td>...............</td>
</tr>
<tr>
<td>Dr. Sally Dafaallah Awadelkriem</td>
<td>Internal examiner</td>
<td>...............</td>
</tr>
</tbody>
</table>

Date: 5, May, 2018
DECLARATION

To my father

To my mother

To my brothers and sisters

To my friends and everyone supported me
To teachers and supervisors.
ACKNOWLEDGEMENTS

I would like to thank Dr. Awadallah Mohamed Ahmed for his useful advices and unlimited support; I also extend my sincere thanks and appreciation for Dr. Mohamed Albaraa Hassan for the precious advices and guidance’s.
Optimal Route Selection in Computer Network using Genetic Algorithm

HUAYAM YOUNIS ABD ALLAH GISSM ALLAH

ABSTRACT

Due to huge demand of Internet, ISPs are trying to meet the increasing traffic demand with improved utilization of existing resources and application of new technologies. Routing is one of the most important issues that have a significant impact on the network's performance, the process of finding the optimal paths between nodes in network is a hard special in large networks because there are many potential intermediate destinations. Data routing in computer networks is a process of transferring packets from source node to destination node with minimum cost. Metaheuristic algorithms such as genetic algorithm is widely employed for solving the optimization problems. Genetic Algorithm is a metaheuristic method based on the mechanics of natural genetics and natural selection. The problem of finding a path with minimum cost between a given pair of nodes is one of the most fundamental problems in computer networks. The main objective of this research is to study existing solutions and to choose an appropriate method for solving the research problem and to evaluate the solution to find near optimal route from a single source node to destination node. The methodology solve this problem which formulated as a mathematical model. Many research works have been introduced alternative methods and techniques for solving shortest path problem in networks. This research proposes a solution based on genetic algorithm to find the near optimal solution for the SSP. The used genetic operators are tournament selection with elitism techniques, multipoint crossover and insertion mutation. The crossover operation exchanges partial chromosomes and the mutation operation maintains the genetic diversity of the population. The proposed algorithm has been tested using many instances of network and experimental results show that the proposed algorithm finds the path with minimum cost from a single source node to the destination node for any given network of different sizes (5, 6, and 30). In addition to that the results proved the usefulness of the GA in finding the near optimal solution for SPP. As future work it is recommended to combine the GA with simulated annealing algorithm for mutation to get faster diversity in larger network sizes.
اختيار أفضل وجهة في شبكة الحاسوب باستخدام الخوارزمية الجينية

هيم يونس عبدالله قسم الله

ملخص الدراسة

 بسبب الطلب الهائل على الإنترنت، يحاول مزودو خدمات الإنترنت تلبية الطلب المتزايد على حركة المرور مع تحسين استخدام الموارد القائمة وتطبيق التقنيات الجديدة. التوجه هو واحد من أهم المسائل التي لها تأثير كبير على أداء الشبكة. عملية العثور على المسارات المثلى بين العقد في الشبكة صعبة في الشبكات الكبيرة لوجود العديد من الوجهات الوسيطة المحتملة. توجه البيانات هو عملية نقل الحزم من العقد المصدر إلى العقدة الوجهة بأقل تكلفة. الخوارزميات الفوقية مثل الخوارزمية الجينية هي الأكثر استعمالاً لحل معظم مشاكل الأمثلية. الخوارزمية الجينية هي خوارزمية فوقية لطريقة الأمثلية المبنية على أساس الميكانيكا الوراثية الطبيعية والاختيار الطبيعي. عملية العبور تبادل جزئياً الكروموسومات وعملية الطفرة تحافظ على التنوع الوراثي للمجتمع. مشكلا إيجاد مسار بأقل تكلفة بين زوج معين من العقد هي واحدة من أهم المشاكل الأساسية في شبكات الحاسوب. الهدف الرئيسي من هذا البحث هو تطوير حل لحل مشكلة أقصر مسار للحصول على الطريق الأمثل من مصدر واحد لعقدة الوجهة. هناك العديد من الأعمال البحثية لإيجاد طرق وتقنيات بديلة لإيجاد أقصر الطرق في الشبكات. يقترح هذا البحث خوارزمية جينية لإيجاد الحل الأقرب للأمثلة المشكلة. الخوارزمية المقترحة التي تم تنفيذها هي اختيار المبارة بتقنية الإنتخاب متعددة النقاط وإدراج الطفرة. أظهرت النتائج أن الخوارزمية المقترحة يمكنها إيجاد المسار الأقرب من عقد مصدر واحد إلى عقدة الوجهة لأي بنية شبكة بمختلف الأحجام (5، 6، 30). وقد تم تطبيق الخوارزمية المقترحة في العديد من أمثلة بنية الشبكات وأظهرت النتائج فائدة الخوارزمية الجينية في إيجاد الحل الأقرب للأمثل. توصي الدراسة بدمج الخوارزمية الجينية مع خوارزمية محاكاة التلدين للطفرة للحصول على أسرع حل في الشبكات كبيرة الحجم.
# TABLE OF CONTENTS

Declaration .................................................................................................................................................. iv

Acknowledgements ....................................................................................................................................... v

Abstract ...................................................................................................................................................... vii

ملخص الدراسة ......................................................................................................................................... vii

Table of Contents ...................................................................................................................................... viii

List of Tables ............................................................................................................................................... xi

List of Figures ............................................................................................................................................. xii

List of Abbreviations ................................................................................................................................. xiv

## CHAPTER ONE  INTRODUCTION ........................................................................................................ 1

1.1 Introduction ............................................................................................................................................ 1

1.2 Problem Statement ................................................................................................................................. 2

1.3 Research Objective ................................................................................................................................. 2

1.4 RESEARCH LAYOUT ........................................................................................................................... 2

## CHAPTER TWO  LITERATURE REVIEW ........................................................................................... 3

2.1 Background ............................................................................................................................................ 3

2.2 Genetic Algorithm ................................................................................................................................. 3

2.2.1 Selection operator ............................................................................................................................ 4
2.2.2 Crossover (Recombination) ................................................................. 4
2.2.3 Mutation .............................................................................................. 5
2.2.4 Replacement ......................................................................................... 5
2.3 Optimization Problem ............................................................................ 6
2.4 Graph ....................................................................................................... 6
2.4.1 Directed Graph ..................................................................................... 6
2.4.2 Undirected Graph ................................................................................ 7
2.4.3 Weighted Graph ................................................................................... 7
2.4.4 Cycle Graph .......................................................................................... 8
2.5 Routing ..................................................................................................... 8
2.6 Related Works .......................................................................................... 9

CHAPTER THREE RESEARCH METHODOLOGY ....................................... 13

3.1 Introduction ............................................................................................. 13
3.2 Network Model ........................................................................................ 13
3.3 Problem formulation ............................................................................... 14
3.4 Proposed solution .................................................................................... 14
3.4.1 Chromosome Representation .............................................................. 14
3.4.2 Fitness Function .................................................................................... 15
3.4.3 Initialize Population ............................................................................. 16
3.4.4 Selection (reproduction) ..................................................................... 16

3.4.4.1 Tournament Selection ....................................................................... 16
3.4.4.2 Elitism

3.4.5 Crossover

3.4.5.1 Two point crossover

3.4.6 Mutation

3.4.7 Replacement

3.4.8 Termination Condition

CHAPTER FOUR

4.1 Experimental setup

4.1.1 Instance 1

4.1.2 Instance 2

4.1.3 Instance 3

4.2 Result discussion

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

5.2 Recommendation

References

Appendix
LIST OF TABLES

Table No. | Page No.
--- | ---
Table 2.1 Related works comparison | 11
Table 3.1: The cost on links | 13
Table 4.1: Instance 1 parameters settings | 24
Table 4.2: Instance 2 parameters settings | 26
Table 4.3: The possible paths instance 2 | 27
Table 4.5: Instance 3 parameters settings | 29
<table>
<thead>
<tr>
<th>Figure No.</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1 Directed Graph</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2.2 Undirected Graph</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2.3 A cycle graph</td>
<td>8</td>
</tr>
<tr>
<td>Figure 3.1: Graph model</td>
<td>14</td>
</tr>
<tr>
<td>Figure 3.2: The path representation</td>
<td>15</td>
</tr>
<tr>
<td>Figure 3.3: Genes representation</td>
<td>15</td>
</tr>
<tr>
<td>Figure 3.4: Edges initialization</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3.5: Tournament selection method</td>
<td>17</td>
</tr>
<tr>
<td>Figure 3.6: Two selected parents</td>
<td>18</td>
</tr>
<tr>
<td>Figure 3.7: The produced offspring</td>
<td>18</td>
</tr>
<tr>
<td>Figure 3.8: Selected chromosome for mutation</td>
<td>19</td>
</tr>
<tr>
<td>Figure 3.9: Gene 2 out</td>
<td>19</td>
</tr>
<tr>
<td>Figure 3.10: Reinsert gene 2 randomly</td>
<td>19</td>
</tr>
<tr>
<td>Figure 3.11: An old solution</td>
<td>19</td>
</tr>
<tr>
<td>Figure 3.12: The path replacement</td>
<td>19</td>
</tr>
<tr>
<td>Figure 3.13: The genetic algorithm flow chart</td>
<td>20</td>
</tr>
<tr>
<td>Figure 4.1 Example of network topology</td>
<td>23</td>
</tr>
<tr>
<td>Figure 4.2: Genetic-based solution</td>
<td>24</td>
</tr>
<tr>
<td>Figure 4.3: Instance 1 obtained solution</td>
<td>24</td>
</tr>
<tr>
<td>Figure 4.4: Instance 1 fitness values</td>
<td>25</td>
</tr>
<tr>
<td>Figure 4.5: Example for Network topology</td>
<td>26</td>
</tr>
<tr>
<td>Figure 4.6: Genetic algorithm-based solution</td>
<td>27</td>
</tr>
<tr>
<td>Figure 4.7: GA solution instance 2</td>
<td>27</td>
</tr>
<tr>
<td>Figure 4.8: Instance 2 the fitness values</td>
<td>28</td>
</tr>
<tr>
<td>Figure 4.9: Example for Network topology</td>
<td>29</td>
</tr>
<tr>
<td>Figure 4.10: The result of the algorithm</td>
<td>30</td>
</tr>
<tr>
<td>Figure 4.11: GA solution for given network</td>
<td>30</td>
</tr>
<tr>
<td>Figure 4.12: Fitness values instance 3</td>
<td>31</td>
</tr>
</tbody>
</table>
# LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>SP</td>
<td>Shortest Path</td>
</tr>
<tr>
<td>SSP</td>
<td>Shortest path problem</td>
</tr>
<tr>
<td>PMC</td>
<td>Partial mapped crossover</td>
</tr>
<tr>
<td>Pm</td>
<td>Mutation probability</td>
</tr>
<tr>
<td>Pc</td>
<td>Crossover probability</td>
</tr>
</tbody>
</table>
CHAPTER ONE
INTRODUCTION

1.1 INTRODUCTION

Routing is one of the most important issues that have a significant impact on the network's performance which is a process of finding paths between nodes in network it is hard in large networks because of the many potential intermediate destinations. Data network routing is a process of transferring packets from source node to destination node with minimum cost (Kumar, 2010). Routing algorithm responsible for deciding the path for incoming packets on which they should be transmitted. There are several search algorithms for the shortest path (SP) problem: the breadth-first search algorithm, the Dijkstra’s algorithm and the Bellman-Ford algorithm. Broadly there are mainly two types of routing policies: In static routing, the routes between the nodes are pre-computed based on certain factors, all packets between any two nodes follow the same path. When network topology changes the path between two nodes may also change and static routing fails. Hence in dynamic routing policy, the routes are not stored but are generated when required. Routing process uses routing table at each node to store all nodes, then the router decides which neighbour to choose from routing table to reach specified destination.

Some network performance Metrics are: utilization, throughput, response time and delay. Link performance may be measured in terms of bandwidth or link delay (Kumar, 2010).

Optimization relates to the optimum solution of a problem, when a model is proposed, cost and profit value analysis is a selective parameter for decision-making. Proposed cost should be less and profit should be more, in other words, for minimum input, maximum output should be achieved the process of attaining the selected parameters at minimum rate is optimization (Sharma, et al., 2016). The goal is to find the values of the variables in the process that yield the best value of the performance criterion. Some typical performance criteria: Maximum profit, minimum cost, minimum effort, minimum error, minimum waste, best product quality. The metric of optimization in the proposed work is cost of path between the nodes. The total cost is the sum of cost of individual hops make it as minimum as possible. The artificial intelligence
techniques are the most widely employed tool for solving most of the optimization problems. These methods (e.g. genetic algorithm simulated annealing and tabu search) seem to be promising and are still evolving (Gandomkar, et al., 2005). Genetic Algorithm is a non-traditional and optimization method based on the mechanics of natural genetics and natural selection (Ghose, 2002).

1.2 PROBLEM STATEMENT

Having a graph $G = (V, E)$ undirected, connected graph, where $E$ is the set of weighted edge and $V$ is a set of vertices each vertex is associated with one hope, the weighted edge represents the distance / cost between two hopes. There are two nodes source node and destination node both of them are defined as an inputs. The goal is how to find the path with the minimum cost between source node and destination node.

1.3 RESEARCH OBJECTIVE

The main objective of this research is to study existing solutions and to choose an appropriate method for solving the research problem and to evaluate the solution.

1.4 RESEARCH LAYOUT

The research is organized in five chapters as follows:
Chapter two presents the literature review, the related studies and genetic algorithm.
Chapter three, present the methodology that used to achieve the research objectives
Chapter four contains the results and discussion, finally chapter five presents the conclusion and the recommendations. Finally there are the references list and the appendix.
CHAPTER TWO
LITERATURE REVIEW

2.1 BACKGROUND

This chapter reviews some of the definitions related to the area of research.

2.2 GENETIC ALGORITHM

Genetic Algorithm (GA) basic elements are: set of chromosomes represent the possible solutions, the selection according to fitness value, the crossover to produce anew offspring, and random mutation of new children.

GA search space of a given problem (set of possible solution) contains a collection of individuals (chromosomes), each chromosome contain many entity called genes. Each gene represents an entity that is structurally independent of other genes. These individuals are encoded (encoding is a process of representing individual genes) using binary strings, real numbers, integer numbers or strings of letters depending on the given problem. (Sivanandam, 2007)

GA starts with generating a large set of possible solutions to a given problem, then each solution had to be evaluated to determine the fitness value using the fitness function. The fittest individuals are selected for genetic operations (crossover, mutation) and become population to the next generation. GA explores entire solution space to reach an optimal set of solutions. (Sivanandam, 2007)

The steps of GA:

1. Initialization:
   Create random initial population across the search space (set of feasible solutions). Each individual represent a candidate solution also, called a chromosome, each individual has a fixed-length.

2. Evaluation:
   Once the population initialized the individuals, and then they will be evaluated based on the fitness function.

3. Selection:
   Select individuals among population according to their fitness function i.e. individuals with highest fitness values copied to the next generations while the
lowest ones ignored or eliminated, the main idea of the selection process is to prefer better solutions to worse ones. Many selection methods includes: roulette wheel selection, ranking selection, random selection, elitism and tournament selection. (Sivanandam, 2007)

These methods discussed below:

**2.2.1 Selection operator**

i. **Roulette Wheel Selection**

The probability proportional to the fitness value in proportional roulette wheel, individuals are selected with a probability that is directly proportional to their fitness values the individual is selected based on the relative fitness with its competitors. This is similar to dividing the wheel into a number of slices, fittest chromosomes get larger slice (Kumar, 2010).

ii. **Random Selection**: Randomly selects a parent from the population.

iii. **Ranking Selection**: rank-based selection scheme it first sort the individuals in the population according to their fitness and then computes selection probabilities according to their rank rather than fitness values (The worst has fitness 1 and the best has fitness N).

iv. **Tournament Selection**: It’s a method for selecting individuals from the population. The selection involves running several "tournaments" among a few individuals chosen at random from large population. The winner of each tournament (the one with the best fitness) is selected for crossover. When the tournament size which is the number of individuals compete each tournament is small, then the tournament selection gives a chance to all individuals to be selected. If the tournament size is large then weak individuals have a smaller chance to be selected may cause loss of diversity. (Sivanandam, 2007)

**2.2.2 Crossover (Recombination)**

Crossover is done to explore a new solution space, to combine some parts of selected parents to create a new offspring. The idea is to take the best characteristics from the parents. Crossover is made in hope that the new chromosomes will contain good parts from the old chromosomes.

Crossover types:

a. Single point crossover: select randomly crossover point within paired chromosomes and swap the part after crossover point first parent with
the second one to produce two new children (chromosomes) that hold some part of both the parent genetic material.

b. Two pint crossover:
Choose two crossover sites on parents at random and the segment between two pint is swapped between the parents to produce two new children, the advantage of having more crossover points is that the problem space may be searched more thoroughly (Sivanandam, 2007).

c. Uniform crossover:
The two children are chosen according to a random generated binary crossover mask of the same length as the chromosome. Where there is a one in the crossover mask, the gene is copied from the first parent, and where there is a zero in the mask the gene is copied from the second parent. A new crossover mask is randomly generated for each pair of parents. The child therefore contains a mixture of genes from each parent (Sivanandam, 2007).

2.2.3 Mutation
Mutation applied after crossover operation by changing some genes in the chromosome randomly, Mutation prevents the algorithm to be trapped in a local minimum, plays the role of recovering the lost genetic materials and maintains diversity in the population (Sivanandam, 2007).

2.2.4 Replacement
Once new offspring solutions are created using crossover and mutations operations, there is a need to introduce them into parental population. The strategy to replace the worst members (chromosomes) from the population with the new ones called replacement of the chromosomes.

Finally repeat steps (evaluation, selection, crossover, mutation and replacement) until termination condition is reached.

Outlines of basic of Genetic Algorithm

1. [Start] generate random population of n chromosomes (suitable solutions for the problem)
2. [Fitness] Evaluate the fitness function f(x) of each chromosome x in the population
3. [New population] Create a new population by repeating following steps until the new population is complete.
a. [Selection] select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to get selected).

b. [Crossover] with a crossover probability, crossover the parents to form new offspring (children). If no crossover was performed, offspring is the exact copy of parents.

c. [Mutation] with a mutation probability, mutate new offspring at each locus (position in chromosome).

d. [Accepting] place new offspring in the new population.

4. [Replace] use new generated population for a further sum of the algorithm.

5. [Test] if the end condition is satisfied, stops, and returns the best solution in current population.


2.3 OPTIMIZATION PROBLEM

Optimization is the act of achieving the best possible result under given circumstances. In design, construction, maintenance... engineers have to take decisions. The goal of all such decisions is either to minimize effort or to maximize benefit. The effort or the benefit can be usually expressed as a function of certain design variables. Hence, optimization is the process of finding the conditions that give the maximum or the minimum value of a function (Astolfi, A, 2005).

2.4 GRAPH

A graph is a way of representing connections or relationships that exist between pairs of objects, a graph G is simply a set V of vertices(nodes) and a collection E of pairs of vertices from V, called edges(arcs). Graphs modelling can be used in optimization problems in many domain applications including mapping, transportation, computer networks, and electrical engineering (Goodrich, et al., 2008).

2.4.1 Directed Graph

A directed graph, also called a digraph, is a graph whose edges are all directed (Weiss, 2007). Directed graph shown in figure 2.1.
2.4.2 Undirected Graph

If an edge is directed, its first endpoint is its origin and the other is the destination of the edge.

If all the edges in a graph are undirected, then the graph will be an undirected graph (Weiss, 2007).

Figure 2.2 Undirected Graph.

2.4.3 Weighted Graph

A weighted graph is a graph that has a numeric value (for example, integer) label $w(e)$, associated with each edge $e$ called a weight of edge $e$, often referred as the “cost” of the edge. For $e = (u, v)$, let notation $w(u, v) = w(e)$. Usually the edge weights are nonnegative.

An edge (undirected or directed) is a self-loop if its two endpoints coincide. A self-loop may occur in a graph associated with a city map.
A path is a sequence of alternating vertices and edges that starts at a vertex and ends at a vertex such that each edge is incident to its predecessor and successor vertex.

Said that a path is simple if each vertex in the path is distinct, and a cycle is simple if each vertex in the cycle is distinct, except for the first and last one.

The distance from a vertex to a vertex $v$ in $G$, denoted $d(u, v)$, is the length of a minimum-length path (also called shortest path) from $u$ to $v$, if such a path exists (Goodrich, et al., 2008).

**2.4.4 Cycle Graph**

A cycle is a path that starts and ends at the same vertex, and that includes at least one edge (Weiss, 2007).

![A cycle graph](image)

**Figure 2.3 A cycle graph**

A directed path is a path such that all edges are directed and are traversed along their direction. A graph is connected if all any vertices are there connected to each other (Goodrich, et al., 2008).

**2.5 ROUTING**

Routing is a process of transferring packets from source to destination with minimum cost. Cost factors may be the distance of router, network throughput of a link or link availability and reliability. In a large network routing is an issue if the source and destination are not on the same network.
Routing process uses a data structure called routing table at each node to store all the nodes which are at one hop distance from it. The algorithms that choose the routes and data structure is the major issue of network layer design. The routing algorithms are the part of the network layer and are used to specify the best optimal path on a large network. The route followed by the packets may change every time when a packet arrives every time or it may remain same just as followed by the first packet. Routing algorithms can be grouped in two categories first is adaptive and the second is non-adaptive. Adaptive routing algorithm changes their routing decisions to reflect changes in the topology or traffic load on the links.

In contrast, Non-adaptive routing algorithms do not base their routing decisions on measurements or estimates of the current traffic and topology of the network. There are number of routing algorithms under the categories adaptive and non-adaptive. These are used to find the path between sources and destinations. Most of these algorithms work on the cost to find the routs so each path evaluated in terms of cost (Kumar, et al., 2014).

2.6 RELATED WORKS

Many research works have been introduced to find and solve SPPs. The following section presents some of the most relevant research works.

In 2005 (Abeyesundara, et al., 2005) propose a new method to solve the shortest path problem using genetic algorithms, Roulette rank selection is used and multi point crossover. The experimental results show that this algorithm finds more than one possible solution for a given source and destination and this makes it easy to find the next shortest path which exists other than the optimal solution.

In 2010 (Kumar, 2010) implemented a genetic algorithm to finds the set of optimal routes to send the traffic from source to destination. The Dijkstra’s algorithm was considered as the most efficient method for shortest path computation in computer networks. But when the network is becomes very big, then it becomes inefficient since a lot of computations need to be repeated. Also it cannot be implemented in the permitted time, advanced multipoint crossover, roulette wheel selection and insertion
mutation are used. Results found that fitness value improves at each generation for chromosomes. The chromosomes has fixed-length its limitation.

In 2010 (Hamed, 2010) proposes a genetic algorithm to determine the k shortest paths with bandwidth constraints from a single source node to multiple destinations nodes, crossover operation is performed by one-cut point, mutation operation is performed on bit-by-bit. Results show that proposed algorithm is considered to be the first algorithm that uses the genetic algorithms to obtain the k shortest paths from a single source node to multiple destinations nodes.

In 2010 (Fadil, 2010) Presented a genetic algorithm for solving the shortest path routing problem, two point crossover is used, roulette wheel selection and flipping mutation. The results shown that the algorithm can search the solution space in a very effective manner.

In 2012 (Krishna, et al., 2012) implemented a Genetic algorithm to finds the set of optimal routes to send the traffic from source to destination, Genetic algorithm is used for routing in packet switched data networks, GA elements are: roulette wheel selection, PMX crossover and insertion mutation, limitation it does not support load balancing.

In 2014 (Kumar, et al., 2014) presents a Genetic Algorithm to find the shortest path with minimum total cost between source node and the destination node. Used roulette wheel selection and GA is produced the same result as the Dijkstra’s algorithm. Limitation is the fixed length chromosome.

In 2016 (Moza, et al., 2016) they proposed the work using two point crossover technique. The work features selection of an optimal path using genetic algorithm.

As shown above many researches have been introduced to solve SP problem using different GA operations techniques in various networks topologies, most of them used roulette wheel selection the proposed works used the tournament selection. Another difference this work uses multipoint crossover and insertion mutation. Maximum fitness value used as termination condition in proposed work.
Table 2.1 Related works comparison.

<table>
<thead>
<tr>
<th>Num</th>
<th>year</th>
<th>Author</th>
<th>work</th>
<th>methodology</th>
<th>Advantage</th>
<th>Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2005</td>
<td>Abeysundara et al.</td>
<td>A GA approach to solve the SPP for road maps.</td>
<td>Roulette rank selection is used and multi point crossover</td>
<td>propose a new method to solve the shortest path problem using GA</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2010</td>
<td>Kumar</td>
<td>Exploring genetic algorithm for shortest path optimization in data networks.</td>
<td>Advanced multipoint crossover, roulette wheel selection and insertion mutation.</td>
<td>implemente d a genetic algorithm to finds the set of optimal routes to send the traffic from source to destination</td>
<td>Fixed-length chromosomes</td>
</tr>
<tr>
<td>3</td>
<td>2010</td>
<td>Hamed</td>
<td>A genetic algorithm for finding the k shortest paths in a network.</td>
<td>One-cut point crossover, bit-by-bit mutation.</td>
<td>proposes a GA to determine the k shortest paths with bandwidth constraints from a single source node to multiple destinations nodes</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>Year</td>
<td>Author(s)</td>
<td>Title</td>
<td>Details</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>------</td>
<td>-----------</td>
<td>-------</td>
<td>---------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2014</td>
<td>Kumar, et al.</td>
<td>Role of Genetic Algorithm in Routing for Large Network.</td>
<td>roulette wheel selection</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER THREE
RESEARCH METHODOLOGY

3.1 INTRODUCTION

This work proposed the genetic algorithm which implemented to find the shortest path in computer networks.

3.2 NETWORK MODEL

The network under consideration is represented as $G = (V, E)$ undirected, weighted and connected graph, where $E$ is the set of weighted edge and $V$ is a set of vertices each $e_i$ nodes has positive cost assigned to each link (edge) connects two sequenced nodes in the path randomly, assume that the nodes are perfectly reliable and there is no redundant links in network. Assume that a network with 6 nodes (vertices) and 8 edges as in figure 3.3.1. Each link has a non-negative cost randomly generated between 0 and 7 as shown in table 3.1.

Table 3.1: The cost on links

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
3.3 PROBLEM FORMULAT

Total cost is the sum of cost of individual links (delay) $e_1 e_2 e_3 \ldots e_m$ in the path.

$$\text{cost} = \sum_{i=1}^{m} e_m$$  \hspace{1cm} (3.1)

Objective function = \( \min \sum_{i=1}^{m} e_m \), source \( \rightarrow \) destination

Where \( i = 1, 2, 3, \ldots m \)  \hspace{1cm} (3.2)

The fitness of the chromosome is evaluated as:

$$\text{Fitness} = \text{HopsNo} \times \text{Pop} – \text{TotalCost}$$  \hspace{1cm} (3.3)

Where \( \text{HopsNo} \), represents number of hops in paths, it defines the number of intermediate nodes visited along the path from source node to destination node.

\( \text{Pop} \), Represents the initial population.

\( \text{TotalCost} \) It’s the total cost of path (Kumar, 2010).

3.4 PROPOSED SOLUTION

This section present the GA-Based solution, which guided by the mathematical model. In genetic algorithm a chromosome representation is required to describe each individual in the population and to determine how to structure the given problem.

3.4.1 Chromosome Representation

Before a genetic algorithm can be put to work on any problem, a method is needed to encode potential solutions to that problem in a form that a computer can process.
The coding of an individual (path) is composed by m strings $e_1 e_2 e_3 \ldots e_m$ each $e_i$ represents distance (length) between two nodes (Chaudhary, et al., 2011).

A path or route encoded as list of integers or string contains nodes from source to destination, the path may contain one or more intermediate nodes between them figure 3.3 show that.

| 1 | 3 | 4 | 6 |

Figure 3.2: The path representation

Each chromosome (path) contains set of genes. The genes are the nodes between the source node $n_0$ and destination node $u_i$ (Hamed, 2010). In this research the paths will be generated randomly with random length when generating the initial population, and then genetic operators will be used to generate the next generations.

The first gene in the path (chromosome) represents the source node (node1); the last one represents the destination node (node6), while other genes represent intermediate nodes (node3 and 4) along the path from the source to the destination. As show below:

| 1 | 3 | 4 | 6 |

Figure 3.3: Genes representation

### 3.4.2 Fitness Function

Use the fitness function to evaluate fitness for each chromosome (path); it’s a measure of the objective to obtain good solutions. The fitness function used in this work to minimize the cost from source to destination node.

1. First check if the chromosome is connected using intermediate nodes.
2. The second step if it’s from specific source to the destination (if not the path will be discarded) when it achieve the two mentioned condition then go and calculate the cost in equation 3.2. The fitness value of the paths is calculated, the path with the highest fitness value will have minimum cost.
3.4.3 Initialize Population

Once the problem was encoded in chromosomes with variable lengths and defined its fitness function, initialize the graph edges to generate possible paths from a given node to destination node in the network. Initially n random paths are considered (chromosome). This n defines the population size. These chromosomes act as population of the first generation and used GA operators to the next generations figure 33.4 show that. The edges is selected randomly and edges number also so that the chromosomes have variable length, which is the difference between the proposed work and the others mentioned in the related works.

| e3 | e5 | e2 | e4 | e1 |

Figure 3.4: Edges initialization

3.4.4 Selection (reproduction)

The selection operator is intended to improve the average quality of population, calling the selection process for each generation increases the algorithm complexity. (Mahmoud, 2007) In the proposed work tournament selection is used with elitism.

3.4.4.1 Tournament Selection

In tournament selection individuals with size 5 are chosen at random, the individuals are competes each other based on the fitness. Only one individual from each subgroup selected (with the best fitness) to join the next generation population. As shown in figure 33.5 the tournament size=5.
3.4.4.2 Elitism

The best individuals are copied to the new population and retain in each generation.

3.4.5 Crossover

Crossover combines two chromosomes (the parents) to produce two (one or more than two sometimes) new chromosomes (children). Made it in hope new chromosomes will include the good (materials) parts of the old one. Crossover probability (Pc) is a parameter to describe how often crossover will be performed.

3.4.5.1 Two point crossover:

Two crossover points are chosen and the contents between these points are exchanged between two mated parents.

Two strings are aligned, and two crossover points are selected uniformly at random along the length of the strings (Sivanandam, 2007).

Two point crossover is used within two selected paths randomly generated in variable length, the children produced has the same length of the longer path. Shown in figure 3.6 and figure 3.7. Consider two strings:
Figure 3.6: Two selected parents

The produced offspring:

Child 1

Child 2

Figure 3.7: The produced offspring

3.4.6 Mutation

Mutation process plays role of recovering the lost genetic materials and maintains the diversity in the population. The important parameter is the mutation probability (Pm) it decides how often parts of chromosome will be mutated. The path mutation applied to a randomly selected single solution (chromosome) from the population with a certain probability. Mutation makes small random changes in the solution, which will gradually add some characteristics to the population, which could not be supplied by the crossover operator (Mahmoud, 2007). Insertion mutation used in the proposed work; only one gene is selected to be displaced and inserted back into the chromosome. First select the chromosome for mutation figure 3.8, take node 2 out of sequence as shown in figure 3.9 and reinsert the 2 at a randomly chosen position figure 3.10 show that.
3.4.7 Replacement

After the new offspring solutions are created using crossover and mutation, next is to introduce them into the parental population, the replacement strategy is to replace the worst population member with the new offspring. The path 1 2 5 6 with cost 13 must be replaced with another with lower cost than it. Replaced with 1 3 4 6 with cost 8 as shown in figure 3.11 and figure 3.12.
### 3.4.8 Termination Condition

The termination condition is met either after a specified number of generations or no improvement occurs on the best solution for successive generations. In this implementation, the maximum fitness is used as termination condition to GA (Mahmoud, 2007).

The performance of GA is based on efficient representation, evaluation of fitness function and other parameters like the population size, rate of crossover, mutation and the strength of selection (Nair, et al., 2010).

![Figure 3.13: The genetic algorithm flow chart](image)
The proposed study shown in figure 3.13 the used genetic operators are tournament selection with elitism techniques, multipoint crossover and insertion mutation.
CHAPTER FOUR

Results and Discussion

This chapter reviews the experimental setup and network examples topologies.

4.1 EXPERIMENTAL SETUP

The computer properties are processor Intel(R) Celeron(R) CPU B830@ 180 GHz, 2.00 GB RAM and 32 Operating System. Net Beans IDE 8.1 and MS excel 2010 used to implement the proposed genetic algorithm and used to draw the flow chart of the result. The algorithm takes the population size, crossover probability ($P_c$), mutation probability ($P_m$), the type of selection operation, number of generations, uniform rate, and network topology. The cost randomly generated between 1 and 30 and assigned to each link as delay. The algorithm has implemented in Java.

The experiment is carried in eight steps as explained below:

Step 1: Initialize the graph and generate random edges with random number each edge contains the source, destination and its cost.

Step 2: Evaluate edges with the objective function:

1. Connected graph by connected edges, if the fitness is equal to zero; means that the nodes are not connected to each other.

2. From given source to destination

3. The path with the minimum cost will be selected.

Step 3: The selection operation takes place, using tournament selection method to select individuals to be used as parents and the tournament size 5.

Step 4: Perform crossover on the selected paths (parents) to produce new children with crossover probability, using the multipoint crossover.

Step 5: Mutation operator will be used for the crossover chromosomes by selecting random path, pick a random node and insert it at random position. Mutation will be done according to the mutation probability.

Step 6: Apply mutation on resulting crossover with mutation probability $p_m = 0.015$, mutation insert one gene in the chromosome randomly.

Step 7: Replacing the worst member of the population with the new offspring population which created by the selection, the crossover, and the mutation operation.
Step 8: Repeat the above step until some condition (e.g. less delay solution) is satisfied.

4.1.1 Instance 1

Consider that a network with 6 nodes (vertices) and 8 edges. Each link has a non-negative cost, population size =15, uniform rate =0.5, mutation rate=0.015, tournament selection size=5 with elitism, multipoint crossover and insertion mutation with 5000 maximum generations, consider node 1 as source and node 6 as the destination the cost generated at random between 1 to 7. Table 4.1 show parameters setting, figure 4.2 genetic-based solution, figure 4.3 Instance 1 obtained solution and figure 4.4 presents convergence rate.

Figure 4.1 Example of network topology.
Table 4.1: Instance 1 parameters settings

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers of nodes</td>
<td>6</td>
</tr>
<tr>
<td>Population size</td>
<td>8</td>
</tr>
<tr>
<td>Edge number</td>
<td>15</td>
</tr>
<tr>
<td>Uniform rate</td>
<td>0.5</td>
</tr>
<tr>
<td>Mutation rate</td>
<td>0.015</td>
</tr>
<tr>
<td>Tournament size</td>
<td>5</td>
</tr>
<tr>
<td>Elitism</td>
<td>True</td>
</tr>
<tr>
<td>Source node</td>
<td>1</td>
</tr>
<tr>
<td>Destination node</td>
<td>6</td>
</tr>
<tr>
<td>Maximum generation</td>
<td>5000</td>
</tr>
</tbody>
</table>

Figure 4.2: Genetic-based solution.

Figure 4.3: Instance 1 obtained solution
The algorithm successfully determine the shortest path $1 \ 3 \ 4 \ 6$.

![Figure 4.4: Instance 1 fitness values](image)

The results show that the GA find path with minimum cost $1 \ 3 \ 4 \ 6$ from node 1 to the destination node 6 with cost equals 8 and the algorithm convergence to the solution.

### 4.1.2 Instance 2

Having graph2 contains 5 nodes (vertices) and 7 edges, population size =15, uniform rate =0.5, mutation rate=0.015, tournament selection size=5 with elitism, multipoint crossover and insertion mutation with 5000 maximum generations, consider node 1 as source and node 6 as the destination the cost generated at random between 1 to 6. Below there are two tables represent parameters setting and the possible solutions for the given instance.
Figure 4.5: Example for Network topology

Table 4.2: Instance 2 parameters settings

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers of nodes</td>
<td>5</td>
</tr>
<tr>
<td>Edges number</td>
<td>7</td>
</tr>
<tr>
<td>Population size</td>
<td>15</td>
</tr>
<tr>
<td>Uniform rate</td>
<td>0.5</td>
</tr>
<tr>
<td>Mutation rate</td>
<td>0.015</td>
</tr>
<tr>
<td>Tournament size</td>
<td>5</td>
</tr>
<tr>
<td>Elitism</td>
<td>True</td>
</tr>
<tr>
<td>Source node</td>
<td>1</td>
</tr>
<tr>
<td>Destination node</td>
<td>5</td>
</tr>
<tr>
<td>Maximum generation</td>
<td>5000</td>
</tr>
</tbody>
</table>
Table 4.3: The possible paths instance 2

<table>
<thead>
<tr>
<th>Path</th>
<th>cost</th>
<th>Connected edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1 2 4 5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>1 3 5</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>1 2 3 5</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 4.6: Genetic algorithm-based solution.

The algorithm successfully determine the shortest path 1 2 3 4 6 with cost 4.

Figure 4.7: GA solution instance2.
4.1.3 Instance 3

Consider a network contains 30 nodes (vertices) and 39 edges, population size =150, uniform rate =0.5, mutation rate=0.015, tournament selection size=5 with elitism, multipoint crossover and insertion mutation with 5000 maximum generations, considered node 1 as source and node 30 as the destination the cost generated at random between 1 to 30. The parameters settings are shown below in Table 44.4.
Figure 4.9: Example for Network topology

Table 4.4 : Instance 3 parameters settings

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers of nodes</td>
<td>30</td>
</tr>
<tr>
<td>Edges number</td>
<td>39</td>
</tr>
<tr>
<td>Population size</td>
<td>150</td>
</tr>
<tr>
<td>Uniform rate</td>
<td>0.5</td>
</tr>
<tr>
<td>Mutation rate</td>
<td>0.015</td>
</tr>
<tr>
<td>Tournament size</td>
<td>5</td>
</tr>
<tr>
<td>Elitism</td>
<td>True</td>
</tr>
<tr>
<td>Source node</td>
<td>1</td>
</tr>
<tr>
<td>Destination node</td>
<td>30</td>
</tr>
<tr>
<td>Maximum generations</td>
<td>5000</td>
</tr>
</tbody>
</table>
The algorithm successfully determine the shortest path 1 8 10 14 15 24 23 25 19 26 20 28 29 30 with cost equal 13.

Figure 4.10: The result of the algorithm

Figure 4.11: GA solution for given network
The GA finds the path with minimum cost from node 1 to the destination node 30 with cost equals 13 and the algorithm convergence to the solution as shown in figure 4.12.

4.2 RESULT DISCUSSION

The results show that the proposed algorithm has ability to find path with minimum cost from a single source node to the destination node for any given network instances of different sizes (5, 6, and 30) in reasonable time. In all case studies, the algorithm was successful in determining the shortest path. The convergence was guaranteed to obtain the optimal path in each case.
CHAPTER FIVE
CONCLUSION AND RECOMMENDATION

5.1 CONCLUSION
In this research the SPP studied, the problem has been formulated as a mathematical model then a genetic based algorithm has been proposed to solve the research problem guided by the mathematical model.
The proposed algorithm has been tested using different network instances, and genetic operators (crossover, mutation and population size).

5.2 RECOMMENDATION
As future work the recommendation is to combine the GA with simulated annealing algorithm for mutation to get faster diversity in larger network sizes.
REFERENCES


APPENDIX

The appendix code contains six classes. The first class an edge class.

```java
package treeinjava;

public class edge {
    int cost;
    String source, destination;

    edge(String s, String d, int c) {
        cost = c;
        source = s;
        destination = d;
    }

    @Override
    public String toString() {
        return source + " " + destination + " " + cost;
    }
}
```

The second class is sharedInfo contains variables shared to all package.

```java
package treeinjava;

import java.util.ArrayList;
import java.util.List;

public class sharedInfo {
    static List<edge> baseEdges = new ArrayList<>();
    static String Source = "D";
    static String Destination = "I";
    static int totalCost = 0;

    static void init_Graph() {
        // Code for init_Graph
    }
}
```
// graph 1

/*
baseEdges.add(new edge ("1","2",5));
baseEdges.add(new edge ("1","3",3));
baseEdges.add(new edge ("1","4",7));
baseEdges.add(new edge ("2","4",3));
baseEdges.add(new edge ("3","4",3));
baseEdges.add(new edge ("2","5",5));
baseEdges.add(new edge ("4","6",2));
baseEdges.add(new edge ("5","6",3));
Source = "1";
Destination = "6";
*/

// graph 2

/*
baseEdges.add(new edge ("1","2",1));
baseEdges.add(new edge ("1","3",5));
baseEdges.add(new edge ("2","4",3));
baseEdges.add(new edge ("3","4",1));
baseEdges.add(new edge ("2","3",1));
baseEdges.add(new edge ("4","5",1));
baseEdges.add(new edge ("3","5",6));
Source = "1";
Destination = "5";
*/

// graph 3

/*
baseEdges.add(new edge ("1","2",10));
baseEdges.add(new edge ("1","8",1));
baseEdges.add(new edge ("1","9",20));
baseEdges.add(new edge ("2","3",20));
baseEdges.add(new edge ("3","4",10));
baseEdges.add(new edge ("3","8",20));
baseEdges.add(new edge ("3","5",10));
baseEdges.add(new edge ("4","6",20));
baseEdges.add(new edge ("5","7",10));
*/
baseEdges.add(new edge ("5","12",20));
baseEdges.add(new edge ("6","5",10));
baseEdges.add(new edge ("6","17",20));
baseEdges.add(new edge ("7","10",20));
baseEdges.add(new edge ("8","7",20));
baseEdges.add(new edge ("8","10",1));
baseEdges.add(new edge ("9","13",10));
baseEdges.add(new edge ("10","14",1));
baseEdges.add(new edge ("13","14",10));
baseEdges.add(new edge ("14","15",1));
baseEdges.add(new edge ("12","11",30));
baseEdges.add(new edge ("11","14",20));
baseEdges.add(new edge ("11","16",10));
baseEdges.add(new edge ("17","16",30));
baseEdges.add(new edge ("17","22",10));
baseEdges.add(new edge ("14","19",20));
baseEdges.add(new edge ("15","24",1));
baseEdges.add(new edge ("24","23",1));
baseEdges.add(new edge ("23","25",1));
baseEdges.add(new edge ("25","19",1));
baseEdges.add(new edge ("19","26",1));
baseEdges.add(new edge ("16","18",30));
baseEdges.add(new edge ("22","18",20));
baseEdges.add(new edge ("22","27",30));
baseEdges.add(new edge ("18","20",30));
baseEdges.add(new edge ("26","20",1));
baseEdges.add(new edge ("20","28",1));
baseEdges.add(new edge ("27","28",10));
Source = "1";
Destination = "28";
*/
baseEdges.stream().forEach((baseEdge) -> {
    totalCost += baseEdge.cost;
});
public static String baseEdgesString()
{
    String listAsString = "";
    for(int i = 0;i<=baseEdges.size()-1;i++)
    {
        listAsString += baseEdges.get(i).toString() + ";
    }
    return listAsString;
}

---

The third class is an individual class.
package treeinjava;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
public class indv {
    Random rand;
    public List <edge> edges =new ArrayList<>();
    int cont =0;
    int fromSourceToDestination;
    int totalCost = 0;
    int nn;
    String path = "";
    float fit = 0;
    public edge getGene(int index)
    {
        return edges.get(index);
    }
    public void setGene(int index,edge value)
    {
    }
edges.set(index, value);
}
public int size()
{
    return edges.size();
}
indv(int i)
{
}
indv()
{
    rand = new Random();
    int n = rand.nextInt(sharedInfo.baseEdges.size()) + 1;
    for (int i = 0; i <= n; i++)
    {
        edges.add(sharedInfo.baseEdges.get(rand.nextInt(n)));
    }
    nn = n;
    rand = null;
}
public float getFitness()
{
    //if (fit == 0)
    //{
    //    fillfitness();
    //}
    return fit;
}
void fillfitness()
{
    int indexOfRoot = -1;
    for (int i = 0; i <= edges.size() - 1; i++)
    {
        edge ed = edges.get(i);
if (ed.source.equals(sharedInfo.Source))
    indexOfRoot = i;
}
if (indexOfRoot == -1)
{
    return;
}
indexOfRoot--;
String Des = getIndexOf(sharedInfo.Source);
String Last = "";
String Src;
int costis;
costis = getCost(sharedInfo.Source,Des);
cont = 0;
path = sharedInfo.Source + "--" + costis + "-->" + Des + " , ";
while (!Des.equals("-1"))
{
    cont++;
    Src = Des;
    Des = getIndexOf(Des);
    //costis += getCost(Src,Des);
    if (!Des.equals("-1"))
    {
        Last = Des;
        int localconst = getCost(Src,Des);
        costis += localconst;
        path += Src + "--" + localconst + "-->" + Des + " , ";
        //path += Src + " " + Des + " " + localconst + ",";
    }
}
//if (Last.equals(sharedInfo.Destination))
fromSourceToDestination = 0;
totalCost = 0;
if (path.endsWith(sharedInfo.Destination+"","*))
{
    fromSourceToDestination = 100;
    totalCost = costis;
}

//fit = (cont * 10) + (fromSourceToDestination * 10 * 
sharedInfo.baseEdges.size()) + ((sharedInfo.totalCost - totalCost) * 10 ) ;
    fit = 0;
    if (fromSourceToDestination == 100)
    {
        fit = (sharedInfo.totalCost - totalCost) * 10;
    }
}
}
int getCost(String s,String d)
{
    for(int i = 0;i<=edges.size()-1;i++)
    {
        edge ed = edges.get(i);
        if (ed.source.equals(s) && ed.destination.equals(d))
            return ed.cost;
    }
    return 0;
}
String getIndexOf(String source)
{
    for(int i = 0;i<=edges.size()-1;i++)
    {
        edge ed = edges.get(i);
        if (ed.source.equals(source))
            return ed.destination;
    }
return ":1";
@Override
public String toString()
{
    String listAsString = "";
    /*for(int i = 0;i<=edges.size()-1;i++)
    {
        listAsString += edges.get(i).toString() + 
    } */
    String fstd = "False";
    if (fromSourceToDestination == 100)
    {
        fstd = "True";
    }
    if (path.endsWith(", "))
        path = path.substring(0,path.length()-2);
    listAsString += "Connected Edges: " + cont + 
        "\nFrom Source To Destination: "+ fstd + 
        "\nTotal Cost: "+totalCost + 
        //"\nNumber Of All Generated Edges In This Gene: "+nn+ 
        "\nPath:"
    return listAsString;
}

public String ShortString()
{
    return "t:"+totalCost+"f:"+fit+"p:"+path;
}
}
Class number four is the population class.
	package treeinjava;
	public class Population {
		indv[] individuals;
		// Create a population
	public Population(int populationSize, boolean initialise) {
			individuals = new indv[populationSize];
			// Initialise population
			if (initialise) {
				// Loop and create individuals
				for (int i = 0; i < size(); i++) {
					indv newIndividual = new indv();
					newIndividual.fillfitness();
					saveIndividual(i, newIndividual);
				}
			}
		}
	public void saveIndividual(int index, indv ind) {
			individuals[index] = ind;
		}
	public indv getIndividual(int index){
			return individuals[index];
		}
	public indv getFittest(){
			indv fittest = individuals[0];
			fittest.fillfitness();
			for(int i = 0 ; i < size() ; i++) {
				getIndividual(i).fillfitness();
				if (fittest.getFitness() <= getIndividual(i).getFitness())
				}
public int size() {
    return individuals.length;
}

------------------------------------------------------------------
Class number five is the algorithm class and its operations.
package treeinjava;
import java.util.Random;
public class Algorithm {

    private static final double uniformRate = 0.5;
    private static final double mutationRate = 0.015; // 0.5;
    private static final int tournamentSize = 5;
    private static final boolean elitism = true;

    /* Public methods */

    // Evolve a population
    public static Population evolvePopulation(Population pop) {
        Population newPopulation = new Population(pop.size(), false);
        /*System.out.println("Current Population: =====================");
        for (int i = 0; i < pop.size(); i++) {
            System.out.println(i + ": " + pop.individuals[i].ShortString());
        }
        System.out.println("**Fittest:" + pop.getFittest().ShortString());
        */
        return newPopulation;
    }
}
// Keep our best individual
if (elitism) {
    newPopulation.saveIndividual(0, pop.getFittest());
}/*
System.out.println("+++++++++++++++++++++++++++++++");
System.out.println(newPopulation.getIndividual(0).ShortString());
System.out.println("+++++++++++++++++++++++++++++++");*/

// Crossover population
int elitismOffset;
if (elitism) {
    elitismOffset = 1;
} else {
    elitismOffset = 0;
}

// Loop over the population size and create new individuals with crossover
for (int i = elitismOffset; i < pop.size(); i++) {
    indv indiv1 = tournamentSelection(pop);
    indv indiv2 = tournamentSelection(pop);
    indiv1.fillfitness();
    indiv2.fillfitness();
    indv newIndiv = crossover(indiv1, indiv2);
    newIndiv.fillfitness();
    newPopulation.saveIndividual(i, newIndiv);
    //System.out.println(i + ": Cross 1:" + indiv1.ShortString() + " 2:" + indiv2.ShortString() + " == " + newIndiv.ShortString());
}

/*System.out.println("New Population: xxxxxxxxxxxxxxxxxxxxxxxxxxxxx");
for (int i = 0; i < newPopulation.size(); i++) {
    System.out.println(i + ": " + newPopulation.individuals[i].ShortString());
}*/
System.out.println("**Fittest:" + newPopulation.getFittest().ShortString());
System.out.println("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx");

// Mutate population
for (int i = elitismOffset; i < newPopulation.size(); i++) {
    mutate(newPopulation.getIndividual(i));
}

/*System.out.println("New Population: oooooooooooooooooooooooooooooo");
for (int i = 0; i < newPopulation.size(); i++) {
    System.out.println(i+":" + newPopulation.individuals[i].ShortString());
}
System.out.println("**Fittest:" + newPopulation.getFittest().ShortString());
System.out.println("oooooooooooooooooooooooooooooooooooooooooo");*/
return newPopulation;

// Crossover individuals
private static indv crossover(indv indiv1, indv indiv2) {
    indv newSol;
    indv shortone;
    if (indiv1.size() > indiv2.size())
    {
        newSol = new indv(1);
        for (int i= 0;i < indiv1.size();i++)
        {
            newSol.edges.add(indiv1.getGene(i));
        }
        shortone = indiv2;
    }
    else
    { ...
else
{
    newSol = new indv(1);
    for (int i= 0;i < indiv2.size();i++)
    {
        newSol.edges.add(indiv2.getGene(i));
    }
    shortone = indiv1;
}

// Loop through genes
for (int i = 0; i < shortone.size(); i++) {
    // Crossover
    //if (Math.random() <= uniformRate) {
    if (i % 2 == 0) {
        newSol.setGene(i, shortone.getGene(i));
    }
    }
    return newSol;
}

// Mutate an individual
private static void mutate(indv indiv) {
    // Loop through genes
    for (int i = 0; i < indiv.size(); i++) {
        if (Math.random() <= mutationRate) {
            // Create random gene
            //int gene = (int) Math.round(Math.random());
            Random rand = new Random();
            int n = rand.nextInt(sharedInfo.baseEdges.size()) + 1;
            indiv.setGene(i, sharedInfo.baseEdges.get(rand.nextInt(n)));
            indiv.fillfitness();
        }
    }
}
private static indv tournamentSelection(Population pop) {
    // Create a tournament population
    Population tournament = new Population(tournamentSize, false);
    // For each place in the tournament get a random individual
    for (int i = 0; i < tournamentSize; i++) {
        int randomId = (int) (Math.random() * pop.size());
        tournament.saveIndividual(i, pop.getIndividual(randomId));
    }
    // Get the fittest
    indv fittest = tournament.getFittest();
    return fittest;
}
for (int i = 0; i < fitHistory.length-1;i++)
{
    fitHistory[0] = 0;
}
/*Random rand = new Random();
for (int i = 0; i < 50;i++)
{
    int n = rand.nextInt(sharedInfo.baseEdges.size()) + 1;
    System.out.println(n);
}*/
/*
System.out.println(sharedInfo.baseEdgesString());
indv iv ;
int all = 0;
for (int i = 0; i < 50;i++)
{
    iv = new indv();
    iv.fillfitness();
    if (iv.fromSourceToDestination != 100) continue;
    System.out.println("indv: "+(i+1)+"==");
    System.out.println(iv.toString());
    all++;
}
System.out.println("all:"+all);
*/
Population myPop = new Population(25,true);

int generationCount = 0;
int maxGenerationCount = 500;
while(checkcontinueCondition() && maxGenerationCount > generationCount){
    //while( maxGenerationCount > generationCount){
        generationCount++;
    myPop.getFittest().fillfitness();
fitHistory[generationCount % fitHistory.length] = myPop.getFittest().getFitness();
System.out.println("Generation: "+generationCount+" Fittest: "+myPop.getFittest().getFitness());
myPop = Algorithm.evolvePopulation(myPop);
}
if (myPop.getFittest().fit != 0)
{
    System.out.println("Solution found!");
    System.out.println("Generation: "+generationCount);
    System.out.println("Genes:");
    System.out.println(myPop.getFittest().toString());
}
else
{
    System.out.println("Solution not found!");
}
}
static boolean checkContinueCondition()
{
    float curr = fitHistory[0];
    if (curr == 0) return true;
    for (int i = 0; i < fitHistory.length -1;i++)
    {
        if (fitHistory[i] != curr)
            return true;
    }
    return false;
}