



**Project Report on**

# **Fourth Party Logistics Routing Model with Fuzzy Duration Time**

---

Submitted in partial fulfillment of the requirement for the award of the degree of

**BACHELOR OF TECHNOLOGY**

**IN**

**PRODUCTION AND INDUSTRIAL ENGINEERING**

**AT**

**INDIAN INSTITUTE OF TECHNOLOGY ROORKEE**

*Submitted by:*

Puneet Pandey (11119036)

Vikalp Aggarwal (11119046)

*Guided by:*

Dr. Akshay Dvivedi

Assistant Professor,  
Department of Mechanical  
and Industrial Engineering,  
IIT Roorkee

## TABLE OF CONTENTS

|  |           |
|--|-----------|
| <b>1. Introduction</b>                           | <b>8</b>  |
| <b>1.1. Current Scenario</b>                     | <b>8</b>  |
| <b>1.2.Scope of the Project</b>                  | <b>9</b>  |
| <b>2. Literature Review and Analysis</b>         | <b>10</b> |
| <b>2.1.Logistics</b>                             | <b>10</b> |
| <b>2.2.Third Party Logistics</b>                 | <b>10</b> |
| <b>2.3.Fourth Party Logistics</b>                | <b>12</b> |
| <b>2.4.Fuzzy Models</b>                          | <b>13</b> |
| <b>2.5.K-th Shortest Path Routing</b>            | <b>14</b> |
| <b>2.6.Generic Algorithm</b>                     | <b>15</b> |
| <b>3. Methodology</b>                            | <b>17</b> |
| <b>3.1.Explanation of equations</b>              | <b>20</b> |
| <b>4. Algorithm designed for the 4PLRPF</b>      | <b>23</b> |
| <b>4.1.Flow Chart and steps for algorithm</b>    | <b>25</b> |
| <b>4.2.Two step Genetic Algorithm</b>            | <b>26</b> |
| <b>4.3.The K-th Shortest Path Algorithm</b>      | <b>29</b> |
| <b>4.4.Fitness Function</b>                      | <b>32</b> |
| <b>4.5.Selection Mechanism</b>                   | <b>33</b> |
| <b>4.6.Crossover Operation</b>                   | <b>34</b> |
| <b>4.7.Mutation Operation</b>                    | <b>37</b> |
| <b>4.8.Second Step GA used for 3PL suppliers</b> | <b>37</b> |
| <b>5. Discussion &amp; Future</b>                | <b>52</b> |
| <b>6. Conclusion</b>                             | <b>53</b> |

## **DECLARATION BY THE CANDIDATES**

We hereby clarify that the work being presented here, entitled “Fourth Party Logistics Routing Model with Fuzzy Duration Time”, is an authentic record of our own work from the period August 2014 till May 2015 under the guidance of Dr. Akshay Dvivedi, Assistant Professor, Department of Mechanical and Industrial Engineering, Indian Institute of Technology Roorkee. The matter embodied in this report to the best of our knowledge has never been presented for the award of any other degree elsewhere.

DATE:

.....

Puneet Pandey

.....

Vikalp Aggarwal

## **CERTIFICATE**

This is to certify that the report submitted by Puneet Pandey and Vikalp Aggarwal on “Fourth Party Logistics Routing Model with Fuzzy Duration Time” in the partial fulfillment of the degree is an authentic record of the project work which they have satisfactorily completed under my supervision.

.....

Dr. Akshay Dvivedi

DATE:

## **ACKNOWLEDGEMENT**

We would like to express our sincere thanks and gratitude to Dr. Akshay Dvivedi for his encouragement, over-hearted support and guidance which has been instrumental in the successful completion of the project in the allotted time. We are also deeply indebted to him for his valuable insights and directions throughout the past one year.

The comments and suggestions of the evaluation committee have been a constructive feedback to our work. For the active participation and positive input, we extend our special thanks to Dr. Navneet Arora, Dr. Pradeep K. Jha, and Dr. Kaushik Pal.

.....

Puneet Pandey

.....

Vikalp Aggarwal

## TABLE OF FIGURES

| <b>S.No.</b> | <b>Name of Figure</b>   | <b>Page number</b> |
|--------------|---|--------------------|
| 1            | Layers to Logistic services   | <b>12</b>          |
| 2            | Multi-graph of a 4PLRPF   | <b>17</b>          |
| 3            | Flowchart for the fuzzy simulation based KTGA   | <b>25</b>          |
| 4            | Flowchart for the initialization process  | <b>26</b>          |
| 5            | Linkage matrix B representing multi-graph   | <b>27</b>          |
| 6            | The process of converting a feasible route  | <b>28</b>          |
| 7            | Schematic explanation of Crossover operation, circle represents node (city) and letter in square box represents name of the 3PL supplier used | <b>36</b>          |

## **ABSTRACT**

The importance of supply chain has been enhanced in the era of 21<sup>st</sup> century because of fourth party as it integrates the whole chain as delivers a macroscopic view to the business leaders. The uncertainty involved in various parameters such as availability of 3<sup>rd</sup> party suppliers (3PL), cost of transportation, time required for transportation makes the analysis of this macroscopic view of the supply chain difficult and quite complicated. In this project, a routing problem is solved for calculating the path with minimum cost, and more importantly, some constraints are considered in fuzzy set to include the uncertainty involved. The experimental results obtained in the end of this project clearly show the value of the model we have developed for 4PLRPF (4<sup>th</sup> party logistics routing problem with fuzzy duration time). The algorithm used is a two-step generic algorithm and the constraints of trans-shipment problem have been appropriately tailored into the problem with clear and logical assumptions.

## 1. INTRODUCTION:

*“There is only one boss - The customer. And he can fire everybody in the company from the chairman on down, simply by spending his money somewhere else.”*

### 1.1 CURRENT SENARIO

As the businesses have expanded throughout the globe, demand for logistic suppliers has shown a sharp spike in recent times. It has been observed that seventy-five percent of Fortune 500 companies use 3<sup>rd</sup> party logistic providers. 3PL suppliers usually provide facility of warehousing, inter-and-intra-city transportation, hence including them becomes imperative for running the business. This gives rise to the 4<sup>th</sup> party logistic (4PL) providers. The 4PL providers manage and oversee all the major functioning of the 3PL suppliers; they have created a need of mutual understanding and business co-operation that has benefitted both the customers and the businesses. A problem of uncertainty arises when fourth party is used; many factors like transportation time between the nodes or cities, processing time at the nodes, cost of transportation are not crisp values. This makes the 4PL problem a multi-graph approach, while 3PL situations could be modeled as a single-graph, thereby attracting a lot of research interest from every corner of the world. The research done so far included fixed conditions and they were not able to mimic the real life conditions which are necessarily fuzzy. This project considers the fuzzy set of constraints; the algorithm developed henceforth can be successfully applied to the real world practical problems.

In 4PLRPF, two nodes have more than one 3PL provider between them. So it is a multi-graph problem with fuzzy parameters, and thus, can be considered as a constrained shortest-path problem. We have used two-step genetic algorithm to find the route of minimum cost.



## **1.2 SCOPE OF THE PROJECT**

In the current business landscape of India Inc. the competition among the e-commerce businesses is fiercer than ever before. India has been identified as a big consumer market for selling goods online and hence, has now emerged as an untapped business opportunity. Therefore, a lot of companies ranging from multinational business giants to small homegrown and garage startups are honing their operations to grab this opportunity.

The model developed in this project is an attempt to improve the logistics operations of the business. This model can be directly implemented into the market to get an extra edge over the competitors. The parameters included in the model are very realistic and can be customized for usage. The generic algorithm has been appropriately transformed into optimizing the flow of goods from starting node – the seller - to the ending node – the customer. Generic algorithm has been chosen over others for a very intuitive reason: it is that algorithm that nature has inducted in itself for optimization of activities. Through Darwin's Theory of Evolution, this algorithm is unarguably one of the best for optimization of similar problem. Ants use the genetic algorithm to transport food in the walls of their colonies. DNA strands evolving for the off-spring use the same algorithm. Similar methods if used in the transportation of delivery packages will surely be more efficient and less time consuming and costly.

Hence, the project has tremendous business and research scopes.

## 2. LITERATURE REVIEW AND ANALYSIS

Following are few key words that must be clarified before understanding the model-

**2.1 Logistics:** Logistics is the management of the flow of goods between the point of origin and the point of consumption in order to meet some requirements, of customers or corporations. The resources managed in logistics can include physical items, such as food, materials, animals, equipment and liquids, as well as abstract items, such as time, information, particles, and energy. The logistics of physical items usually involves the integration of information flow, material handling, production, packaging, inventory, transportation, warehousing, and often security.

The complexity of logistics can be modeled, analyzed, visualized, and optimized by dedicated simulation software. The minimization of the use of resources is a common motivation in logistics for import and export.

**2.2 Third-party Logistics:** A third-party logistics provider (abbreviated 3PL, or sometimes TPL) is a firm that provides service to its customers of outsourced (or "third party") logistics services for part, or all of their supply chain management functions. Third party logistics providers typically specialize in integrated operation, warehousing and transportation services that can be scaled and customized to customers' needs based on market conditions, such as the demands and delivery service requirements for their products and materials. Often, these services go beyond logistics and include value-added services related to the production or procurement of goods, i.e., services that integrate parts of the supply chain. When this integration occurs, the provider is then called a third-party supply chain management provider (3PSCM) or supply chain management service

provider (SCMSP). 3PL targets a particular function in supply management, such as warehousing, transportation, or raw material provision.

There are four categories of 3PL providers

**2.2.1 Standard 3PL Provider:** this is the most basic form of a 3PL provider. They would perform activities such as, pick and pack, warehousing, and distribution (business) – the most basic functions of logistics. For a majority of these firms, the 3PL function is not their main activity.

**2.2.2 Service Developer:** this type of 3PL provider will offer their customers advanced value-added services such as: tracking and tracing, cross-docking, specific packaging, or providing a unique security system. A solid IT foundation and a focus on economies of scale and scope will enable this type of 3PL provider to perform these types of tasks.

**2.2.3 The Customer Adapter:** this type of 3PL provider comes in at the request of the customer and essentially takes over complete control of the company's logistics activities. The 3PL provider improves the logistics dramatically, but does not develop a new service. The customer base for this type of 3PL provider is typically quite small.

**2.2.4 The Customer Developer:** this is the highest level that a 3PL provider can attain with respect to its processes and activities. This occurs when the 3PL provider integrates itself with the customer and takes over their entire logistics function. These providers will have few customers, but will perform extensive and detailed tasks for them.[7]

### 2.3 Fourth-Party Logistics(4PL):

Information technology plays a key role in logistics and supply chain management. In fact logistics integration, which is a complex exercise, is completely dependent on IT support. Third party logistic suppliers provide logistics solutions to clients on the basis of their

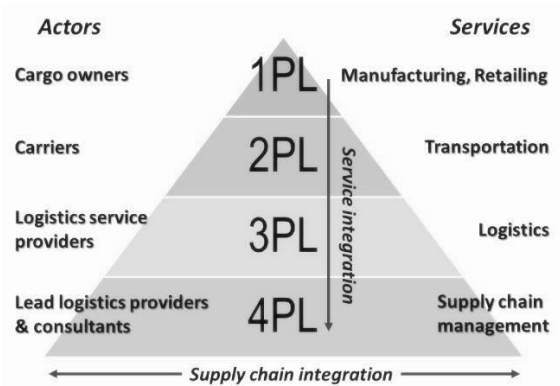


Figure 1: Layers to logistic services

domain knowledge they have acquired over the years. 4 PL companies provide logistics solutions built around the domain knowledge provided by third party logistics companies. Thus 4 PLs have emerged out of the vacuum created by 3PLs.

Fourth Party Logistics (4PL) is the integration of all companies involved along the supply chain. 4PL is the planning, steering and controlling of all logistic procedures (for example flow of information, material and capital) by one service provider with long term strategic objectives. Fourth party logistics (4PL) has evolved as a breakthrough supply chain solution comprehensively integrating the competencies of third party logistics (3PL) providers, leading edge consulting firms and technology providers

4PLs see the process and what is required for the process to succeed. A 4PL is a supply chain manager & enabler who assembles and manages resources, build capabilities and technology with those of complimentary service providers. They act as the first point for delivering unique and comprehensive supply chain solutions. 4PL leverages combined capabilities of management consulting and 3PLs. They act as an integrator assembling

the resources, capabilities, and technology of their own organization and other organizations to design, build and run comprehensive supply chain solutions. 4 PL is an emerging trend and it is a complex model and offers greater benefits in terms of economies of scale.

**2.4 Fuzzy Logic:** Fuzzy logic is a form of many-valued logic that deals with approximate, rather than fixed and exact reasoning. Compared to traditional binary logic (where variables may take on true or false values), fuzzy logic variables may have a truth value that ranges in degree between 0 and 1. Fuzzy logic has been extended to handle the concept of partial truth, where the truth value may range between completely true and completely false. Furthermore, when linguistic variables are used, these degrees may be managed by specific functions.

In recent years, the number and variety of applications of fuzzy logic have increased significantly. The applications range from consumer products such as cameras, camcorders, washing machines, and microwave ovens to industrial process control, medical instrumentation, decision-support systems, and portfolio selection.

Fuzzy logic has two different meanings. In a narrow sense, fuzzy logic is a logical system, which is an extension of multivalued logic. However, in a wider sense fuzzy logic (FL) is almost synonymous with the theory of fuzzy sets, a theory which relates to classes of objects with un-sharp boundaries in which membership is a matter of degree. In this perspective, fuzzy logic in its narrow sense is a branch of FL. Even in its more narrow definition, fuzzy logic differs both in concept and substance from traditional multivalued logical systems.

In Fuzzy Logic Toolbox™ software, fuzzy logic should be interpreted as FL, that is, fuzzy logic in its wide sense. The basic ideas underlying FL are explained very clearly and insightfully in Foundations of Fuzzy Logic. What might be added is that the basic concept underlying FL is that of a linguistic variable, that is, a variable whose values are words rather than numbers. In effect, much of FL may be viewed as a methodology for computing with words rather than numbers. Although words are inherently less precise than numbers, their use is closer to human intuition. Furthermore, computing with words exploits the tolerance for imprecision and thereby lowers the cost of solution.

Another basic concept in FL, which plays a central role in most of its applications, is that of a fuzzy if-then rule or, simply, fuzzy rule. Although rule-based systems have a long history of use in Artificial Intelligence (AI), what is missing in such systems is a mechanism for dealing with fuzzy consequents and fuzzy antecedents. In fuzzy logic, this mechanism is provided by the calculus of fuzzy rules. The calculus of fuzzy rules serves as a basis for what might be called the Fuzzy Dependency and Command Language (FDCL). Although FDCL is not used explicitly in the toolbox, it is effectively one of its principal constituents. In most of the applications of fuzzy logic, a fuzzy logic solution is, in reality, a translation of a human solution into FDCL.

**2.5 *K-th shortest path routing:*** The K shortest path routing algorithm is an extension algorithm of the shortest path routing algorithm in a given network.

It is sometimes crucial to have more than one path between two nodes in a given network. In the event there are additional constraints, other paths different from the

shortest path can be computed. To find the shortest path one can use shortest path algorithms and extend them to find more than one path. The K Shortest path routing algorithm is a generalization of the shortest path problem. The algorithm not only finds the shortest path, but also K other paths in order of increasing cost. K is the number of shortest paths to find. The problem can be restricted to have the K shortest path without loops (loop less  $K^{\text{th}}$  shortest path) or with loop.

There are two main variations of the  $K^{\text{th}}$  Shortest path routing algorithm. Other variations only fall in between these.

- 1) In the first variation, loops are allowed, that is paths are allowed to revisit the same node multiple times. The following papers deal with this variation.
- 2) The second variation deals with simple paths. It is also called loop less  $K^{\text{th}}$  Shortest path routing problem.

**2.6 Genetic Algorithm:** In the field of artificial intelligence, a genetic algorithm (GA) is a search heuristic that mimics the process of natural selection. This heuristic (also sometimes called a metaheuristic) is routinely used to generate useful solutions to optimization and search problems.[1] Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

Genetic algorithms find application in bioinformatics, phylogenetic, computational science, engineering, economics, chemistry, manufacturing, mathematics, physics, pharmacometrics and other fields.

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

A typical genetic algorithm requires:

- 1) a genetic representation of the solution domain,
- 2) a fitness function to evaluate the solution domain.

## PSEUDO CODE FOR GENETIC ALGORITHM

Algorithm GA is

```
// start with an initial time
t := 0;

// initialize a usually random population of individuals
initpopulation P (t);

// evaluate fitness of all initial individuals of population
evaluate P (t);

// test for termination criterion (time, fitness, etc.)
while not done do

    // increase the time counter
    t := t + 1;

    // select a sub-population for offspring production
    P' := selectparents P (t);

    // recombine the "genes" of selected parents
    recombine P' (t);

    // perturb the mated population stochastically
    mutate P' (t);

    // evaluate it's new fitness
    evaluate P' (t);

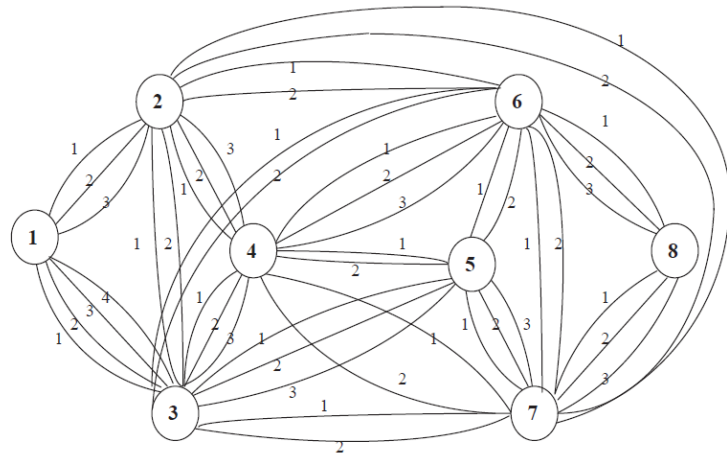
    // select the survivors from actual fitness
    P := survive P,P' (t);
od
end GA.
```



### 3. METHODOLOGY

The 4PLRPF can be defined as selecting logistic companies for a 4 PL provider to optimize the supply chain on purchase, sale and distribution logistic in diverse phases and locations. The primary aim of any 4PL provider is to minimize the total cost on any transportation route while following all of the constraints and assumptions. Assume that there is a network of 4PLRPF as shown below:

In the adjoining diagram, different nodes (having number 1 to 8 written in circle) represent cities from where the good can be transported and different arcs between the nodes represent different 3PL providers.



**Figure 2: Multi-graph of a 4PLRPF**

Whenever a 4PL provider is assigned a task to ship any goods, he first gathers data of the available 3PL suppliers and constructs the route using different cities and logistic providers.

As shown in figure above, the 4PLRPF can be represented by multi-graph  $G(V,E)$  where  $V(|V| = n)$  is the set of nodes, which represents cities or ports, and  $E$  is a set of edges, which connect pairs of nodes and represents different 3PL suppliers that are available to provide the transportation service between connected nodes. The variables used in representation of the 4PLRPF are defined as follows:

$r_{ij}$  : The number of edges (3PL suppliers) between node  $i$  and node  $j$ .

$e_{ijk}$ : The  $K$ -th edge between node  $i$  and node  $j$  ( $i,j=1,2,\dots,n$ ).

$x_{ijk}$ : Binary variable that represent, when there is a need to transport goods from source to the destination, whether edge  $e_{ijk}$  undertakes the transportation task between node  $i$  and node  $j$  if both node  $i$  and node  $j$  are selected. If  $e_{ijk}$  undertakes the corresponding task,  $x_{ijk} = 1$ ; else  $x_{ijk} = 0$ .

$y_i$ : Binary variable that represent whether node  $i$  undertakes transportation task. If node  $i$  undertakes the corresponding task,  $y_i = 1$ ; else  $y_i = 0$ .

$C_{ijk}$ ,  $P_{ijk}$ ,  $A_{ijk}$ : The cost, transportation capacity, and reputation of the 3PL supplier represented by  $e_{ijk}$ .

$E_{ijk}$ : The fuzzy duration time given by the  $K^{\text{th}}$  3PL supplier when they transport between  $i$  and  $j$ .

$C'_j$ : The cost of node  $j$ , including the cost of processing, inventory, loading, unloading etc.

$N_j$ : The fuzzy duration time of node  $j$ , including the time of processing, loading & unloading, changing vehicle, storage etc.

$P'_j$ : The processing capacity of node  $j$ , including the capacities of processing, manufacturing, storage etc.

$P, A$ : The transportation capacity and reputation that the selected 3PL suppliers are required by the customer to satisfy.

$P'$ : The transportation capacity that the selected nodes are required by the customer to satisfy.

$T$ : The due date of the task required by the customer.

With the help of above variables, a mathematical model can be made with an objective of delivering from the source node to the final destination which has minimum cost and subject to

constraints like final delivery, load requirement, supplier reputation etc. given by the customer.

The mathematical model for the 4PLRPF can be represented as follows:

$$Z = \min \{ \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^{r_{ij}} C_{ijk} x_{ijk} + \sum_{i=1}^n C'_i y_i \} \dots (1)$$

s.t.

$$\sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^{r_{ij}} E_{ijk} x_{ijk} + \sum_{i=1}^n N_i y_i \leq T \dots \dots (2)$$

$$\sum_{k=1}^{r_{ij}} \sum_{i=1}^n x_{ijk} - \sum_{k=1}^{r_{ij}} \sum_{i=1}^n x_{jik} = \begin{cases} -1 & \text{if } j = 1 \\ 1 & \text{if } j = n \\ 0 & \text{otherwise} \end{cases} \dots \dots (3)$$

$$\sum_{i=1}^n \sum_{k=1}^{r_{ij}} x_{ijk} = y_j, \quad j \in \{1, 2, \dots, n\} \quad (4)$$

$$\sum_{j=1}^n \sum_{k=1}^{r_{ij}} x_{ijk} = y_i, \quad i \in \{1, 2, \dots, n-1\} \quad (5)$$

$$P x_{ijk} \leq P_{ijk}, \quad i, j \in \{1, 2, \dots, n\}, \quad k \in \{1, 2, \dots, r_{ij}\} \quad (6)$$

$$P' y_i \leq P'_i, \quad i \in \{1, 2, \dots, n\} \quad (7)$$

$$A x_{ijk} \leq A_{ijk}, \quad i, j \in \{1, 2, \dots, n\}, \quad k \in \{1, 2, \dots, r_{ij}\} \quad (8)$$

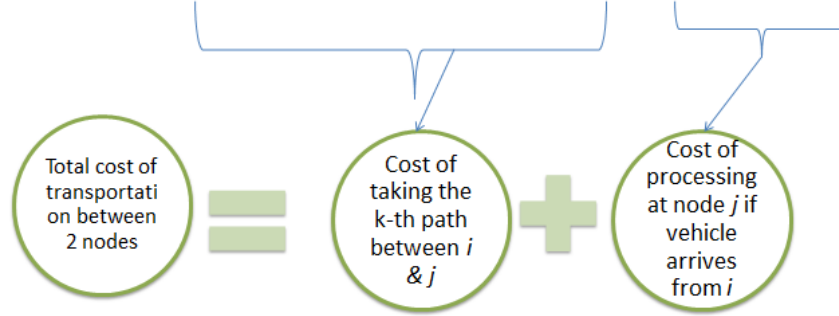
$$x_{ijk} = 0 \text{ or } 1, i, j \in \{1, 2, \dots, n\}, \quad k \in \{1, 2, \dots, r_{ij}\} \quad (9)$$

$$y_i = 0 \text{ or } 1, \quad i \in \{1, 2, \dots, n\} \quad (10)$$

### Explanation of equations:

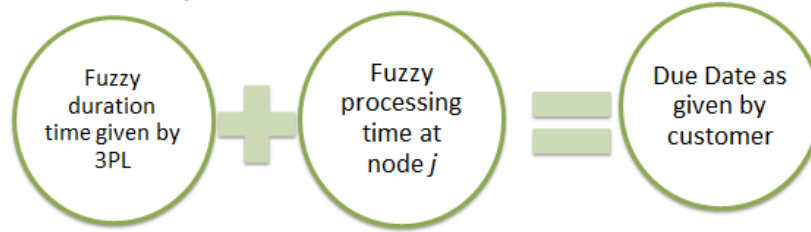
Equation 1:

$$Z = \min \left\{ \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^{r_{ij}} C_{ijk} x_{ijk} + \sum_{i=1}^n C'_i y_i \right\}$$



Equation 2:

$$\sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^{r_{ij}} E_{ijk} x_{ijk} + \sum_{i=1}^n N_i y_i \leq T$$



Equation 3:

$$\sum_{k=1}^{r_{ij}} \sum_{i=1}^n x_{ijk} - \sum_{k=1}^{r_{ij}} \sum_{i=1}^n x_{jik} = \begin{cases} -1 & \text{if } j = 1 \\ 1 & \text{if } j = n \\ 0 & \text{otherwise} \end{cases}$$

Eq. (3) means to keep a balance of the network flow;

Equation 4 & 5:

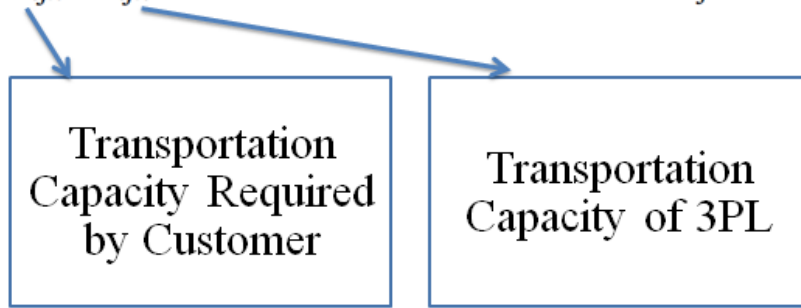
$$\sum_{i=1}^n \sum_{k=1}^{r_{ij}} x_{ijk} = y_j, \quad j \in \{2, 3, \dots, n\}$$

$$\sum_{j=1}^n \sum_{k=1}^{r_{ij}} x_{ijk} = y_i, \quad i \in \{1, 2, \dots, n-1\}$$

Eq. (4) and (5) ensure that the selected nodes and edges should make up of routes from the source to the destination;

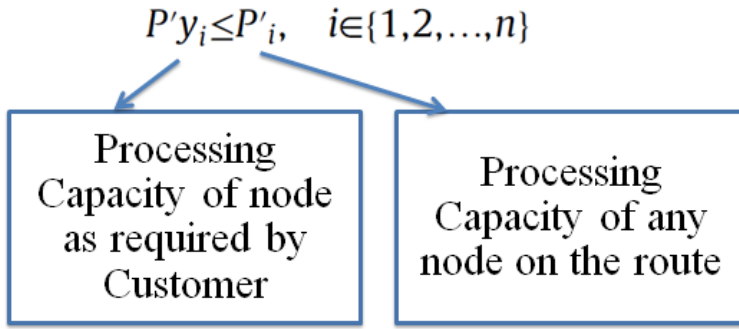
Equation 6:

$$Px_{ijk} \leq P_{ijk}, \quad i, j \in \{1, 2, \dots, n\}, \quad k \in \{1, 2, \dots, r_{ij}\}$$



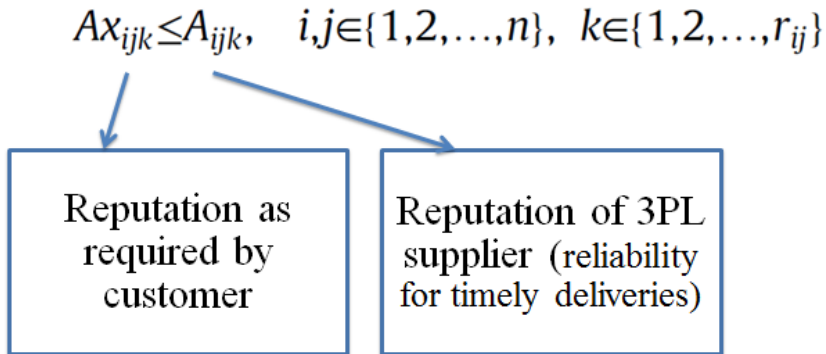
Eq. (6) ensures that the capacity constraint of the selected 3PL supplier is not less than the transportation capacity  $P$  required by the customer;

Equation 7:



Eq. (7) ensures that the capacity of any node on the route is not less than  $P'$ , which is required by the customer;

Equation 8:



Eq. (8) ensures that the reputation of the selected 3PL supplier is not less than  $A$ , which is required by the customer;

Equation 9& 10:

$$x_{ijk} = 0 \text{ or } 1, i, j \in \{1, 2, \dots, n\}, k \in \{1, 2, \dots, r_{ij}\}$$

$$y_i = 0 \text{ or } 1, i \in \{1, 2, \dots, n\}$$

Eq. (9) and (10) represent that  $x_{ijk}$  and  $y_i$  are 0-1 decision, respectively.

#### 4. ALGORITHM DESIGNED FOR THE 4PLRPF

The 4PLRPF described earlier is a difficult combinational optimization problem. KTGA with the fuzzy simulation is proposed to solve it as follows. The adjoining figure shows the flowchart of the proposed algorithm which is briefly described as follows:

- 4.1.1 Initialize the linkage matrix according to the model graph
- 4.1.2 Generate the initial population of PS individuals by the K-th shortest path algorithm, where PS denotes the population size.
- 4.1.3 Calculate the fitness function of the individuals with fuzzy simulation.
- 4.1.4 Select individuals by the roulette wheel selection scheme.
- 4.1.5 Perform crossover operations on pairs of selected individuals with a crossover probability  $P_c$ .

- 4.1.6 Perform mutation operations to individuals with a mutation probability  $P_m$ .
- 4.1.7 Check whether the maximum allowable number of generations, denoted  $NG_1$ , for the first step GA has been reached. If it is false, go to Step3; otherwise, stop the first step GA and output the solutions in the final population for the second step GA.
- 4.1.8 Expand solutions obtained by the first step GA to  $Q*PS$  solutions by selecting 3PL suppliers of the best individual randomly.
- 4.1.9 Calculate the fitness function of the individuals with fuzzy simulation.
- 4.1.10 Select individuals by the roulette wheel selection scheme.
- 4.1.11 Perform crossover operation on pairs of the selected 3PL suppliers' array of the obtained  $Q*PS$  individuals with a crossover probability  $P_c$
- 4.1.12 Perform the second step GA on the 3PL suppliers' array of the obtained  $Q*PS$  individuals for  $NG_2$  generations.
- 4.1.13 Stop the process and output the best solution according to the fitness function. In the following sections, fuzzy simulation process and major components of KTGA will be described in detail, respectively.



Flowchart of fuzzy simulation based KTGA:

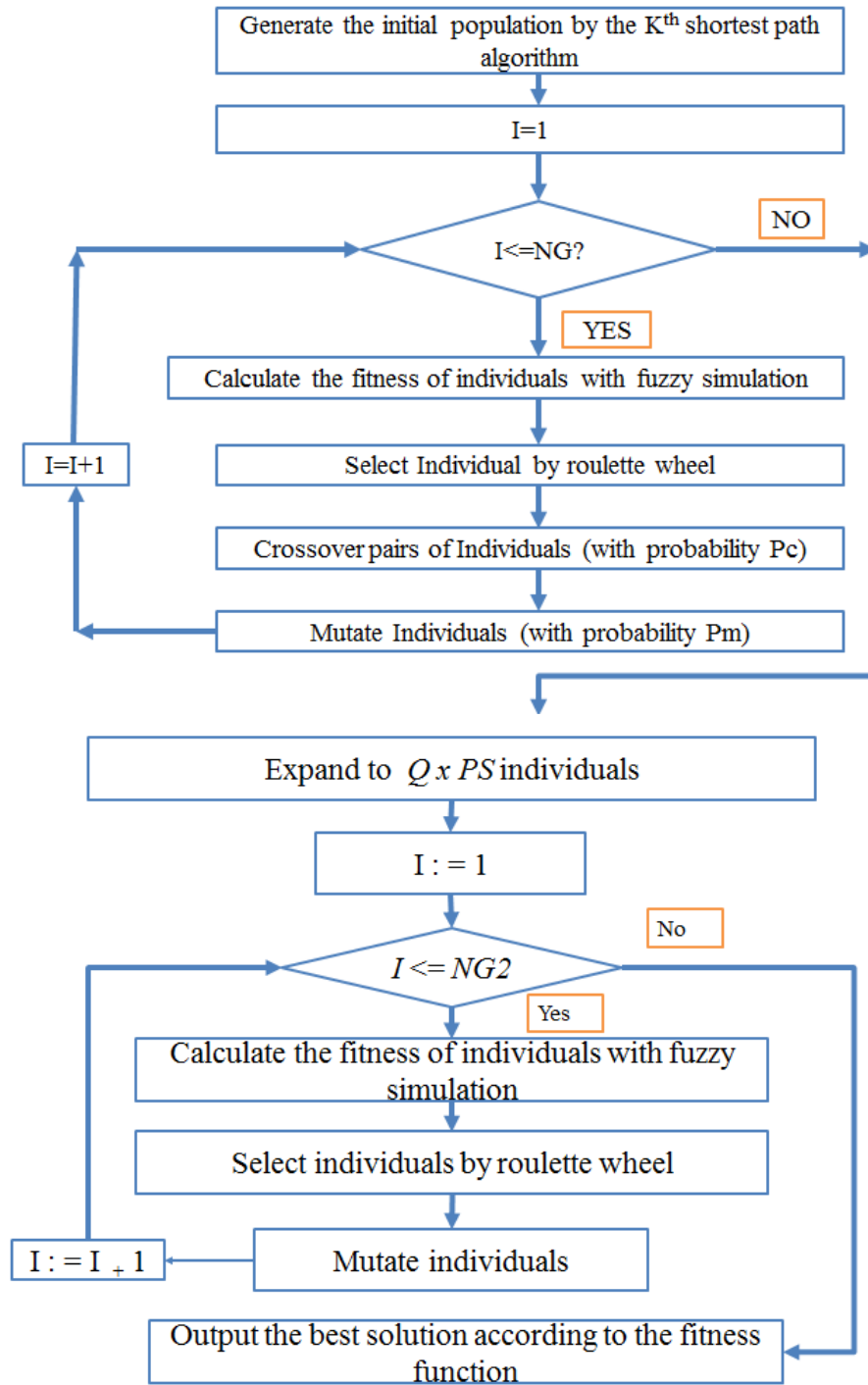


Figure 3: Flowchart for the fuzzy simulation based KTGA

## 4.2 TWO STEP GENETIC ALGORITHM:

According to the characteristic of the 4PLRPF, an encoding based on two arrays is adopted to represent a chromosome in KTGA, where the first array consists of nodes on a route encoded by natural number and the second array consists of 3PL suppliers (edges) on a route encoded by integer number. The length of a chromosome is variable. Suppose the number of nodes in the graph is  $n$  (i.e.,  $|V| = n$ ) and the index number is allocated to each node and edge (as shown in Fig). Then, the encoding of a chromosome is shown as follows:

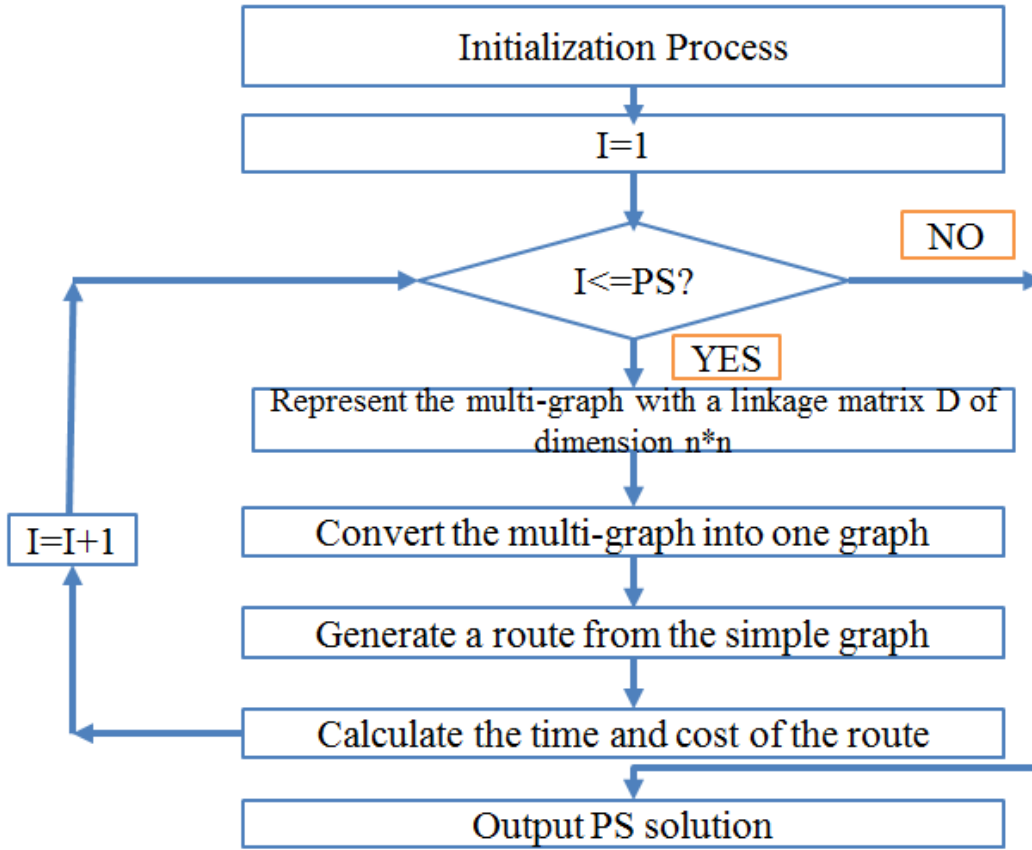
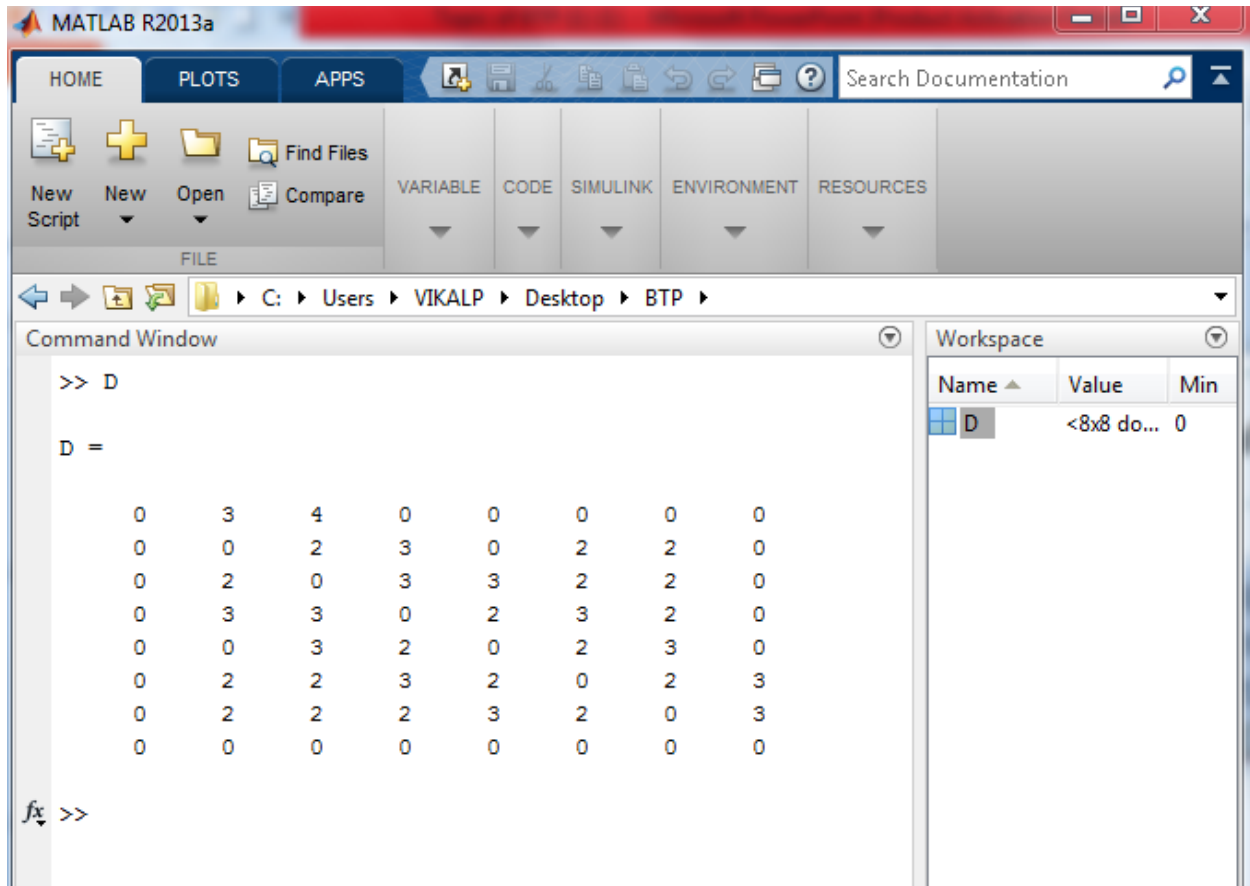


Figure 4: Flowchart of the initialization process

Step1: Represent the multi-graph with a linkage matrix  $D$  of dimension  $n \times n$  with respect to a source node and a destination node. Each element  $d_{ij} \in D$  represents whether node  $i$  is linked to

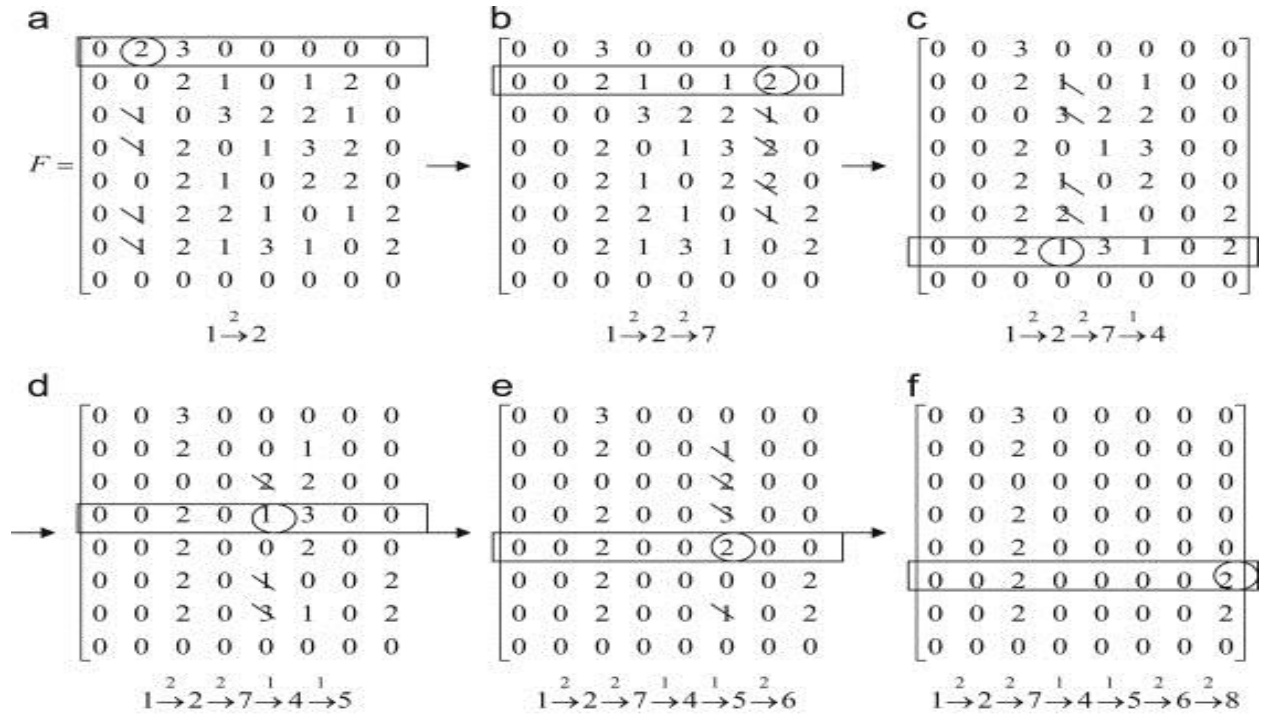
node  $j$  directly in the multi-graph. If there is no edge between node  $i$  and node  $j$ ,  $d_{ij}$  is set to zero; otherwise,  $d_{ij}$  is set to the number of edges between node  $i$  and node  $j$ . Furthermore, during the transportation, goods are not allowed to return to the source node after they leave from it, and they will terminate their traveling after they arrive at the destination. Take Fig. as an example, the corresponding linkage matrix  $D$  with node 1 ( $v_l=1$ ) as the source and node 8 ( $v_m=8$ ) as the destination as given as follows:



**Figure 5: Linkage matrix  $B$  representing multi-graph**

Step 2: Convert the multi-graph into a simple graph (i.e., there is at most one edge between any two nodes) as follows: an edge is randomly selected from available edges between any adjacent nodes in the multi-graph so that a sub-graph  $G'(V', E')$ , where  $|V'|=|V|=n$ ,  $E'$  is the set of edges in the sub-graph, is obtained, which can be described as a matrix  $F$  ( $F=(f_{ij})_{n \times n}$ ). Take the element

$d_{12}$  ( $d_{12}=3$ ) in the matrix D as an example, we first generate an integer randomly in the range of 1 to  $d_{12}$ , say 2 is generated. Then, set  $f_{ij}$  to 2, which means in the simple graph  $G'$  when goods are transported from node 1 to node 2, the second 3PL supplier is chosen to complete the task, and so on, then we can get the matrix as shown in figure:



**Figure 6: The process of converting a feasible route**

Step 3: Generate an initial route from the simple graph  $G'$  as follows. Starting from the source node  $v_1$ , we choose an adjacent node randomly as the second node on the route, record its index, mark it as “used”, which means we cannot use the node any more on this route, and set the elements of the corresponding column in the current simple graph to 0. Then, we randomly choose an un-used node from those nodes that are adjacent to the second node on the current route, record its index, mark it as “used”, and update the current simple graph accordingly. This process continues until the destination node is reached. If a node which is chosen has no other nodes adjacent and it is not the destination, then stop and re-start from the source node. Fig.

illustrates the process of generating a feasible route from the simple graph  $F$  converted from the multi-graph  $D$  in Eq. (16)

Step4: Calculate the time and cost of the corresponding route.

Step5: Repeat Steps 2–4 until PS solutions are generated.

### 4.3 The $K$ -th SHORTEST PATH ALGORITHM:

The quality of an initialization population has a direct impact on the algorithm. The method proposed in Section 3.2.2 can get an initialization population, but it is not effective when the number of nodes increases (see Section 4.3).  $K$ -th shortest path (KSP) algorithm is an effective way to solve constrained shortest path problem, but its application is limited to the simple graph. For the multi-graph, we can convert the multi-graph into simple graphs, then use KSP algorithm to solve every simple graph and get the shortest one. But it will be a large number if we enumerate all the possibility results. In order to get high quality solutions, we use KSP algorithm to get the initial individuals, which is employed in Step 3 during the initialization phase. According to the algorithm presented in Bellman (1958), this process is shown as follows:

*Step1:* Find the shortest path  $p^1$  that links from  $v_1$  to  $v_m$  in the simple graph  $G'(V', E')$  generated in Section 3.2.2. Record its adjacent matrix as  $F$  ( $F=(f_{ij})_{n \times n}$ ,  $i, j=1, 2, \dots, n$ ) and the element  $f_{ij}$  is the 3PL supplier selected to do the transportation if both  $i$  and  $j$  are on the route. Set  $k:=1$ ,  $p^k:=\{v_1^k, \dots, v_m^k\}$  ( $v_i^k (i=1, 2, \dots, m)$  is the  $i$ -th node on  $p^k$ ,  $m$  is the number of nodes on  $p^k$ ),  $P:=\{p^1\}$  and  $ii:=1$ .

*Step 2:* Calculate the time  $T_p^k$  needed on the route  $p^k$ . If  $T_p^k$  is less than the time  $T$  requested by the customer, terminate the algorithm and output  $p^k$  as the best individual; otherwise, go to Step 3.

*Step 3:* Set  $Ee := E' - \{(v_{ii}^k, v_{ii+1}^k) \mid (v_{ii}^k, v_{ii+1}^k) \text{ is an edge of } p^k\}$ , then find the shortest path  $pp_{ii}$  in  $G'(V', Ee)$ .

*Step 4:* If  $ii < m-1$ ,  $ii := ii+1$ ,  $Ee := E'$  and turn to Step 3; otherwise,  $Ee := E'$  and record a series of the shortest paths got in Step 3 as  $W$ , where  $W = \{pp^{ii} \mid ii = 1, \dots, m-1\}$ .

*Step 5:*  $P := P \cup W - \{p^k\}$  and  $k := k+1$ . Order the elements of  $P$  according to the cost from low to high and record the route with the lowest cost in  $P$  as the  $K$ -th shortest path  $p^k$ .

*Step 6:* If  $k > K$ , terminate the algorithm and output  $p^k$  as an individual in the initial population; otherwise turn to Step 2.

## C++ CODE FOR FINDING THE K-TH PATH

```
#include<stdio.h>
#include<conio.h>
using namespace std;

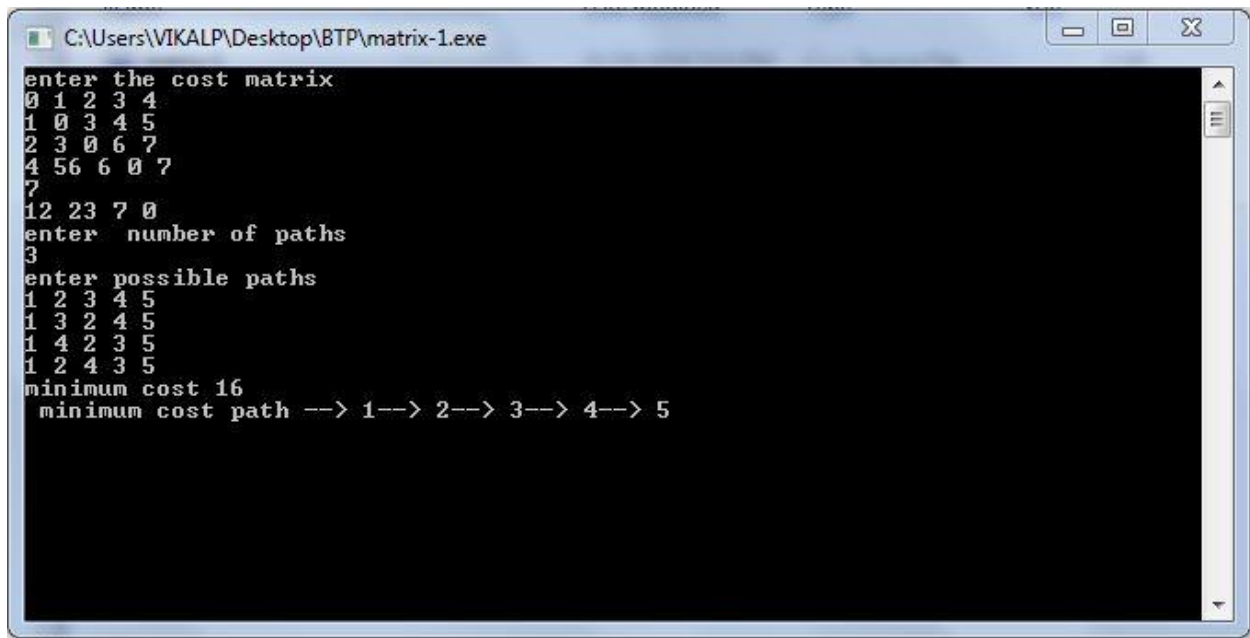
int main()
{
    int path[5][5],i,j,min,a[5][5],p,st=1,ed=5,stp,edp,t[5],index;    //define
    cost matrix of 5x5
    printf("enter the cost matrix\n");
    for(i=1;i<=5;i++)
    for(j=1;j<=5;j++)                                //value of the matrix first column
    and then move to second column
    scanf("%d",&a[i][j]);                               // enter the values manually
    printf("enter number of paths\n");
    scanf("%d",&p);                                     // p = number of paths
    printf("enter possible paths\n");
    for(i=1;i<=p;i++)
    for(j=1;j<=5;j++)
    scanf("%d",&path[i][j]);                           // enter the number of possible paths
    for(i=1;i<=p;i++)                                  // incrementing i till p (total
    number of paths)
    {
        t[i]=0;                                         // t= cost of the path; i = 1 here
        (refer to line 18)
        stp=st;                                         // st = 1 int
        for(j=1;j<=5;j++)                             //incrementing j till 5 ( total nodes
        in the path)
        {
            edp=path[i][j+1];                          // reading nodes of the path
            t[i]=t[i]+a[stp][edp];                      // adding the cost of the whole
            path, t[1] is the total cost of 1st path
            if(edp==ed)                                  // reached the destination or not
            ??
            break;
            else
            stp=edp;                                     // for moving along the path like starting from 1 then 3 in [1
            3 4 5 0]
        }
    }
    min=t[st];index=st;
    for(i=1;i<=p;i++)
    {
        if(min>t[i])
        {
            min=t[i];
            index=i;
        }
    }
    printf("minimum cost %d",min);
    printf("\n minimum cost path ");
    for(i=1;i<=5;i++)
    {
        printf("--> %d",path[index][i]);
    }
}
```

```

if(path[index][i]==ed)
break;
}
getc h();
return 0;
}

```

## Result:



```

C:\Users\VIKALP\Desktop\BTP\matrix-1.exe
enter the cost matrix
0 1 2 3 4
1 0 3 4 5
2 3 0 6 7
4 5 6 0 7
7
12 23 7 0
enter number of paths
3
enter possible paths
1 2 3 4 5
1 3 2 4 5
1 4 2 3 5
1 2 4 3 5
minimum cost 16
minimum cost path --> 1--> 2--> 3--> 4--> 5

```

## 4.4 FITNESS FUNCTION

Fitness function is usually designed according to the objective function and constrains of the problem, which can reflect the requirements of the customers and the consideration of the 4PL provider. In this paper, the time constraint is estimated by the fuzzy simulation, so when it is used as a punishment added to the objective function, by the theory mentioned in Section 3.1, the fitness function can be represented by



Equation (17)

$$f = \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n C_{ijk} X_{ijk} + \sum_{i=1}^n C'_i y_i + \beta * \max(0, f - T)$$

Where is estimated in Section 3.1 and is a punishing factor, which is set according to a particular problem. Here  $\beta$  is set to be a value and it is defined in Section 4.2.

After evaluating the fitness of the PS individuals in the population according to Eq. (17), we sort the individuals in the order of increasing fitness value (i.e., the better the chromosome is, the smaller the ordinal number it has). Then select them as an evaluation function, denoted by, to assign a probability of reproduction to each chromosome  $I_i$  so that its likelihood of being selected is proportional to its fitness relative to the other chromosomes in the population. That is, the chromosomes with higher fitness will have more chance to produce offspring by using roulette wheel selection.

Now let parameter  $\gamma \in (0, 1)$  in the genetic system be given, then the rank-based evaluation function of calculating the fitness of each chromosome  $I_i$  is given as follows:

$$eval(I_i) = \gamma(1 - \gamma)^{i-1}, i = 1, \dots, PS$$

#### 4.5 SELECTION MECHANISM:

The selection process is based on the roulette wheel selection scheme, which is a fitness-proportional selection. We spin the roulette wheel  $PS$  times. Each time we select a single

individual into the mating pool to undergo crossover and mutation. The selection process is described as follows:

Step1: Calculate the cumulative probability  $q_i$  for each individual  $I_i$  ( $i=1, \dots, PS$ ) as follows:

$$q_0 = 0, q_i = \sum_{j=1}^{j=i} eval(I_j), i = 1, \dots, PS$$

Step 2: Generate a random number  $r \in (0, q_{PS})$ .

Step 3: Select the individual  $I_i$  such that  $q_{i-1} < r \leq q_i$  ( $i \in \{1, \dots, PS\}$ ).

Step 4: Repeat Steps 2 and 3  $PS$  times to select  $PS$  individuals.

## 4.6 CROSSOVER OPERATION:

In every generation, crossover operation is applied on parents with a crossover probability  $P_c$ . This probability makes the expected number  $P_c * PS$  of individuals undergo the crossover operation.

From  $i=1$  to  $PS$ : generating a random number  $r$  from the interval  $[0, 1]$ , the individual  $B_i$  is selected as a parent if  $r < P_c$ . Denote the selected parents by  $B'_1, B'_2, \dots$ . Then divide them into the following pairs and delete the last one if the number of them is odd:

$(B'_1, B'_2), (B'_3, B'_4), (B'_5, B'_6), \dots$

Let us illustrate the crossover operator on each pair by  $(B'_1, B'_2)$ . Firstly, we determine the number of nodes on  $B'_1$  and  $B'_2$ , and record the less one as  $L$ . Secondly, we generate a random crossover point  $c$  between 2 and  $L-2$ . Thirdly, the crossover operator on  $B'_1$  and  $B'_2$  will produce two children  $BB'_1$  and  $BB'_2$  by exchanging the genetic materials after the crossover point  $c$ . An example of the crossover operation is given below. We assume the following two parents  $B'_1$  and  $B'_2$ :

$$B'_1: path[v_1, v_2, \dots, v_{c-1}: v_c, \dots, v_{m-1}, v_m]$$

$$PL[e_{v_1v_2k_1}, \dots, e_{v_{c-2}v_{c-1}k_{c-2}}: e_{v_{c-1}v_ck_{c-1}}, \dots, e_{v_{m-1}v_mk_{m-1}}]$$

$$B'_2: path[v'_1, v'_2, \dots, v'_{c-1}: v'_c, \dots, v'_{m-1}, v'_m]$$

$$PL[e_{v'_1v'_2k'_1}, \dots, e_{v'_{c-2}v'_{c-1}k'_{c-2}}: e_{v'_{c-1}v'_ck'_{c-1}}, \dots, e_{v'_{m-1}v'_mk'_{m-1}}]$$

We calculate  $L = \min(m, t)$  and generate a random integer  $c \in [2, L-2]$ . Then, two children  $BB'_1$  and  $BB'_2$  are generated as follows:

$$BB'_1: path[v_1, v_2, \dots, v_{c-1}: v'_c, \dots, v'_{m-1}, v'_m]$$

$$PL[e_{v_1v_2k_1}, \dots, e_{v_{c-2}v_{c-1}k_{c-2}}: e_{v'_{c-1}v'_ck'_{c-1}}, \dots, e_{v'_{m-1}v'_mk'_{m-1}}]$$

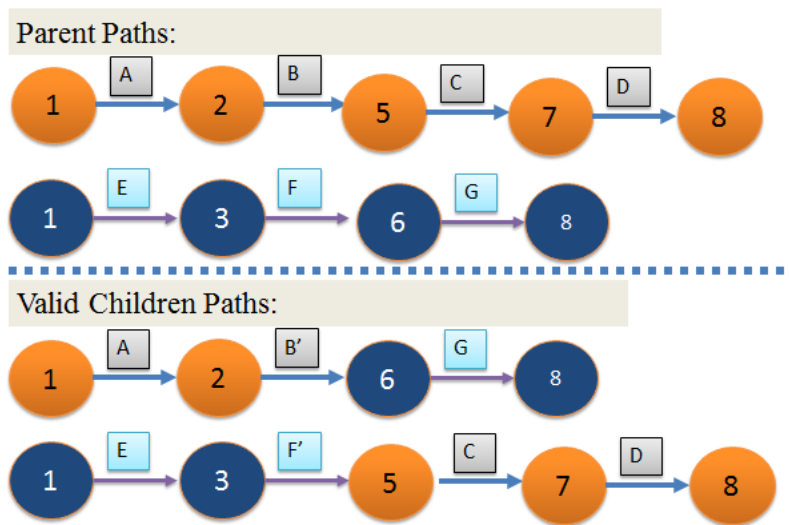
$$BB'_2: path[v'_1, v'_2, \dots, v'_{c-1}: v_c, \dots, v_{m-1}, v_m]$$

$$PL[e_{v'_1v'_2k'_1}, \dots, e_{v'_{c-2}v'_{c-1}k'_{c-2}}: e_{v_{c-1}v_ck_{c-1}}, \dots, e_{v_{m-1}v_mk_{m-1}}]$$

After the crossover operation, we need to check the feasibility of each child before accepting it. Take  $BB'_1$  as an example; if it is still a route from the source to the destination, we replace the parent  $B'_1$  with  $BB'_1$ . If not, try to do some reparation to the child as follows.

If there are same nodes on the route, delete the part between them, remove one duplicate node, and calculate the time and cost on the new route; otherwise, we examine whether the node before the crossover point is connected with the other nodes behind it. If that happens, we delete the part between them and randomly add a corresponding 3PL supplier, and replace the parent  $B'_1$  with  $BB'_1$ ; otherwise, quit with failure. Finally, if the child is infeasible, we use the corresponding parent instead.

If there are same nodes on the route, delete the part between them, remove one duplicate node, and calculate the time and cost on the new route; otherwise, we examine whether the node before the crossover point is connected with the other nodes behind it. If that happens, we delete the part between them and randomly add a corresponding 3PL supplier, and replace the parent  $B'_1$  with  $BB'_1$ ; otherwise, quit with failure. Finally, if the child is infeasible, we use the corresponding parent instead.



**Figure 7: Schematic explanation of Crossover operation, circle represents node (city) and letter in square box represents name of the 3PL supplier used.**

## 4.7 MUTATION OPERATION

Mutation is carried out on the PL array of individuals according to a probability  $P_m$ , as follows.

In a similar manner to the process of selecting parents for crossover operation, we repeat the following steps from  $i=1$  to  $PS$ : generating a random number  $r$  from the interval  $[0,1]$ , the individual  $Bi$  is selected as a parent for mutation if  $r < P_m$ .

For each selected parent, we mutate it in the following way. Firstly, we calculate the total number of nodes in the PL array, denoted as  $P1$ . Secondly, generate an integer  $p$  randomly from the interval  $[1, P1]$ . And at last, choose another 3PL supplier randomly to replace the one between node  $p-1$  and node  $p$  of the individual.

## 4.8 SECOND STEP GA USED FOR 3PL SUPPLIERS

When the first step GA finishes, it can obtain a satisfactory solution, but may not be able to find the optimum (see Table 4, KGA). Considering the characteristic of individuals, we found that after several generations, 99.5% of the best solutions' paths in the population will be the same as the previous leader with only some changes of the 3PL suppliers' array. For this reason, the second step GA is employed to adjust the routes which have the same path with the leader. In order to enhance the effect of optimization, we expand the individuals with the same path as the best individual of the population to  $Q*PS$  individuals. The detailed description is given below:

**Step1:** Find individuals in the final population obtained by the first step GA which have the same path as the best individual  $I_{best}$  in the population, and denote the total number of such individuals as  $\eta_{best}$ .

**Step2:** Expand the best individual  $I_{best}$  to  $Q*PS$  individuals by selecting 3PL suppliers randomly on the route as follows: if  $s$  and  $t$  are two adjacent nodes on the route represented by  $I_{best}$ , select an integer randomly from 1 to  $d_{st}$  ( $d_{st}$  is the number of 3PL suppliers between node  $s$  and node  $t$ ) as the 3PL supplier chosen to undertake the transportation from  $s$  to  $t$ . This process is repeated until all the edges of the route are assigned a 3PL supplier and hence a new individual with the same path as  $I_{best}$  is generated.

**Step3:** Repeat Step 2 until  $Q*PS - \eta_{best}$  new individuals are produced. These new individuals and the original  $\eta_{best}$  individuals obtained by the first step GA together form the initial population for the second step GA.

**Step4:** Select individuals for variation by the roulette wheel scheme.

**Step5:** Crossover the selected parents' 3PL array with a probability  $P_c$ .

**Step6:** Mutate the offspring' 3PL array with a probability  $P_m$ .

**Step7:** Calculate the fitness for each offspring and update the elitism if a better solution arises.

**Step 8:** Repeat Steps 4–7 for  $NG2$  cycles.

**Step9:** Output the best solution obtained as the final solution.

MATLAB code to optimize  $K^{\text{th}}$  path solution with genetic algorithm:

```
n= input('enter square matrix dimension '); %% matrix index
A=B; %% B is the matrix input by user Multigraph
matrix
p=zeros(1,n); %% path storage in p
p(1)=1; %% start with 1 always
s=zeros(1,n-1); %% the chosen supplier
for i=1:1:n
    a(i:i,:)=A(p(i):p(i),:); %% select a row
    x=find(a(i:i,:)); %% find all the non zero columns of selected row
    p(i+1)=x(randi(numel(x))); %% randomly select one column number
    v=A(p(i):p(i),p(i+1):p(i+1)); %% we already knew the address, time to
select the element
    s(i)=randi(v); %% randomly select the "supplier"
    A(:,p(i+1):p(i+1))=0; %% once you reached ith column (city), you need not
go there again
    A(:,p(i):p(i))=0; %% make the first city reaching way zero
%%if the last city is reached, you need not go
there again
    if p(i+1)==n
        break
    end
end
end
p
s
```

| B <8x8 double> |   |   |    |    |    |   |   |    |   |
|----------------|---|---|----|----|----|---|---|----|---|
|                | 1 | 2 | 3  | 4  | 5  | 6 | 7 | 8  | 9 |
| 1              | 0 | 3 | 5  | 2  | 0  | 4 | 7 | 2  |   |
| 2              | 2 | 0 | 4  | 1  | 0  | 5 | 8 | 0  |   |
| 3              | 4 | 6 | 0  | 0  | 8  | 4 | 5 | 1  |   |
| 4              | 2 | 3 | 5  | 0  | 7  | 0 | 3 | 11 |   |
| 5              | 1 | 4 | 25 | 0  | 0  | 3 | 4 | 6  |   |
| 6              | 4 | 5 | 2  | 1  | 10 | 0 | 6 | 4  |   |
| 7              | 7 | 8 | 0  | 5  | 4  | 5 | 0 | 6  |   |
| 8              | 8 | 2 | 5  | 45 | 5  | 8 | 1 | 0  |   |
| 9              |   |   |    |    |    |   |   |    |   |
| 10             |   |   |    |    |    |   |   |    |   |

**Table 1: Sample data used for initialization**

## RESULT OF INITIALIZATION ON SAMPLE DATA:

```
Command Window

>> CODE
enter square matrix dimension 8

p =

    1     7     2     4     3     8     0     0

s =

    7     8     1     3     1     0     0

fx >> |
```

### Explanation of Code for Genetic Algorithm

#### Ariel View

A GA tries to simulate the process of evolution that happens on Earth. First you create a bunch of organisms who each have a unique set of genes (usually chosen randomly). They are the first generation. Then you evaluate and rank the fitness of each of the organisms according to some criteria (e.g. which creature gets the fattest after two years). Then you chose some of the more fit organisms and let them reproduce with each other to produce the second generation. One the first generation has reproduced all the members of the first generation "die". Then you evaluate the second generation and the cycle repeats.



The GA keeps producing new generations until a perfect organism has been created or we get to, say, the 1000th generation and decide to stop. A run is the process of doing all the generations.

## Program Outline

The program can be seen as a small hierarchy of functions which are all in one file:

Function Heirarchy.cpp

```
1  main ()
2      AllocateMemory ()
3      DoOneRun ()
4          InitializeOrganisms ()
5          EvaluateOrganisms ()
6          ProduceNextGeneration ()
7          SelectOneOrganism ()
8
```

main()

Following is the code for main():

main()'s Code.cpp

```
1  #include <stdio.h>
2
3  int main() {
4      int finalGeneration;
5      AllocateMemory();
6      finalGeneration = DoOneRun();
7      printf("The final generation was: %d\n",
8             finalGeneration);
9  }
10
```

We include `stdio.h` so we can use the `printf()` function to print how many generations it took to get a perfect organism.

#### **`AllocateMemory()`**

Our first real task is to allocate memory to store the organisms in. The only information we need to store about each organism is its genetics (i.e. its set of genes).

We'll start with 100 organisms who each have 20 genes. We'll store each organism's genes as an array of char variables (not a "string" per se). The code for

`AllocateMemory()`:

```

[*] AllocateMemory()'s Code.cpp
1  #define NUMBER_ORGANISMS 100
2  #define NUMBER_GENES 20
3
4  char **currentGeneration, **nextGeneration; // globals
5  char *modelOrganism;
6  int *organismsFitnesses;
7
8  void AllocateMemory(void) {
9      int organism;
10
11      currentGeneration =
12          (char**)malloc(sizeof(char*) * NUMBER_ORGANISMS);
13      nextGeneration =
14          (char**)malloc(sizeof(char*) * NUMBER_ORGANISMS);
15      modelOrganism =
16          (char*)malloc(sizeof(char) * NUMBER_GENES);
17      organismsFitnesses =
18          (int*)malloc(sizeof(int) * NUMBER_ORGANISMS);
19
20      for(organism=0; organism<NUMBER_ORGANISMS; ++organism){
21          currentGeneration[organism] =
22              (char*)malloc(sizeof(char) * NUMBER_GENES);
23          nextGeneration[organism] =
24              (char*)malloc(sizeof(char) * NUMBER_GENES);
25      }
26  }
27

```

Notice that `currentGeneration` and `nextGeneration` are two global pointers which we'll use so we can manipulate the organism data in our various functions. `modelOrganism` is what we'll use as the "perfect" organism. Later on, we'll be comparing our regular organisms with the model to see *how* perfect they are. Conceptually though, the model organism isn't a real organism the way the others are. I hope the rest of the code isn't too dense to understand, I'll be skipping over most of the coding-specific issues for brevity.

### DoOneRun ()

First we initialize the organisms, then we use a simple loop to go from one generation to the next until a *perfect generation* is found. A perfect generation is one that has at least one organism which has the exact same genes as the model organism. When we get to a perfect generation, we return the generation number.

```
DoOneRun()'s Code.cpp
1  #define FALSE 0
2  #define TRUE 1
3
4
5  int DoOneRun(void) {
6      int generations = 1;
7      int perfectGeneration = FALSE;
8
9      InitializeOrganisms();
10
11     while(TRUE) {
12         perfectGeneration = EvaluateOrganisms();
13         if( perfectGeneration==TRUE ) return generations;
14         ProduceNextGeneration();
15         ++generations;
16     }
17 }
18
```

You'll notice I've written the code in a very simple way so it is easier to learn. Later on, I'll give a more concise and efficient version of the program.

### InitializeOrganisms ()

Before we start each run, we need to randomize the genes of the normal organisms

and the model. Notice that we don't need to do anything to the `nextGeneration` data structure since it is only a temporary holding area we use when we're producing the following generation.

Each gene can be one of four *alleles*: C, G, A, or T. But for simplicity, we'll represent their values as 0, 1, 2, and 3, respectively.

```
InitializeOrganisms()'s Code.cpp
1  #include <stdlib.h>
2
3  #define ALLELES 4
4
5  void InitializeOrganisms(void) {
6      int organism;
7      int gene;
8
9      // initialize the normal organisms
10     for(organism=0; organism<NUMBER_ORGANISMS; ++organism){
11         for(gene=0; gene<NUMBER_GENES; ++gene){
12             currentGeneration[organism][gene] = rand()%ALLELES;
13         }
14     }
15
16     // initialize the model organism
17     for(gene=0; gene<NUMBER_GENES; ++gene){
18         modelOrganism[gene] = rand()%ALLELES;
19     }
20 }
21
```

#### EvaluateOrganisms()

The evaluation stage has two purposes. Primarily, we have to determine the fitness of

all the organisms so that later on, in `ProduceNextGeneration()`, we'll know which were the better organisms and therefore which should reproduce more often. Our secondary purpose is to decide if we've got a perfect generation, one with at least one organism that has the same genetics as the model.

Each organism's fitness is simply the number of its genes that match the model (they have to be in the right position too). So, if the organism we're considering is `AAAAAACAAAAAAAAAAAAA` and the model is `CCCCCCCCCCCCCCCCCCCC`, then the organism gets a fitness of 1. The one point comes from both the model and the organism having a 'C' as their seventh gene.

We introduce another global variable `totalOfFitnesses` here. It is simply the sum of each organisms fitness. Ignore it for now, we'll use it later on in `SelectOneMember()`.

The code is as follows:

EvaluateOrganisms()'s Code.cpp

```
1  #define MAXIMUM_FITNESS NUMBER_GENES
2
3  int totalOfFitnesses;
4
5  int EvaluateOrganisms(void){
6      int organism;
7      int gene;
8      int currentOrganismsFitnessTally;
9
10     totalOfFitnesses = 0;
11
12     for(organism=0; organism<NUMBER_ORGANISMS; ++organism){
13         currentOrganismsFitnessTally = 0;
14
15         // tally up the current organism's fitness
16         for(gene=0; gene<NUMBER_GENES; ++gene){
17             if( currentGeneration[organism][gene]
18                 == modelOrganism[gene] ){
19                 ++currentOrganismsFitnessTally;
20             }
21         }
22
23         // save the tally in the fitnesses data structure
24         // and add its fitness to the generation's total
25         organismsFitnesses[organism] =
26             currentOrganismsFitnessTally;
27         totalOfFitnesses += currentOrganismsFitnessTally;
28
29         // check if we have a perfect generation
30         if( currentOrganismsFitnessTally == MAXIMUM_FITNESS ){
31             return TRUE;
32         }
33     }
34     return FALSE;
35 }
36
```

#### `ProduceNextGeneration()`

Once we've figured out the fitness of all the organisms in our current generation, we can select the best organisms and reproduce them. We'll temporarily store each organism of the new generation, the children, in the `nextGeneration` data structure. After we've created all the children, we copy them into the `currentGeneration` data structure and the reproduction phase is complete.

Now a little more detail on how the children are created. For each child, we first use `SelectOneMember()` twice to get the two parents. Then we randomly select a *crossover point* and start copying over the parents' genes to the child. The genes to the left of the crossover point are copied over from parent one, while the genes right of the crossover point come from parent two.

Each time we copy over a gene, there is a chance it will *mutate* into a random gene.



```

1  #define MUTATION_RATE 0.001
2
3  void ProduceNextGeneration(void){
4      int organism;
5      int gene;
6      int parentOne;
7      int parentTwo;
8      int crossoverPoint;
9      int mutateThisGene;
10
11     // fill the nextGeneration data structure with the
12     // children
13     for(organism=0; organism<NUMBER_ORGANISMS; ++organism){
14         parentOne = SelectOneOrganism();
15         parentTwo = SelectOneOrganism();
16         crossoverPoint = rand() % NUMBER_GENES;
17
18         for(gene=0; gene<NUMBER_GENES; ++gene){
19
20             // copy over a single gene
21             mutateThisGene = rand() % (int)(1.0 / MUTATION_RATE);
22             if(mutateThisGene == 0){
23
24                 // we decided to make this gene a mutation
25                 nextGeneration[organism][gene] = rand() % ALLELES;
26             } else {
27                 // we decided to copy this gene from a parent
28                 if (gene < crossoverPoint){
29                     nextGeneration[organism][gene] =
30                         currentGeneration[parentOne][gene];
31                 } else {
32                     nextGeneration[organism][gene] =
33                         currentGeneration[parentTwo][gene];
34                 }
35             }
36         }
37     }
38
39     // copy the children in nextGeneration into
40     // currentGeneration
41     for(organism=0; organism<NUMBER_ORGANISMS; ++organism){
42         for(gene=0; gene<NUMBER_GENES; ++gene){
43             currentGeneration[organism][gene] =
44                 nextGeneration[organism][gene];
45         }
46     }
47 }

```

`SelectOneOrganism()`

How you select organisms for reproduction will determine how effective your GA is. The simplest method is *roulette wheel* sampling, which we'll use here. Metaphorically, each organism is "assigned" a slice of the roulette wheel. The *size* of the slice each organism gets is proportional to its fitness. Then, we "spin" the wheel and whichever slice we land on, that organism gets selected.

How we'll implement this is best shown with a visual example. Imagine we have only five organisms with fitnesses of 3, 2, 0, 5, and 2. So the `totalOfFitnesses` is 12. I'll abbreviate `organismsFitnesses` below to `oF`. We "line up" the organisms from left to right as follows:

|                  |    |       |    |       |    |    |    |       |    |    |    |
|------------------|----|-------|----|-------|----|----|----|-------|----|----|----|
| 01               | 02 | 03    | 04 | 05    | 06 | 07 | 08 | 09    | 10 | 11 | 12 |
| totalOfFitnesses |    |       |    |       |    |    |    |       |    |    |    |
| oF[0]            |    | oF[1] |    | oF[3] |    |    |    | oF[4] |    |    |    |

**Table 2: Line-up of organisms**

Notice `oF[2]` didn't get any slots since its fitness was 0. Anyway, the process is, first we pick a random number from 1 to 12. Then we start adding up organisms from left to right until our running total is as big as that number. Suppose we randomly chose 9. Then we'd start by adding `oF[0]` to `runningTotal` to get 3, which isn't as big as than 9 so we keep going. Next we add `oF[1]` so `runningTotal` is 5. After adding `oF[2]` the

total is still 5. Finally, when we add `oF[3]`, and `runningTotal` becomes 10, which is as big as 9, so we select `oF[3]`.

The code for implementing it is as follows:

```
SelectOneOrganism()'s Code.cpp
1  int SelectOneOrganism(void) {
2      int organism;
3      int runningTotal;
4      int randomSelectPoint;
5
6      runningTotal = 0;
7      randomSelectPoint = rand() % (totalOfFitnesses + 1);
8
9      for(organism=0; organism<NUMBER_ORGANISMS; ++organism){
10         runningTotal += organismsFitnesses[organism];
11         if(runningTotal >= randomSelectPoint) return organism;
12     }
13 }
14
```

## **5. DISCUSSION AND FUTURE**

The above model can be used by an existing e-commerce company to optimize its usage of 3PL suppliers and get timely delivery for the products. All the existing technologies, like real time tracking of packages by customer and company, can be included into it. Even though the above model can be replicated into a successful business operation, the work and study done has opened new horizons for further development of optimization model of logistics and supply chain.

One factor that can be worked upon into the future and induced in this model is Credibility. Take for examples – Merchant Banks. Banks compile all the data of their customers and share it with other banks to calculate credit-worthiness of any individual which further helps them to minimize risk of negative loans or defaults. In a similar fashion, the e-commerce company can gather the data on the 3PL suppliers and consider modeling their credibility so that the leading and efficient 3PL suppliers can be identified and further rewarded with more orders and easy finances.

Last mile delivery has not been included in the project model. Last mile delivery refers to how the package should be delivered to the customer when it has reached his node or city. It becomes a challenging task in the metropolitan city where frequency of orders is too high and some lanes of transport are clogged due to high traffic.

Benchmarking can be done once the model is executed. It can be worked upon where this model needs to improve when compared to results of other models. In the model, categorization of orders can also be included, such as marking certain packages resilient, fragile and extremely fragile. Special 3PL suppliers can be developed for each.

## 6. CONCLUSION AND RESULTS

With the increasing intense of market competition, more and more enterprises begin to realize the importance of fourth party logistics and pay more attention to the fourth party logistics routing problem (4PLRP). In this project, the node-to-node, single task 4PLRP with fuzzy duration time (4PLRPF) is discussed, which can be described as a selection of the shortest path problem with constraints of fuzzy variants in a multi-graph. A fuzzy programming model is built up for this problem based on the two-step genetic algorithm with fuzzy simulation is proposed to solve the modeled 4PLRPF. In the proposed KTGA, the  $K$ -th shortest path algorithm is employed for generating high-quality initial individuals. Then, the first step GA is employed to search for satisfactory solutions based on the double arrays encoding. After the first step GA finishes, the best solutions obtained are expanded by randomly selecting 3PL suppliers to obtain more solutions and the second step GA is applied to expand solutions but only on the 3PL supplier array of the encoding scheme.

Numerical experiments were carried out to investigate the performance of the proposed KTGA. The experimental results indicate that the proposed KTGA is a new efficient method to solve the 4PLRPF.

In the future work, other fuzzy factors, such as fuzzy cost could be discussed. Moreover, more realistic problem, such as multi-source problem could be discussed.

## REFERENCES

- 1) *Bade, D., 2010.* New for the millennium: 4L trademark. *Transportation and Distribution* 52(11–12), 1957–1965.
- 2) *Bauknight, D.N., Miller, J.R., 1999.* Fourth party logistics: the evolution of supply chain outsourcing. *CALM Supply Chain and Logistics Journal*. Bellman, R., 1958. On a routing problem. *Quarterly of Applied Mathematics* 16(1), 87–90.
- 3) *Bolanda, N., Dethridgea, J., Dumitrescu, I., 2006.* Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters* 34(1), 58–68.
- 4) *Bolanda, N., Dethridgea, J., Dumitrescu, I., 2006.* Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters* 34(1), 58–68.
- 5) *Bumstead, J., Kempton, C., 2002.* From 4PL to managed supply chain operations. *Logistics and Transport Focus* (4), 18–24.
- 6) *Cao, E., Lai, M.Y., 2010.* The open vehicle routing problem with fuzzy demands. *Expert Systems with Applications* 37(3), 2405–2411.
- 7) *Chen, J.Q., Liu, W.H., Li, X., 2003.* Directed graph optimization model and its solving method based on genetic algorithm in fourth party logistics. *Proceedings of the 2003 IEEE International Conference on Systems, Man and Cybernetics* 2, 1961–1966
- 8) *Dubois, D., Prade, H., 1978.* Operations on fuzzy numbers. *International Journal of Systems Science* 9(6), 613–626.
- 9) *Foster, T., 1999.* 4PLs: the next generation for supply chain outsourcing. *Logistics Management and Distribution Report* 38(4), 35.
- 10) *Gabriel, Y.H., Israel, Z., 1980.* A dual algorithm for the constrained shortest path problem. *Networks* 10, 293–310.
- 11) *Gao, Y., 2011.* Shortest path problem with uncertain arc lengths. *Computers and Mathematics with Applications* 62(6), 2591–2600.
- 12) *Ghoseiri, K., Nadjari, B., 2010.* An ant colony optimization algorithm for the bi-objective shortest path problem. *Applied Soft Computing* 10(4), 1237–1246.
- 13) *Glinas, I.S., Soumis, F., Desrosiers, J., 1998.* A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks* 31(3), 193–204.
- 14) *Han, L.Q., 2003.* Outsourcing, supply chain integration and the fourth party logistic. *Quantitative and Technical Economics* (7), 154–157