Technische Universität München

Chair For Logistics and Supply Chain Management

Inter Discplinary Project

Implementation of a Metaheuristic for the Discrete Network Design Problem

---

Student:
Nishanth Nagendra

Supervisor:
Prof. Dr. Stefan Minner

Advisor:
Dipl.-Math. Pirmin Fontaine

October 12, 2015

Acknowledgement

I would like to thank Mr. Pirmin Fontaine for his constant support and guidance throughout the project especially when I faced some challenges in understanding certain topics. I would also like to thank Prof. Dr. Stefan Minner for providing me with this opportunity to work on such an interesting Inter-Disciplinary project.

Abstract

The field of transportation planning involves a large class of problems that are characterized by multiple levels of decision making. Examples include selection of links for capacity improvements(network design), toll setting, traffic signal setting etc. In each of these problems, government or industry officials make one set of decisions to improve the network performance and at another level users make choices with regard to route, origin-destination etc. The discrete network design problem(DNDP) is one such problem which deals with the selection of link additions to an existing road network, with given demand from each origin to destination. The objective is to make an optimal investment decision to minimize the total travel cost in the network, while accounting for the route choice behavior of network users. This optimization problem is recognized to be NP hard because it is computationally difficult due to its non convexity owing to the bilevel nature and non-linear objective functions in most of the real cases. Finding exact or globally optimum solutions for such problems is very difficult. In mathematical optimization, a metaheuristic is a general high level procedure that can be quickly applied to different kinds of problems to provide a sufficiently good solution especially when there is limited computation capacity. These techniques sample a set of solutions which is too large to be completely sampled. Genetic algorithm is one such kind of a metaheuristic which tries to imitate the evolution of population by starting with a random set of candidate solutions that is evolved towards better solutions. The aim of the project is to implement this metaheuristic on the DNDP and to experiment with small to large size datasets(networks) widely mentioned in the literature to derive empirical results and to conclude on the effectiveness, solvability and quality of solutions obtained.

Contents

List of Figures

List of Tables

1 Introduction

The network design problem(NDP) is concerned with the modification of a transport infrastructure by adding new links or improving existing ones. In most applications, one is interested in selecting from among a relatively small set of improvements to an existing network rather than designing an entirely new network from scratch. Network design problems arise in many transportation modes: urban mass transit, highway, rail etc. Most applications have been in highway improvement, however. The discrete network design problem(DNDP) deals with the selection of link additions to an existing road network, with given demand from each origin to destination. The objective is to make an optimal investment decision in order to minimize the total travel cost in the network, while accounting for the route choice behavior of network users. Another form of this problem is the continuous network design problem(CNDP) which deals with the optimal capacity expansion of existing links.

These combinatorial problems are recognized as NP Hard due to the considerable amount of computational difficulties faced in trying to solve them because of their non-convex nature and due to the form of a bi-level mixed integer program with a large number of 0-1 variables. This Inter disciplinary project deals with only the discrete form of the NDP and considers the following kind of problem which often arises in transportation planning: minimize total travel expenditure subject to a budget constraint on the total construction cost of new links.

1.1 Problem Formulation*

Bilevel problems are mathematical programming problems consisting of a special kind of optimization where one problem is embedded within another. The outer optimization task is commonly referred to as the upper level(or the leader) optimization task with a nested optimization problem(the lower one - also called follower) in the constraints. First, the leader has to decide over a subset of the decision variables which effects the objective value of the follower. Afterwards, the follower has to decide over the other subset of decision variables, which affects the objective value of the leader. The problem we are considering to solve here is a bilevel linear program with binary leader variables and continuous follower variables. An application of bilevel programming is network design problem where the objective of the network designer is to reduce congestion in the network and the objective of the follower is to find the fastest way from origin to destination.Bilevel programming problems are mathematical optimization problems where the set of all decision variables is partitioned between two vectors say x and y. x is to be chosen as an optimal solution of the lower level problem parameterized in y.

** Content derived from the book **Foundations of Bilevel Programming(Non-Convex Optimization and its Applications)** by **Stephan Dempe.** Chapter 1,3*

Let the lower problem be introduced first as follows:

$$\min_{\boldsymbol{x}}\{\boldsymbol{f(x,y)} : \boldsymbol{g(x,y)} \leq \mathbf{0},\ \boldsymbol{h(x,y)} = \mathbf{0}\},$$

where $f : \Re^n \times \Re^m \to \Re,\ g : \Re^n \times \Re^m \to \Re^p,\ h : \Re^n \times \Re^m \to \Re^q, g(x,y) = (g_1(x,y), ..., g_p(x,y))^T, h(x,y) = (h_1(x,y), ..., h_q(x,y))^T$ This is the *lower level or the follower's problem.* Let $\Psi(y)$ denote the solution set of this problem for a fixed $y \in \Re^m$. Then Ψ is so called *point-to-set mapping* from $\Re^m$ into the power set of $\Re^n$ denoted by $\Psi : \Re^m \to 2^{\Re^n}$. Denote some element of $\Psi(y)$ by $x(y)$ and assume for the moment that this choice is unique for all possible y. Then, the aim of the bilevel programming problem is to select the parameter vector y parameterizing the lower level problem which is the optimal one in a certain sense. To be more precise, this selection of y is conducted so that certain (nonlinear) equality and/or inequality constraints are satisfied

$$\boldsymbol{G}(\boldsymbol{x}(\boldsymbol{y})\ \boldsymbol{y}) \leq \mathbf{0},\ \boldsymbol{H}(\boldsymbol{x}(\boldsymbol{y})\ \boldsymbol{y}) = \mathbf{0}$$

and an objective function $F(x(y), y)$ is minimized, where $F : \Re^n \times \Re^m \to \Re, G : \Re^n \times \Re^m \to \Re^k, H : \Re^n \times \Re^m \to \Re^l$. The problem of determining a best solution y^* of parameters for the parametric optimization problem which together with the response $x(y) \in \Psi(y)$ proves to satisfy the constraints and to give the best possible function value for $F(x(y), y)$. That is

$$\text{" }\min_{\boldsymbol{y}}\text{"}\{\boldsymbol{F}(\boldsymbol{x}(\boldsymbol{y}), \boldsymbol{y}) : \boldsymbol{G}(\boldsymbol{x}(\boldsymbol{y}), \boldsymbol{y}) \leq \mathbf{0},\ \boldsymbol{H}(\boldsymbol{x}(\boldsymbol{y}), \boldsymbol{y}) = \mathbf{0}, \boldsymbol{x}(\boldsymbol{y}) \in \boldsymbol{\Psi}(\boldsymbol{y})\}$$

This problem is the *bilevel programming problem* or the *leader's problem.* Strongly speaking, this definition of the bilevel programming problem is valid only in the case when the lower level solution is uniquely determined for each possible $\boldsymbol{y}$. The quotation marks have been used to express this uncertainity in the definition of the bilevel programming problem in case of non-uniquely determined lower level optimal solutions. If the lower level problem has atmost one(global) optimal solution for all the values of the parameter, the quotation marks can be dropped and the familiar notation of an optimization problem arises.

Example: Consider the problem of minimizing the upper level objective function

$$\boldsymbol{3x + y \to \min}$$

subject to $\mathbf{1} \leq \boldsymbol{y} \leq \mathbf{6}$ and $\boldsymbol{x}$ solving the lower level problem.

$$\min_{\boldsymbol{x}}\{\boldsymbol{-x} : \boldsymbol{x + y} \leq \mathbf{8}, \boldsymbol{4x + y} \geq \mathbf{8}, \boldsymbol{2x + y} \leq \mathbf{13}\}$$

The set of pairs $\boldsymbol{(x,y)}$ satisying the constraints of both the lower and upper levels, denoted by M, as well as the minimization problem of the objective functions of both the lower and the upper level problems marked by the arrows are depicted in the figure[ref to figure].

The feasible set of the lower level problem for a fixed value of y is just the intersection of the set M with the set of all points above the point $\boldsymbol{(0,y)}$ on the $\boldsymbol{y}$-axis. Now, if the function

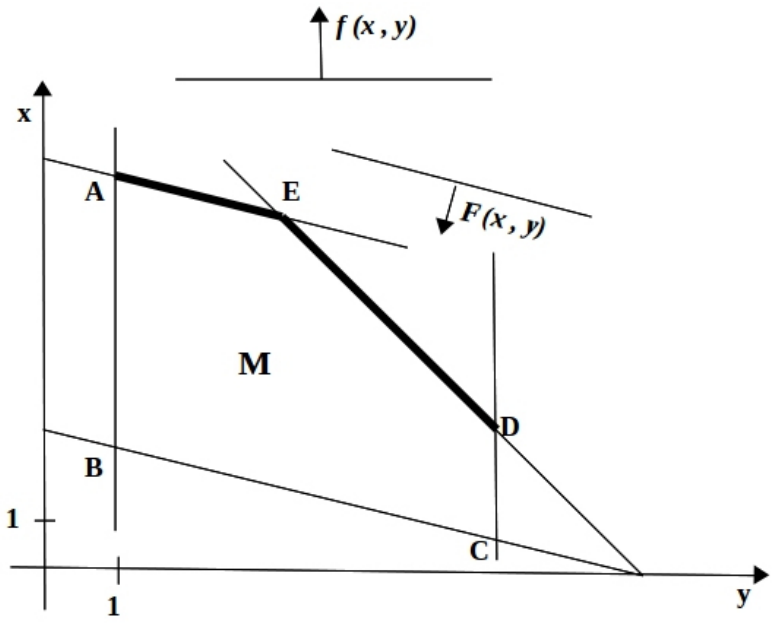

Figure 1: BLP

$\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y}) = -\boldsymbol{x}$ is minimized on this set, we arrive at a point on the thick line which thus is the optimal solution of the lower level problem:

$$\boldsymbol{x(y)} := \begin{cases} \mathbf{6.5 - 0.5}\boldsymbol{y} & \text{for } \mathbf{1} \leq \boldsymbol{y} \leq \mathbf{3}, \\ \mathbf{8 - }\boldsymbol{y} & \text{for } \mathbf{3} \leq \boldsymbol{y} \leq \mathbf{6}, \end{cases}$$

In doing all this for all values of $\boldsymbol{y}$ between $\mathbf{1}$ and $\mathbf{6}$, all points on the thick lines are obtained. Hence, the thick lines give the set of feasible solutions to the upper-level problem. On this set, the upper level objective function $\boldsymbol{F}(\boldsymbol{x},\boldsymbol{y})$ is to be minimized.

$$\boldsymbol{F(x(y),y)} = \begin{cases} \mathbf{19.5 - 0.5}\boldsymbol{y} & \text{for } \mathbf{1} \leq \boldsymbol{y} \leq \mathbf{3}, \\ \mathbf{24 - 2}\boldsymbol{y} & \text{for } \mathbf{3} \leq \boldsymbol{y} \leq \mathbf{6}. \end{cases}$$

Existing methods for solving bilevel problems can be roughly divided into the following categories:

- **Methods based on vertex enumeration:** In these methods, the basic idea is that the vertex points of the accessible region for BLP are vertex points for feasible space of the problem, and, the optimal solution is one of these vertices.
- **Methods based on kuhn tucker conditions:** In these methods, the second level problem is replaced by the Kuhn-Tucker optimiality conditions, yielding a one-level optimization problem with complimentarity constraints. Hence, the problem transfers into a single objective with complimentarity constraints. Several algorithms exist that try to solve these problems. One approach uses branch and bound to solve this. Another approach introduces a penalty function to satisfy the complimentarity constraints. Another approach converts the complimentarity constraints to linear one by adding some binary variables. One more approach develops parametric complimentary pivot(PCP) algorithm

by transferring the problem into a single objective in which the objective function of the first level is augmented into the set of constraints as follows:

$$\boldsymbol{c_1 x \ + \ d_1 y \ \geq \ \alpha}$$

and thereby the problem becomes linear complimentary problem (LCP).

- **Fuzzy approach:** This approach utilizes the concept of membership functions to develop a fuzzy approach for solving the above problems. The upper level decision maker defines his or her objective function and decisions with possible tolerances which are described by membership functions of fuzzy set theory and fuzzy decision. This information constrains the lower-level decision maker's feasible space. A solution search relies on changes of membership functions expressing satisfactory degrees of potential solutions for both upper and lower decision making, instead of vertex enumeration and no higher constraints are generated. No pre-assumption is made that the optimal solution exists at corner points. Potential satisfactory solutions are those in the non-dominated region when considering a multi-level decision making process rather than restricting to potential solutions at corners.

- **Methods based on meta heuristics:** In this category, Mathieu et al. [19] developed a genetic algorithm based bilevel programming algorithm(GABBA). In this method, the leader's decision vector is generated and the follower's reaction is obtained from the solution of a linear programming problem. It is different from the usual genetic algorithms because they only use mutations. Also, in this method the search space is expanded considerably. The reason is that, each feasible chromosome represents an accessible solution, but not necessarily an extreme point. Methods have been proposed based on Simulated Annealing and Hybrid Tabu descent technique. Genetic algorithm based approaches have also been developed where each feasible chromosome represesnts a vertex point of the accessible region and thereby reduce the search space significantly.

1.2 InterDisciplinary Project

The implementation of this project required knowledge of C/C++ programming, data structures, linear programming concepts, usage of the C++ BCL(builder component library) provided by a commercially well known optimization suite called FICO Xpress. The application of these technologies is for a well known real world problem like DNDP observed in transportation science resulting in this project to be of Inter-Disciplinary nature. As a part of the preparation for implementing this project, a good amount of knowledge on **metaheuristics** and **DNDP** was acquired from the course **"TRANSPORTATION LOGISTICS"** offered by the **CHAIR OF LOGISTICS AND SUPPLY CHAIN MANAGEMENT.**

The objective of this course was to get an overview of the modeling techniques, exact as well as heuristic search methods tailored to the different transportation problems studied. The course consisted of a sequence of lectures, exercise classes and case studies. It also deals on how to model and analyze transportation problems using quantitative methods. It covers many of the real world problems observed in Transportation Planning such as Traveling Salesman Problem(TSP), Vehicle Routing Problems(VRP), Network Flow Problems, Inventory Routing etc. and techniques such as Metaheuristics used to solve them.

- **Transportation Problems**: Linear programming formulations, Solving methods like Northwest corner rule, Column minimum rule, Matrix minimum rule, MODI(Modified Distribution) method, Extensions to Transshipment problem and several others.
- **Traveling Salesman Problem**: TSP models(Symmetric and Asymmetric), Exact solution methods, Heuristics like nearest neighbor and r-opt method, Arc Routing problem.
- **Vehicle Routing Problem**: Different modeling schemes for VRP using Linear programming formulations, Sweep method, Savings method as solution techniques for VRP problems, Extensions of VRP and Savings method.
- **Inventory Routing Problem**: Lot-Sizing Problem, Inventory Routing Problem(IRP)
- **Transportation planning in networks**: Dynamic optimization, Hub-Spoke systems, Shortest path problem, dijkstra's algorithm, Minimum spanning tree problem, Maximum flow problem.
- **Metaheuristics**: Basics of metaheuristics, Simulated Annealing with TSP as its case study, Tabu Search, Genetic Algorithm and its application to TSP as a case study, Application of metaheuristic on VRP.
- **Traffic Network Design Problems**: Covers Traffic assignment problem(TAP), Extensions of TAP like Discrete Network Design Problem(DNDP), Line planning.
- **Revenue Management**: Segmentation, Price differentiation, Models of Overbooking.
- **Packing logistics**: Fundamentals of Packing and its stages, Cutting Stock Problem, Knapsack Problem, Bin Packing Problem.
- **Container Shipping and Terminals**: Working of container terminals, Job-Shop and Flow-Shop Problems, Branch and Bound method, Usage of Gantt-Chart.
- Most of the above mentioned problems studied in transportation planning involved modeling them using Linear Programming and looking at their solving methods.

The course gave an introduction to many of the important problems observed in transportation science one of which is DNDP that the IDP deals with specifically. In relevance to the project, the course dealt with the necessary topics such as different kinds of metaheuristics, their algorithms followed by case studies and numerical problems to better understand these techniques. The bilevel formulation of DNDP was also introduced with TAP as its nested optimization problem and some of the techniques to solve them exactly or approximately.

A rough **timeline** from when the project started till its completion time is given below.

- Dec 2014 - Start of IDP with initial literature survey but main focus on the course.
- Feb 4, 2015 - Completion of the exam on the course **"TRANSPORTATION LOGISTICS"**.
- Feb 4, 2015 to April 14, 2015 - Further literature reading, setup, understanding and practical usage of the FICO Xpress software, learning to write code using BCL component.
- April 15, 2015 to July 31, 2015 - Implementation phase with good amount of testing. Further testing will follow on larger datasets with additional code to provide extensions.

- Aug 1, 2015 to Aug 30, 2015 - Documentation and Presentation
- Sep 2015 - Completion of IDP

1.3 Software Requirements

The project has been implemented in C with the help of an API(Linux version) provided by a commercial software suite for C++.

***FICO^{TM}* Xpress Optimization Suite:** FICO Xpress Optimization Suite is a sophisticated mathematical modeling and optimization software. Its tools enable operational research and management professionals, analysts, and consultants to rapidly develop optimization applications that solve complex, real-world business and customer engagement challenges. It provides easy ways to create, deploy and utilize business optimization solutions based on scalable high-performance algorithms, a flexible modeling environment and rapid application and reporting capabilities.

Optimization problems have to be solved computationally to good approximation and require the usage of sophisticated optimization softwares that can provide a library(API- Application programming interface) for the programmer to do the same. The programmer can use this to implement an application demonstrating the solving methods for DNDP. This is first done by modeling the DNDP in the form of a bilevel mixed integer linear program via the API provided by FICO Xpress BCL component and using its optimization modules to solve the nested optimization problem of TAP.

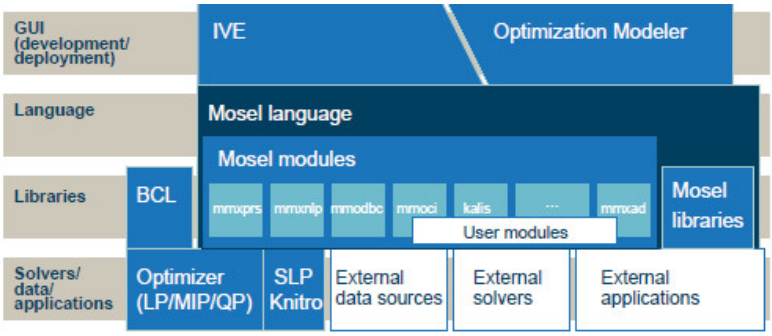

Figure 2: Xpress product suite

Libraries for embedding: An option available from this software for embedding mathematical models into large applications is by developing a model directly in a programming language with the help of a model builder library *Xpresss-BCL*. BCL(Builder Component Library) allows a user to formulate models with objects(decision variables, constraints, index sets) similar to those of a dedicated modeling language. All libraries are available for C, C++, Java, C#, and Visual Basic(VB). For this project we will be using the library for C++.

The Xpress-BCL Builder Component Library(BCL) provides an environment in which the Xpress user may readily formulate and solve linear, mixed integer and quadratic programming models. Using BCLs extensive collection of functions, complicated models may be swiftly and simply constructed, preparing problems for optimization.

Model formulation using Xpress-BCL is constraint-oriented. Such constraints may be built up either coefficient-wise, incrementally adding linear or quadratic terms until the constraint is complete, or through use of arrays of variables, constructing the constraint through a scalar product of variable and coefficient arrays. The former method allows for easier modification of models once constructed, whilst the latter enables swifter construction of new constraints. To complement the model construction routines, BCL supports a number of functions which allow a completed model to be passed directly to the Xpress-Optimizer, solved by the optimizer, and solution information reported back directly from BCL.

2 Modeling

2.1 Traffic Assignment Problem

Many economic systems can be visualized as networks where nodes stand for commodities, and links and paths stand for simple and complex production processes. The type of system which can be thus described in the most natural way is probably a transportation network. In this case the node stand for "cities", the link stand for roads directly connecting two cities, and the paths stand for roads connecting two cities directly or indirectly. A certain demand is associated with every pair of connected nodes of the network. This demand will be distributed among paths which join the pair of nodes. This gives rise to a traffic pattern, the determination of which is known as the ***traffic assignment problem***. Every link of the network is associated with a "traveling" cost which is assumed to be a function of the "traffic volume" on the link.

In some cases the traffic pattern can be regulated by some central authority, as for example, a network used for the transportation of military supplies or for a railroad network. It is obvious that in this case, the problem which the central authority faces is to determine the traffic pattern which minimizes the total cost over the whole network.

On, the other hand a broad class of transportation networks can be described as user optimized. Here travel patterns are set up by individual users each choosing the cheapest way(in the light of other user's decisions) to arrive at his/her respective destination, rather than having their travel pattern dictated by a choice consistent with some aggregate system optimum.

Out of the above 2 criterias, we can observe that the second criteria of user optimization problem can be reformulated as a total cost minimization problem for an appropriately chosen objective function. When viewed like this the problem then falls into the "multicommodity network flow" class. Literature says that for the case of linear cost(congestion) function, the problem reduces to a linear programming problem that can be solved fairly efficiently by the Dantzig-Wolfe decomposition algorithm. In case of non-linear objective function, its linear approximation is computed and Frank-Wolfe algorithm is used to solve the same. The following content presents the linear programming formulation of the traffic assignment problem.

Definitions:

- Set of Nodes N
- Set of links A with
  - capacity $\boldsymbol{c_a}$
  - congestion factor $\boldsymbol{B_a}$
  - free flow travel time $\boldsymbol{T_a}$
- Set of origins $\boldsymbol{R} \subseteq \boldsymbol{N}$ and destination $\boldsymbol{S} \subseteq \boldsymbol{N}$ with demand $\boldsymbol{q_{rs}}$

Decision variables:

- $\boldsymbol{x_a}$ travelers on link a
- $\boldsymbol{f_a}^{\boldsymbol{rs}}$ travelers of OD-pair (r,s) on link a

Constraints:

Travel time function on link a:

$$\boldsymbol{t_a(x) = T_a\left(1 + B_a\left(\frac{x}{c_a}\right)^4\right)}$$

Objective:

$$\boldsymbol{\min \sum_{a \in A} \int_0^{x_a} t_a(x)\, dx = \min \sum_{a \in A}\left(T_a B_a + \frac{T_a x_a}{5 {c_a}^4} {x_a}^5\right)}$$

Demand at origin:

$$\boldsymbol{\sum_{j \in N} f_{rj}^{rs} = q_{rs} \ \wedge \ \sum_{j \in N} {f_{jr}}^{rs} = 0 \quad \forall \mathrm{r} \in \mathrm{R}, \forall \mathrm{s} \in \mathrm{S}}$$

Demand at destination:

$$\boldsymbol{\sum_{i \in N} f_{is}^{rs} = q_{rs} \ \wedge \ \sum_{i \in N} {f_{si}}^{rs} = 0 \quad \forall \mathrm{r} \in \mathrm{R}, \forall \mathrm{s} \in \mathrm{S}}$$

Flow conservation:

$$\boldsymbol{\sum_{i \in N} f_{ik}^{rs} = \sum_{j \in N} {f_{kj}}^{rs} \quad \forall \mathrm{r} \in \mathrm{R}, \forall \mathrm{s} \in \mathrm{S}, \mathrm{k} \in \mathrm{N} \backslash \{\mathrm{r}, \mathrm{s}\}}$$

Flow aggregation:

$$\boldsymbol{x_a = \sum_{r \in R, s \in S} {f_a}^{rs} \quad \forall \mathrm{a} \in \mathrm{A}}$$

Non negativity:

$$\boldsymbol{f_a^{rs} \geq 0 \quad \forall \mathrm{r} \in \mathrm{R}, \forall \mathrm{s} \in \mathrm{S}, \forall \mathrm{a} \in \mathrm{A}}$$

2.2 Discrete Network Design problem

As described in section 1. the following content demonstrates the linear programming formulation of the *bilevel discrete-continuous network design problem* subsuming the *traffic assignment problem.*

Decision variables

- $\boldsymbol{x_a}$ travelers on link a
- ${\boldsymbol{f_a}}^{\boldsymbol{rs}}$ travelers of OD-pairs $\boldsymbol{(r, s)}$ on link a
- $\boldsymbol{y_a \ \in \ \{0, 1\}}$ if link is built 1, otherwise 0

Objective: Minimization of total travel time in the network.

$$\mathbf{min} \sum_{a \in A} t_a\left(x_a\right) x_a$$

Budget:

$$\mathbf{s.t.} \sum_{a \in A_2} b_a y_a \ \leq \ B$$

Additional flow restriction constraint for possible new routes in TAP

$$x_a \ \leq \ M_a \ \ \forall \mathrm{a} \in \mathrm{A_2}$$

Bilevel formulation:

$$\mathbf{min} \sum_{a \in A} t_a\left(x_a\right) x_a$$

$$\mathbf{s.t.} \sum_{a \in A_2} b_a y_a \ \leq \ B$$

$$\mathbf{min} \sum_{a \in A} \int_0^{x_a} t_a\left(x\right)$$

$$\mathbf{s.t.\ (TAP\ constraints)}$$

$$x_a \ \leq \ M_a \ \ \forall \mathrm{a} \in \mathrm{A_2}$$
$$y_a \ \in \ 0,1 \ \forall \mathrm{a} \in \mathrm{A_2}$$

Linearization of non-linear convex functions

Let $f\left(x\right)$ be an increasing, convex and non-linear function. Assume $m+1$ approximation points (ν_0, f_0), (ν_1, f_1),...,(ν_m, f_m) with $f_i := f\left(\nu_i\right)$. Further, $f\left(x\right)$ is a function in the single flow variable x_a and $v_m \geq \ \mathbf{max}_{a \in A}\, x_a$. The trivial upper bound is $\sum_{r \in R, s \in S} q_{rs}$. However, as x_a can be much smaller than $\sum_{r \in R, s \in S} q_{rs}$, empirical upper bounds can further improve the quality of the approximation. Define $a_i = \frac{f_i - f_{i-1}}{\nu_i - \nu_{i-1}}$ and $b_i := -v_{i-1} a_i + f_{i-1}$. Then $f\left(x\right)$ can be approximated by the following function:

$$\overline{f}(x) := \begin{cases} a_i x + b_i, & \text{for } x \in [\nu_{i-1}, \nu_i), i = 1, ..., m-1 \\ a_i x + b_i, & \text{for } x \in [\nu_{i-1}, \infty), i = m \end{cases}$$

It is clear that $a_i - a_{i-1} \geq 0$ and Nemhauser and Wolsey(1988) stated that no binary variables are needed for the approximation.

Instead $\overline{f}(x)$ can be minimzed by the following LP:

$$\min f_0 + a_1 x_1 + \sum_{i=2}^{m} (a_i - a_{i-1})\, x_i$$

$$\text{s.t. } x_1 \leq x_i + \nu_{i-1} \quad i = 2, ..., m$$
$$x_1 \geq 0 \quad i = 1, ..., m$$

As in each (ν_i, f_i) a new slope a_i starts, we have to add $(a_i - a_{i-1})\, x_i$ from that point on

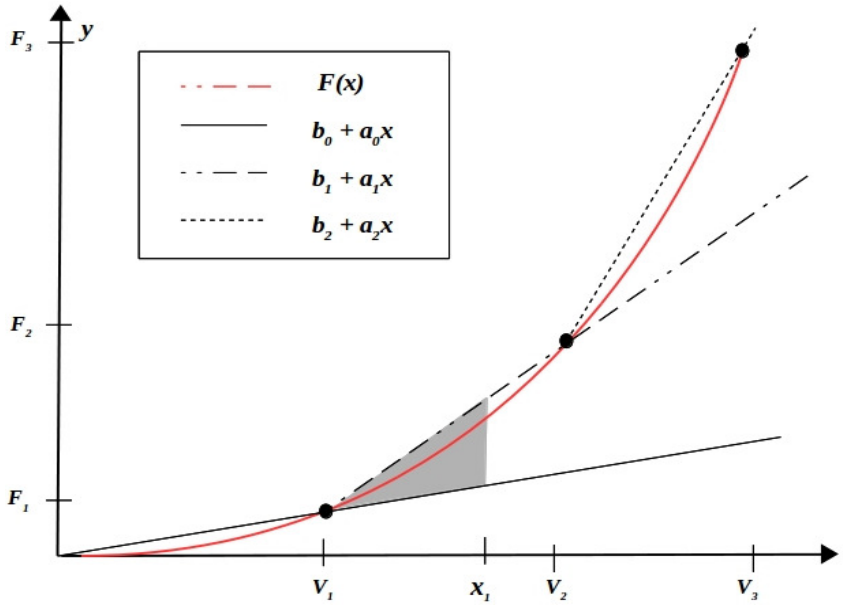

Figure 3: Example of an approximation of $F(x)$ with 4 data points

with $x_i = x_1 - \nu_{i-1}$, but do not subtract anything if $x_1 \leq \nu_{i-1}$. Because of the minimization problem and the definition of the objective function constraints, (51) and (52) (52) ensure that x_i takes the value of $\min\{0, x_1 - \nu_{i-1}\}$ and the defined optimization problem minimizes $f(x)$. In the example of Fig. 2, $x_1 \geq v_1$ and we have to add $(a_2 - a_1)\, x_2$ with $x_2 = (x_1 - \nu_1)$ (gray area), but $x_3 = 0$.

2.3 Solution Technique: Meta Heuristics

Modern heuristic techniques, also called metaheuristics are a family of procedures which benefit from some sort of intelligence in their search for finding the solution to a problem. It is a higher level procedure designed to find, generate, or select a heuristic(partial search algorithm) that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity. They may not provide optimal solution but they provide a sufficiently good solution rapidly and effectively. Simulated annealing, genetic algorithm, tabu search, neural network, ant system are some examples of such meta heuristics.

This project implements the genetic algorithm on DNDP. 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, phylogenetics, computational science, signal and image processing, Bayesian inference, machine learning, risk analysis and rare event sampling, Engineering and robotics, economics, manufacturing, mathematics, mathematical finance, molecular chemistry, computational physics, pharmacokinetic, pharmacometrics, and other fields.

3 Genetic Algorithm

Genetic algorithms are robust search and optimization techniques which are finding applications in a number of practical problems. The robustness of GAs is due to their capacity to locate the global optimum in a multimodal landscape. A plethora of such multimodal functions exist in engineering problems(optimization of neural network structure and learning neural network weights, solving optimal control problems, designing structures, and solving flow problems) are a few examples. It is for the above reason that considerable attention has been paid to the design of GAs for optimizing multimodal functions.

3.1 Understanding the algorithm

This is based on **"Genetic Algorithms"** by *David Goldberg, Addison Wesley, 1989.* 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.[6]

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. A generic pseudo code of GA is given below:

```
Initialize population P
Repeat
P' = { }
for i = 1 to n
   x := Selection of individual of P
   y := Selection of individual of P
   child := Crossover(x, y)
   if random(0,1) ≤ (probability of mutation) then
        child := Mutation(child)
   P' := P' ∪ child
end for
P := P'
```

The following 4 paragraphs explain the basic parts of a genetic algorithm.

Initialization: The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Often, the initial population is generated randomly, allowing the entire range of possible solutions (the search space). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

Selection: During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based

process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as the former process may be very time-consuming.

The fitness function is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always problem dependent. For instance, in the knapsack problem one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The fitness of the solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise.

In some problems, it is hard or even impossible to define the fitness expression; in these cases, a simulation may be used to determine the fitness function value of a candidate (e.g. computational fluid dynamics is used to determine the air resistance of a vehicle whose shape is encoded as the phenotype), or even interactive genetic algorithms are used.

Genetic operators: The next step is to generate a second generation population of solutions from those selected through a combination of genetic operators: crossover (also called recombination), and mutation.

For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated. Although reproduction methods that are based on the use of two parents are more "biology inspired", some research[7][8] suggests that more than two "parents" generate higher quality chromosomes.

These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions. These less fit solutions ensure genetic diversity within the genetic pool of the parents and therefore ensure the genetic diversity of the subsequent generation of children.

Opinion is divided over the importance of crossover versus mutation. There are many references in Fogel (2006) that support the importance of mutation-based search. Although crossover and mutation are known as the main genetic operators, it is possible to use other operators such as regrouping, colonization-extinction, or migration in genetic algorithms.[9] It is worth tuning parameters such as the mutation probability, crossover probability and population size to find reasonable settings for the problem class being worked on. A very small mutation rate may lead to genetic drift (which is non-ergodic in nature). A recombination rate that is too high may lead to premature convergence of the genetic algorithm. A mutation rate that is too high may lead to loss of good solutions, unless elitist selection is employed.

Termination: This generational process is repeated until a termination condition has been

reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of the above

A typical genetic algorithm requires:

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

The above paragraphs showed us the basic parts of a GA. Another important concern while using the approach of GA is about deciding how to represent the candidate.

A standard representation of each candidate solution is an array of bits.[6] Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming.

Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

3.2 Encoding

Fundamental to GA structure is the encoding mechanism for representing the optimization problem's variables. The encoding mechanism depends on the nature of the problem variables. For example, when solving for the optimal flows in a transportation problem, the variables(flows in different channels) assume continuous values, while the variables in a traveling salesman problem are binary quantities representing the inclusion or exclusion of an edge in the Hamiltonian circuit. In each case the encoding mechanism should map each solution to a unique binary string. A large number of optimization problems have real-valued continuous variables. A common method of encoding them uses their integer representation. Each variable is first linearly mapped to an integer is encoded in a specified range and the integer is encoded using a fixed number of binary bits. The binary codes of all the variables are then concatenated to obtain a binary string. For example consider a continuous variable defined in a range from $\mathbf{-1.28}$ to $\mathbf{1.28}$. We could encode this continuous variable with an accuracy of two decimal places by multiplying its real value by 100 and then discarding the decimal portion of the product. Thus the value that the variable attained is linearly mapped to integers in the range [-128, 128]. The

binary code corresponding to each integer can be easily computed.

Encoding of chromosomes is one of the problems faced during the usage of genetic algorithm for solving a particular problem. Encoding depends very much of kind of problem. The convergence of the genetic algorithm has close relation with its encoding method. Encoding method is the prime attention in design of the algorithm. Holland's schema theorem advocates the binary encoding gives the encoding rule of the minimum signs. Although binary encoding is simple and easy to do, it will add extra-calculated time for encoding and decoding to the algorithm. Also, when encoding real numbers, the binary coding will generate the encoding error which will decrease the precision of the algorithm. While real encoding can overcome the above problems and search a larger seach space.

Binary Encoding: Binary encoding is the most common, mainly because the first works about GA used this type of encoding. In binary encoding every chromosome is a string of bits, 0 or 1.

Chromosome A: 10110010110010101110010 1
Chromosome B: 11111110000011000001111 1

Binary encoding gives many possible chromosomes even with a small number of alleles. On the other hand, this encoding is often not natural for many problems and sometimes corrections must be made after crossover/or mutation.

Example of a problem: **Knapsack problem**
The problem: There are items with given value and size. The knapsack has given capacity. Select items to maximize the value of items in knapsack, but do not extend knapsack capacity.
Encoding: Each bit says, if the corresponding item is in knapsack.

Permutation Encoding: This type of encoding can be used in ordering problems, such as traveling salesman problem or task ordering problem. In permutation encoding, every chromosome is a string of numbers, which represents number in a sequence.

Chromosome A: 1 5 3 2 6 4 7 9 8
Chromosome B: 8 5 6 7 2 3 1 4 9

Examples of chromosomes with **permutation encoding:**
Permutation encoding is only useful for ordering problems. Even for these problems, for some types of crossover and mutation corrections must be made to leave the chromosome consistent(i.e. have real sequence in it).

Example problem: **Traveling Salesman Problem(TSP)**
The problem: There are cities and given distances between them. Traveling salesman has to visit all of them, but he doesn't wants to travel very much. Find a sequence of cities to minimize travelled distance.
Encoding: Chromosome says order of cities, in which salesman will visit them.

Value Encoding: Direct value encoding can be used in problems, where some complicated value, such as real numbers, are used. Use of binary encoding for this type of problems would be very difficult. In value encoding, every chromosome is a string of some values. Values can be anything connected to problem, form numbers, real numbers or chars to some complicated objects.

$$Chromosome A : 1.23245.32430.45562.32932.4545$$
$$Chromosome B : ABDJEIFJDHDIERJFDLDFLFEGT$$
$$Chromosome C : (back), (back), (right), (forward), (left)$$

Example of chromosomes with value encoding:
Value encoding is very good for some special problems. On the other hand, for this encoding it is often necessary to develop some new crossover and mutation operations specific to the problem.

Example of Problem: **Finding weights for neural network**
The problem: There is some neural network with given architecture. Find weights for inputs of neurons to train the network for wanted output.
Encoding: Real values in chromosomes represent corresponding weights for inputs.

Tree Encoding: Tree encoding is used mainly for evolving programs or expressions, for genetic programming. In tree encoding every chromosome is a tree of some objects, such as functions or commands in programming language. Example of chromosomes with tree encoding

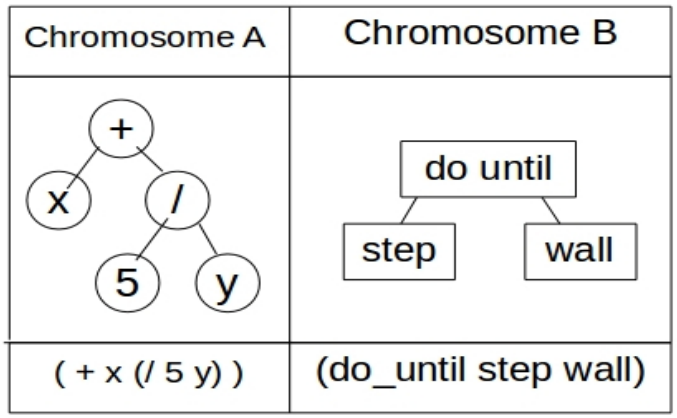

Figure 4: **Tree Encoding**

Tree encoding is good for evolving programs. Programing language LISP is often used to this, because programs in it are represented in this form and can be easily parsed as a tree, so the crossover and mutation can be done relatively easily.

Example of Problem: **Finding a function from given values**
The problem: Some input and output values are given. Task is to find a function, which will give the best (closest to wanted) output to all inputs.
Encoding: Chromosome are functions represented in a tree.

3.3 Selection

Genetic algorithms(GA) use a selection mechanism to select individuals from the population to insert into the mating pool. Worthy candidates are usually selected into the mating pool and the ones which are unworthy get eliminated. The hope is that this will lead to generate future

generations with desirable and better features/genes from these fitter candidates. Individuals from the mating pool are used to generate new offsprings with the resulting offsprings forming the basis of the next generation. As the individuals in the mating pool are the ones whose genes are inherited by the next generation, it is desirable that the mating pool be comprised of "good" individuals. A selection mechanism in GAs is simply a process that favors the selection of better individuals in the population for the mating pool. The selection pressure is the degree to which the better individuals are favored: the higher the selection pressure, the more the better individuals are favored. This selection pressure drives the GA to improve the population fitness over succeeding generations . The convergence rate of a GA is largely determined by the selection pressure, with higher selection pressures resulting in higher convergence rates. GAs are able to identify optimal or near-optimal solutions under a wide range of selection pressure[5]. However , if the selection pressure is too low, the convergence rate will be slow, and the GA will unnecessarily take longer to find the optimal solution. If the selection pressure is too high, there is an increased chance of the GA prematurely converging to an incorrect (suboptimal) solution.

- Random selection
- Fitness proportionate selection
- Rank selection
- Tournament selection
- steady state selection
- Truncation selection
- Local selection

Following content explains the first four of the above techniques in more detail.

3.3.1 Random Selection

This method does not follow any specific selection scheme. It selects parent chromosomes at random from the existing pool of candidate solutions. In this project, the implementation selects at random 2 candidates from the existing pool to follow up with crossover and mutation operations. This selection scheme is mainly intended for comparison purposes with other selection schemes such as the rank based selection and the tournament selection method. It allows us to observe the positive impact that the other selection schemes may have on the evolution of the population by using methods that influence the selection of candidates based on some criteria of fitness(weighted or absolute). The probability of selection of an individual $\boldsymbol{a_i}$ is:

$$\boldsymbol{P}\left(\boldsymbol{a_i}\right) = \frac{\mathbf{1}}{\boldsymbol{N}}; \quad \text{where } \boldsymbol{N} \text{ is the size of the population}$$

Each candidate is equally likely to be selected in this kind of a random selection scheme.

3.3.2 Fitness Proportionate Selection

This method is also known as ***Roulette wheel selection***. It is a genetic operator used in genetic algorithms for selecting potentially useful solutions for recombination. This selection principle is similar to that of a roulette wheel. The probability of selection of a sector in a roulette wheel is proportional to the magnitude of the central angle of the sector. Similarly in Genetic Algorithm, the whole population in partitioned on the wheel and each sector represents an individual. The proportion of individuals fitness to the total fitness values of the whole population decides the probability of selection of that individual in the next generation. This consequently decides the area occupied by that individual on the wheel. Following are the steps

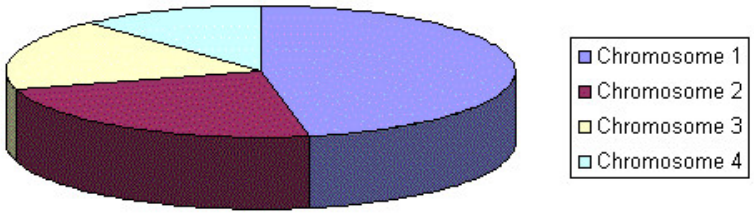

Figure 5: Roulette wheel selection

for Roulette Wheel Selection:

- Calculate the sum of the fitness values of every individual in the population.
- Calculate the fitness value of each individual and their probability of selection by dividing individual chromosomes fitness by the sum of fitness values of whole population.
- Partition the roulette wheel into sectors according to the probabilities calculated in the second step.
- Spin the wheel n number of times. When the roulette stops, the sector on which the pointer points corresponds to the individual being selected.

The probability of selection of an individual a_i is:

$$P(a_i) = \frac{f(a_i)}{\sum_{i=1}^{N} f(a_j)}; j = 1, 2, ..., n$$

where $f(a)$ refers to the function which gives us the fitness of this individual.

3.3.3 Rank Selection

Rank Selection in Genetic Algorithm was introduced by *Baker* to eliminate the disadvantages of fitness proportionate selection. In Linear Ranking selection method, individuals are first sorted according to their fitness value and then the ranks are assigned to them. Best individual gets rank N and the worst one gets rank 1. The selection probability is then assigned linearly to the individuals according to their ranks. The probability of the best individual to be selected is:

$$P(a_{best}) = \frac{N}{\frac{(N*(N+1))}{2}} \approx \frac{2}{N}$$

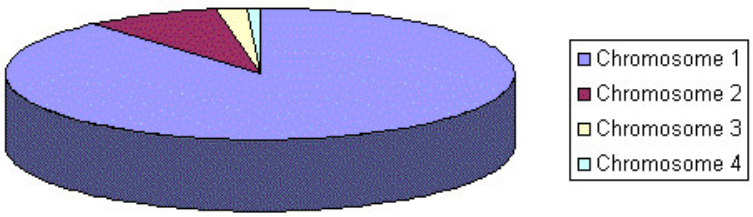

Figure 6: Situation before ranking

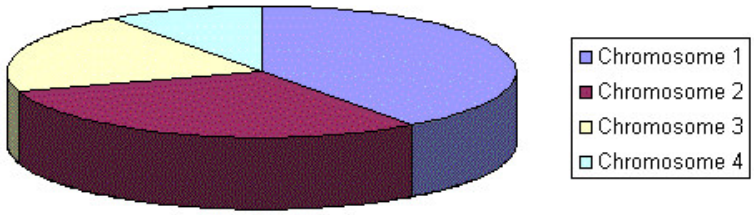

Figure 7: Situation after ranking

The probability of the worst individual to be selected is:

$$\boldsymbol{P\left(a_{best}\right) = \frac{1}{\frac{(N*(N+1))}{2}} \approx \frac{2}{N^2}}$$

The probability of the $\boldsymbol{i^{th}}$ individual(sorted order) to be selected is:

$$\boldsymbol{P\left(a_i\right) = \frac{N-i+1}{\frac{(N*(N+1))}{2}} \approx \frac{2\left(N-i+1\right)}{N^2}}$$

3.3.4 Tournament Selection

Tournament selection is a useful and robust selection mechanism commonly used by genetic algorithms (GAs). The selection pressure of tournament selection directly varies with the tournament size-the more competitors , the higher the resulting selection pressure. Due to the efficiency and ease of implementation, Tournament selection is the most popular selection technique of Genetic Algorithm. In Tournament Selection, $\boldsymbol{n}$ individuals are chosen at random from the entire population. These individuals compete against each other. The individual with the highest fitness value wins and gets selected for further processing in Genetic Algorithm. The number of individual taking part in every tournament is referred as tournament size. Most commonly used tournament size is **2** i.e. in Binary Tournament Selection. There are several advantages of Tournament selection strategy that makes it more efficient than other techniques. These include less time complexity i.e. $\boldsymbol{O\left(n\right)}$, easy parallel implementation, low vulnerability to takeover by dominant individuals, and no requirement for fitness scaling or sorting.

In the above figure, Tournament size is three, which means three individuals compete against each other in one tournament. The larger the tournament size, the greater is the probability of loss of diversity. There are two reasons for loss of diversity. Either the individual did not get the opportunity to be selected (because of random selection), or the individual didnt get selected in the intermediate population because they lost some tournament.

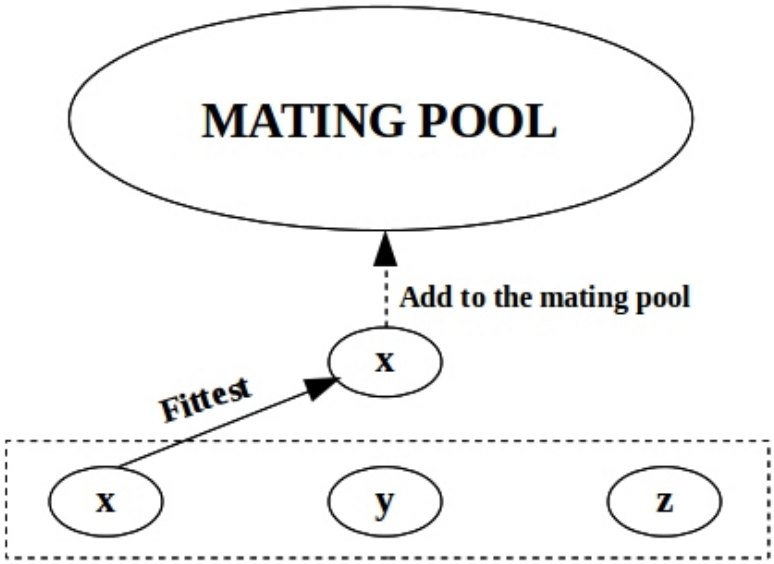

Figure 8: Tournament Selection

Elitist Strategy: Find the individual with the maximum fitness value in the mating pool and preserve it in the offspring. So the individual with the maximum fitness value in the previous generation can be kept. For this reason, the algorithm will be convergent to the global optimal solution with the probability of **1**.

3.4 Crossover

The search of the solution space is done by creating new chromosomes from old ones. Crossover is the technique by which the chromosomes selected from a source population according to a chosen selection scheme from those described earlier are combined to form offsprings which are potential members of a successor population. It is simply a matter of replacing some of the genes in one parent by the corresponding genes of the other. Pairs of chromosomes are randomly selected from the mating pool created using any one of the selection scheme. There are various ways to do a crossover. Few of them are described below. Examples demonstrating the various techniques use binary encoding as the underlying encoding scheme for the chromosomes.

For all the examples that follow, the below 2 parents will be used as the selected chromosomes:

$$\boldsymbol{Parent1 : 1010101010}$$
$$\boldsymbol{Parent2 : 1000010000}$$

Single Point Crossover: In this technique each pair of selected chromosomes undergoes crossover as follows: An integer position $\boldsymbol{k}$ along both the chromosomes is selected uniformly at random between **1** and the chromosome length say $\boldsymbol{l}$. Two new chromosomes are created swapping all the genes between $\boldsymbol{k+1}$ and $\boldsymbol{l}$. Suppose the randomly chosen crossover point is the fifth bit then each new child receives one half of the parent's bits:

Child 1: 10101**10000**
Child 2. **10000**01010

Two Point Crossover: This technique is similar to the single-point crossover except for the fact that there are **2** crossover points chosen at random for both the chromosomes. And, the parents only swap the bits between the **2** crossover points. Suppose the randomly chosen crossover points are the third and the sixth bit then the children are:

Child 1: 101**001**1010
Child 2: **100**010**0000**

Uniform crossover: This technique does not select a set of crossover points. It simply considers each bit position of the **2** parents, and swaps the two bits with a probability of **50%**. Suppose the first, third, fourth and ninth bit positions(of the original parents) are swapped. Then the children are:

Child 1: **1**0**00**101**0**0
Child 2: 1**0**10**010**0**1**0

Another slight variation of this technique is the half-uniform crossover where exactly half of the non-matching bits are swapped. First, the hamming distance(The number of differing bits) is calculated. This number is divided by two. The resulting number is how many of the bits that do not match between the two parents that will be swapped.

Three parent crossover: In this technique, The child is derived from three parents. They are chosen as per some selection scheme. Each bit of the first parent is checked with the bit of the second parent for equality. If they are equal then the bit is taken for the offspring otherwise the bit from the third parent is taken for the offspring. For example, the following three parents:

Parent 1: 110100010
Parent 2: 011001001
Parent 3: 110110101

Child: 110100001

Depending on how the chromosome represents the solution, a direct swap may not be possible. One such case is when the chromosome is an ordered list, such as an ordered list of the cities to be travelled for the traveling salesman problem. There are many crossover methods for ordered chromosomes. The already mentioned N-point crossover can be applied for ordered chromosomes also, but this always need a corresponding repair process, actually, some ordered crossover methods are derived from the idea. However, sometimes a crossover of chromosomes produces recombinations which violate the constraint of ordering and thus need to be repaired. Some of the techniques are given below:

- ***partially matched crossover (PMX):*** In this method, two crossover points are selected at random and PMX proceeds by position wise exchanges. The two crossover points give matching selection. It affects cross by position-by-position exchange operations. In this method parents are mapped to each other, hence we can also call it partially mapped crossover.[5]
- ***cycle crossover (CX):*** Beginning at any gene ***i*** in parent **1**, the ***i***-th gene in parent **2** becomes replaced by it. The same is repeated for the displaced gene until the gene which is equal to the first inserted gene becomes replaced (cycle).

- ***order crossover operator (OX1):*** A portion of one parent is mapped to a portion of the other parent. From the replaced portion on, the rest is filled up by the remaining genes, where already present genes are omitted and the order is preserved.
- Some other examples of techniques are as follows : ***order-based crossover operator (OX2)***, ***position-based crossover operator (POS)***, ***voting recombination crossover operator (VR)***, ***alternating-position crossover operator (AP)***, ***sequential constructive crossover operator (SCX)***

3.5 Mutation

The two individuals(children) resulting from each crossover operation will now be subjected to the mutation operator in the final step to form the new generation. This operator randomly flips or alters one or more bit values at randomly selected locations in chromosomes. The mutation operator enhances the ability of GA to find near optimal solution to a given problem by maintaining sufficient level of genetic variety in the population, which is needed to make sure that the entire solution space is used in the search for the best solution. In a sense, it serves as an insurance policy, it helps prevent the loss of genetic material.

The purpose of mutation in GAs is preserving and introducing diversity. Mutation should allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution. This reasoning also explains the fact that most GA systems avoid only taking the fittest of the population in generating the next but rather a random(or semi-random) selection with a weighting toward those that are fitter. Some of the type of mutation techniques used are as follows:

Random Mutation: This technique is applied to each offspring in the population with a predetermined probability. For a randomly chosen bit position of a chromosome we flip the bit to value either **0** or **1** assuming a binary encoding scheme.

Flip Bit: This mutation operator takes the genome and inverts the bits (i.e. if the genome bit is 1, it is changed to **0** and vice versa).

Boundary: This mutation operator replaces the genome with either lower or upper bound randomly. This can be used for integer and float genes.

Non-Uniform: The probability that amount of mutation will go to 0 with the next generation is increased by using non-uniform mutation operator. It keeps the population from stagnating in the early stages of the evolution. It tunes solution in later stages of evolution. This mutation operator can only be used for integer and float genes.

Uniform: This operator replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene. This mutation operator can only be used for integer and float genes.

Gaussian: This operator adds a unit Gaussian distributed random value to the chosen gene. If it falls outside of the user-specified lower or upper bounds for that gene, the new gene value is clipped. This mutation operator can only be used for integer and float genes.

Order Changing: In case of permutation encoding scheme, order changing is a technique where two numbers are selected an exchanged. Example as shown below:

(12 3 4 5 6 8 9 7) => (18 3 4 5 6 2 9 7)

Others: In case of value encoding, mutation is achieve by adding a small number (for real value encoding) - to selected values is added(or subtracted) a small number.

(1.29 5.68 2.86 4.11 5.55) => (1.29 5.68 2.73 4.22 5.55)

In case of tree encoding schemes, selected nodes are mutated by changing the operator or number.

The genetic algorithm will be used to solve the upper level problem whereas the nested problem is solved using BCL component of FICO Xpress software. For every candidate solution in the upper level problem the nested problem will be solved using BCL. This iterative process is continued to evolve towards stronger solutions for the leader by the use of genetic algorithm operations like crossover, mutation etc till the stopping criteria is satisfied.

4 Implementation

This section describes a high level design of the software implemented with the help of flow charts and pseudo code. Follwed by this is a small snapshot of certain low level details of the software that will help the reader to understand as to what functionalities this software offers in a summary.

4.1 Software Design

The following 3 pages give a high level design of this implementation.

- 9 shows a flow chart explaining the complete flow of the software implementation at a very high level. This flow chart does not expand in full detail the internals of the different selection schemes implemented or of the crossover or mutation implementation. This have been covered in the figures that follow this in the form of a pseudo code.
- 10 shows the pseudo code explaining the implementation details of the different selection schemes: Tournament selection and Rank selection(including how to assign selection probailities to the candidates).
- 11 shows the pseudo code explaining the implementation details of the Random Selection method and Crossover operation. Mutation operation has been omitted considering that it is very simple and straightforward.

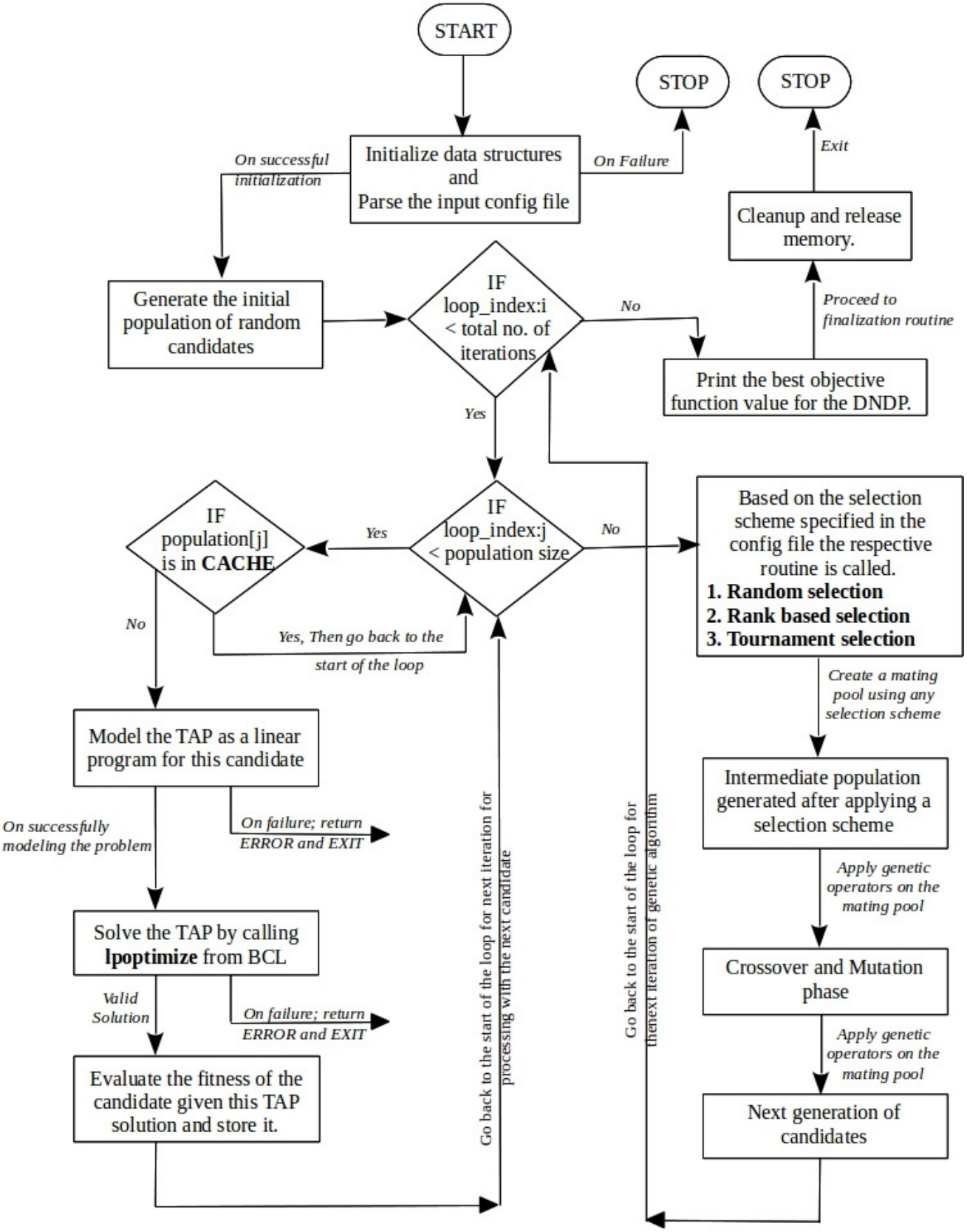

Figure 9: **Flow chart of the software implementation**

Algorithm: *Tournament Selection*

Input: *Original pool, Pool of offsprings, Network data, Size*
Ouput: *Next Generation*

Procedure: *Tournament Selection(original pool, pool of offsprings, network data, size)*
Start procedure
    Crossover(original pool, intermediate pool, pool of offsprings, network data, size)
    Mutation(pool of offsprings, network data, size)
    Copy the pool of offsprings to the original pool replacing the old population
End procedure

Procedure: *select_candidates_k*
Description: *Play a tournament with k random candidates and select the winner. Here the winner is the candidate with the best fitness(least objective function value).*

Algorithm: *Rank Based Selection*

Input: *Original pool(Sorted by fitness and having selection probabilities assigned), Pool of offsprings, Network data, Size*
Output: *Next Generation*

Procedure: *Rank Based Selection(original pool, pool of offsprings, network data, size)*
Start procedure
Label: Repeat till loop_index < size
    Index = select_candidates_rb(original pool, size)
    Add original pool[Index] to intermediate pool
    Jump to Label
    Crossover(original pool, intermediate pool, pool of offsprings, network data, size)
    Mutation(pool of offsprings, network data, size)
    Copy the pool of offsprings to the original pool replacing the old population
End procedure

Algorithm: *Assign Selection Probabilities*

Input: *Original pool(Sorted by fitness), Size*
Output: *Original pool of candidates with their selection probabilities*

Procedure: *assign_selection_rb_prob(original pool, size)*
Start procedure
    Fitness = size
    *total_fitness = (size * (size + 1)) / 2*
    Label: Repeat till loop_index < size
        original pool[loop_index].selection_prob = fitness / total_fitness
        fitness = fitness – 1
        Jump to Label
End procedures

Figure 10: **Pseudo code for algorithms**

Algorithm: Random *Selection*

Input: *Original pool(Sorted by fitness), Pool of offsprings, Network data, Size*
Ouput: *Next Generation*

Procedure: Random *Selection(original pool, pool of offsprings, network data, size)*
Start procedure
    Crossover(original pool, pool of offsprings, network data)
    Mutation(pool of offsprings, network data, size)
    Copy the pool of offsprings to the original pool replacing the old population
End procedure

Algorithm: *Crossover*
RAND_MAX := *Upper bound of the range from which a random number is generated by the library. This is a predefined constant in the standard C library.*

Input: *Original pool, Intermediate pool, Pool of offsprings, Network data, Size*
Output: *Next Generation*

Procedure: Crossover*(original pool, intermediate pool, pool of offsprings, network data, size)*
Start procedure
    random_value = random( ) / RAND_MAX
Label: Repeat forever
    IF the new offspring pool is filled completely THEN
        Break the loop
    END IF
    IF error_flag = 1
        random_value = random( ) / RAND_MAX
    END IF
    crossover points = random( ) {for the range between 1 and total number of new links}
    index1 = random( )
    Keep picking index2 = random( ) until it is not EQUAL to index1
    IF random_value > crossover probability THEN
        Do not crossover the 2 candidates at positions index1 and index2. Skip the rest of the loop and jump back to Label.
        random_value = random( ) / RAND_MAX
    END IF
    Do either a single point / two point crossover operation on 2 candidates at positions index1 and index2 in intermediate pool and store in temporary memory

    IF the above offspring is non zero AND
       the above offspring is budget feasible AND
       the above offspring is not a duplicate from original pool AND
       the above offspring is not a duplicate from the current offspring pool THEN
        Add this to the offspring pool
    ELSE
        IF MAX_ATTEMPTS reached THEN
            Set error_flag = 1 and copy over the best of the 2 parents to the pool
            Skip the rest of the loop
        ELSE
            Repeat the above process till MAX_ATTEMPTS by skipping the rest of the loop and jumping back to Label.
        END IF
    END IF
    Jump to Label
End procedure

Figure 11: **Pseudo code for algorithms**

4.2 Implementation Details

Optimization problem : ***Discrete Network Design Problem***

Algorithm implemented : ***Genetic Algorithm***

Optimization Suite(Software) : ***FICO Xpress***

Library for modeling/solving : ***Xpress BCL(Builder component library for C++)***

Programming Language : ***C***

Operating System : ***Linux***

Selection Schemes : ***Default, Rank based and Tournament***

Crossover operators : ***Single point , Two point***

Mutation operators : ***Random mutation, Bit flip mutation***

Datasets for testing : ***Braess network, Sioux falls network, Berlin mitte center***

Visualization of traffic network simulation : ***NexTA***

Range of budget for DNDP : ***10 - 180***

5 Experiments and Visualization

This section presents details about the system being used for carrying out the experiments, The kind of datasets or transportation networks widely known in literature that are being used for testing the genetic algorithm and their formats. Finally the results from performing these experiments on different networks under different selection schemes and budgets are presented in the form of a table with their run times.

5.1 Setup

Below are the hardware details of the system(obtained from the lscpu command on Linux) on which the experiments have been carried out followed by the software details(obtained using lsb_release -a command on Linux).

Architecture	x86_64
CPU op-mode(s)	32-bit , 64-bit
Byte Order	Little Endian
CPU(s)	8
Thread(s) per core	2
Core(s) per socket	4
Socket(s)	1
NUMA node(s)	1
Vendor ID	GenuineIntel
CPU family	6
Model	60
CPU Mhz	2925.828
L1d cache	32K
L1i cache	32K
L2 cache	256K
L3 cache	6144K

Table 1: Hardware Configuration

Distributor ID	Ubuntu
Description	Ubuntu 14.04.2 LTS
Release	14.04
Codename	trusty
Optimization Software	FICO Xpress Version 27.01.02
Library for modeling/solving	Xpress BCL Version 4.8.1
Compiler	GCC (GNU C/C++ Compiler) Version 4.8.4

Table 2: Software Configuration

5.2 Datasets

First dataset has been taken from [1] which is a small network as shown below.

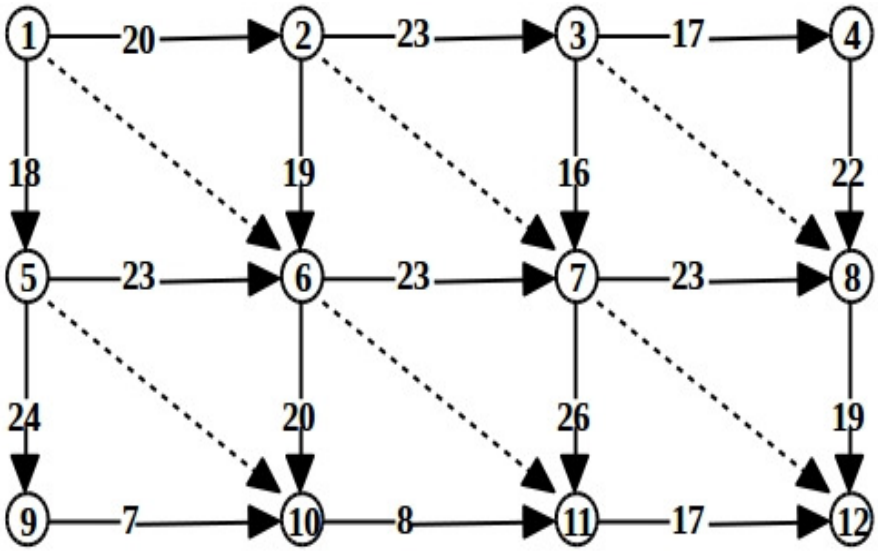

Figure 12: Solid lines indicate existing links and dashed lines indicate potential new links

Index	1	2	3	4	5	6
(i,j)	(1, 6)	(5, 10)	(2, 7)	(6, 11)	(3, 8)	(7, 12)
t_ij	19	25	30	32	21	28
Cost	7	12	7	15	11	18

Table 3: Small Network(s)

Following datasets for testing purposes have been taken from the `http://www.bgu.ac.il/~bargera/tntp/` that houses a repository of many such datasets.

5.3 Experiments and Results

5.4 Visualization

6 Conclusion

6.1 Scope for future work

References

[1] Gao, Ziyou., Wu, Jianjun, Sun, Huijun. Solution algorithm for the bilevel discrete network design problem. *Transportation Research(www.elsevier.com/locate/trb)*, July 2005.

[2] Fontaine, P., Minner, S. Benders Decomposition for Discrete-Continuous Linear Bilevel Problems with application to traffic network design. *Transportation Research(www.elsevier.com/locate/trb)*, September 2014.

[3] Hejazi, S.R., Memariani, A., Jahanshahloo, G., Sepehri, M.M. Linear bilevel programming solution by genetic algorithm. *Computers and Operations Research(www.elsevier.com/locate/dsw)*, July 2000.

[4] Xu, Tianze., Wei, Heng., Hu, Guanghua. Study on continuous network design problem using simulated annealing and genetic algorithm. *Expert Systems with Applications(www.elsevier.com/locate/eswa)*, March 2009.

[5] LeBlanc, Larry J., Boyce, David E. A bilevel programming algorithm for exact solution of the network design problem with user-optimal flows. *Transportation Research(www.elsevier.com/locate/trb)*, June 1986.

[6] Kuo, R.J., Huang, C.C. Application of particle swarm optimization algorithm for solving bi-level linear programming problem. *Computers and Mathematics with Applications(www.elsevier.com/locate/camwa)*, February 2009.

[7] Poorzahedy, Hossain., Rouhani, Omid M. Hybrid meta-heuristic algorithms for solving network design problem. *Transportation Research(www.elsevier.com/locate/trb)*, July 2006.

[8] Wen, U.P., Huang, A.D. A simple Tabu Search method to solve the mixed-integer linear bilevel programming problem. *European Journal Of Operations Research*, February 1996.

[9] Olaf Jahn, Rolf H. Mhring, Andreas S. Schulz, Nicolas E. Stier-Moses. System-Optimal Routing of Traffic Flows with User Constraints in Networks with Congestion. *Operations Research, INFORMS*, August 2005.

[10] Marek Obitko. Introduction to Genetic Algorithms - Tutorials with interactive java applets. http://www.obitko.com/tutorials/genetic-algorithms.

[11] Prof. C. Balaji. Design and Optimization of Energy Systems - Lecture no. 38. http://nptel.ac.in/courses/112106064/38.

[12] Hillel Bar-Gera. Transportation Network Test Problems. http://www.bgu.ac.il/ bargera/tntp/.

[13] Genetic Algorithms. https://en.wikipedia.org/wiki/Genetic_algorithm.