Bike Sharing Network Design with Service Levels: The Case of Montreal City

Mehdi Khatib

A Thesis

in

The Department

of

Mechanical Industrial and Aerospace Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Industrial Engineering) at

Concordia University

Montreal, Quebec, Canada

December 2021

© Mehdi Khatib, 2021

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By:	Mehdi Khatib
Entitled:	Bike Sharing Network Design with Service Levels: The Case of Montreal City

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science in Industrial Engineering

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Daria Terekhov	Chair
Dr. Navneet Vidyarthi	Examiner
Dr. Daria Terekhov	Examiner
Dr. Onur Kuzgunkaya	Supervisor

Approved by _____

Graduate Program Director: Dr. Sivakumar Naranswamy

_____2021 _____

Dean of faculty: Dr. Mourad Debbabi

Abstract

Bike Sharing Network Design with Service Levels: The Case of Montreal City

Mehdi Khatib

The rapid growth of urbanization and use of motor vehicles in the recent decades has led to many social and economic problems such as: rising fuel prices, energy crises, environmental problems and traffic congestion. All these problems together have decreased the quality of life of people all around the world. In recent years, municipal planners have increasingly focused on extending policies to promote a culture of using bicycles instead of cars. In many cases, urban planners try to build the infrastructure needed to increase the usage of bicycles and one of the measures that has been widely used by them in recent years is bike sharing programs. In this study, we design a bike sharing network considering the objectives of users and system designers simultaneously. From the customers' view point, walking short distances before picking up and after dropping off a bike would be a preference and they will be satisfied when they find available bikes or empty docks in the system. From the system designer's perspective, the objective is to achieve these service levels with the minimum network design cost. To achieve this, we develop a mixed integer linear programming model to minimize the cost of opening stations and transportation costs. We consider the pickup and drop off service level constraints in determining the location, dock capacity and demand allocation to the bike stations. A Mixed Integer Linear Programming model is developed and solved using CPLEX Software. In order to validate the network design solutions, we simulate the results of small to medium size instances in Arena. To solve the larger instances of the problem, a Genetic Algorithm is proposed that uses a heuristic method to generate a part of initial solutions and improves the solutions in its stochastic iterations and reaches near-optimal solutions in a reasonable amount of time. The proposed method is illustrated using the city of Montreal as case study.

Acknowledgement

First and foremost, I would like to thank my thesis supervisor, Dr. Onur Kuzgunkaya for his understanding, encouragement and assiduous commitment to the highest standards which inspired and motivated me during my studies. I learnt a lot from his knowledge, wisdom and intelligence.

Moreover, I am deeply indebted to my compassionate dawshi, Aria Azami for his unlimited kindness and support. I want to thank all my dawshis: Hamid, Hamed, Babak and Erfan.

I also wish to thank my family, especially my parents, who have always provided me with such a pure love. I am blessed to have such great brothers for bringing joy to my life. Finally, I would like to thank my beloved wife for believing in me and her love.

Dedicated to my Parents

Table of Contents

List of Figures

List of Tables

Chapter 1 - Introduction

The rapid growth of urbanization and use of motor vehicles in the recent decades has led to many social and economic problems such as: rising fuel prices, energy crises, environmental problems and traffic congestion. All these problems together have decreased the quality of life of people all around the world (Pucher et al. 1999). In recent years, municipal planners have increasingly focused on extending policies to promote a culture of using bicycles instead of cars. The more use of less polluting transportation modes creates a sustainable mobility in the city. In addition to its social benefits such as environmental and economic sustainability, it also has many benefits at the individual level. Cycling can be a fun way to travel, while it can minimize the hassle of using motor vehicles. Since cycling is healthy and an economic transportation mode, it can be more efficient in big city centers compared to private cars and public transportation (Zahedian-Tejenaki and Tavakkoli-Moghaddam 2015, Heinen et al. 2010).

Nowadays, in most developed European cities, having a plan to encourage people to travel by bike is one of the most important issues in any political parties to succeed in elections. In many cases, urban planners try to build the infrastructure needed to increase the usage of bicycles and one of the measures that has been widely used by them in recent years is bike sharing programs (Stinson and Bhat 2005, Midgley 2011). The first public use of a bike sharing system took place in Amsterdam in 1965, known as the white bicycle system. Since then, these systems have been expanded greatly and many models have been developed for them (Bonnette 2007).

The development of bicycle sharing systems in large cities helps to create stability in transportation and public systems, and in addition, it can be used to enable citizens to travel to places in the city or pass some ways that cannot be accessed through other vehicles. The main components of a bike sharing system include bicycles, bike stations and riders. In these systems, rider picks a bike up from a station (origin of the trip), and after a period of time, returns it to the same station or another station in the network (the destination of the trip) (Büttner et al. 2011, Caggiani and Ottomanelli 2013).

However, bicycle sharing systems have some limitations, including the fact that bicycles are often used for short to medium trips and are mostly used for one-way trips. This issue can disrupt the equilibrium of the system over a period of time and in some areas of the network. Therefore, in order to improve the system and increase the users' satisfaction, locating stations in the network

and considering appropriate fleet size is very important (Caggiani and Ottomanelli 2013). Today, it is estimated that there are more than 375 bike sharing networks all around the world and there are about 240,000 bicycles in them. One of the most important factors in the success of these programs is how stations are spread in the bike sharing networks and in what level they satisfy the demand. In order to increase the utilization of bike stations, they should not be too far apart, and the distances in the network must be suitable for traveling by bicycle (Lin and Yang 2011, Shu et al. 2010).

Due to the fact that these systems are designed for public use, the stations should be close to recreational areas, tourist attractions and major commercial centers. Another important factor in locating bicycle stations is its connection to the public transportation system. Actually bicycle can be used as a complement to the transportation system and increase the coverage of public transportation systems. A designer should be aware of the potential distribution of demand at different zones and prioritize the areas that produce more trips (Martens, K. 2007, García-Palomares et al. 2012).

1.1. Goal of the study

In this study, we design a bike sharing network considering the objectives of users and system designers simultaneously. From the customers' view point, walking short distances before picking up and after dropping off a bike would be a preference and they will be satisfied when they find available bikes or empty docks in the system. From the system designer's point of view, the objective is to maximize the revenue with lowest network building cost. This can be achieved by building the infrastructure with the lowest investment while satisfying bike pick up and drop off service levels. In this research we combine both points of view considering the impact of service level of pick up and drop off on all components of the system. Actually a bike sharing system usually provides two types of service to users. First is the availability level to pick up a bike from a specific station which is called service level of pick up, and second is the availability level of docks for users arriving to drop off a bike in the station, which is called drop off service level.

Our objective is to minimize the cost of opening stations and transportation costs, and there are two main decisions in the problem under this study. First the location decision which is affected by the cost of building stations, demands and distances between demand zones and stations. The second one is the allocation decision which allocates the demand of each zone to those stations which are more cost efficient than others. Also aforementioned service level rates have an

important role in this model and affect the whole allocations in the network and the capacity and number of initial bikes in each station.

The city of Montreal is considered as the case study and the data for the experiments is obtained from Bixi's website, bike sharing network of Montreal. Actually, all information needed to fulfill our numerical experiment is extracted from the statistics of the previous years of Bixi company.

A Mixed Integer Linear Programming model is developed for the designed bike sharing network and CPLEX Software with an underlying branch and bound algorithm is used to solve the model. In order to validate the network design solutions and by the help of ARENA Simulation Software, we simulate the result of some experiments in small to medium sizes out of CPLEX.

In order to solve the model in large scales, a Genetic Algorithm is proposed that uses a heuristic method to generate a part of initial solutions and improves the solutions in its stochastic iterations and reaches near-optimal solutions in a reasonable amount of time.

1.2. Research contributions

While previous studies in the literature of the problem are mostly categorized as a bike sharing network design or closed queuing network independently, we integrate these two major problems together and provide a mixed integer optimization model which not only deal with a capacitated location-allocation problem, but also consider the service level of pick up and drop off in all stations. To the best of our knowledge, most articles using queuing models, have studied a single bike station or have concentrated on the service level of pick up in a closed queuing network problem, whereas we incorporate both service levels in a bike sharing network problem. The proposed mixed integer programming model cannot handle large size instances due to the computational complexity of the problem. We propose a genetic algorithm method to deal with instances in large scales and use a heuristic to generate good initial population in assisting the solution methodology to find near-optimal solutions as fast as possible.

1.3. Outline of the thesis

The thesis is organized into five chapters. Following the introductory chapter, we review the related literature in the second chapter. Chapter three begins with a description of the problem and is followed by the MILP model formulation. The service level function is described in detail and the solution method for large size problems is also presented in this chapter. In chapter four, first

we describe the setting of the numerical experiments followed by the sensitivity analysis of the main parameters on the results. We then validate the design solutions with simulation and study the performance of the proposed genetic algorithm by presenting the computational results in small to large scale instances. The results are also analyzed in this chapter. Finally, chapter five provides conclusions and future research directions.

Chapter 2 - Literature Review

The literature in bicycle sharing systems include different types of decision framework, from strategic to operational levels. Strategic decisions provide a design of the bike sharing network and define the main policy such as location and capacity of stations. Operational decisions are taken daily or weekly basis and adjust the system to increase its efficiency. As the scope of research under this study is at the strategic level and also service level has been considered as a main factor in designing a bike sharing network, we categorize the previous studies and articles into two groups and review them in detail.

2.1. Network Design

In order to stablish a bicycle sharing system and to increase its efficiency, many variables and parameters must be taken into account at the strategic level. All decisions related to the number of bike stations and their locations, capacity of each station, total fleet size and number of bikes in each station, allocation of customers and routing allocations can be considered as strategic decisions. In this section we review articles that are focused on these aspects and propose strategic models to design a bike sharing network.

Lin et al. (2013) designed a bike sharing network with the assumption of unlimited bicycle stocks per each station. The main decisions of their model are number of bike stations and their location in the network, building the bike lanes and paths selection between demand zones. They defined the problem as a set of origin and destination demand zones and potential bike stations to be opened and measured the rental availability rate of bicycles for pick up and the coverage of demand between zones. Their objective was to optimize the cost of opening stations (which includes the number and inventory of stations) and the traveling cost of customers. In addition, they determined a penalty for uncovered demands between each pair of origins and destinations, and in this way they wanted to reduce the number of problematic stations in the network (problematic stations are those facing shortage in available bikes or empty docks). Although they just considered the problem of pick up side.

Due to the complexity of the problem the authors could not provide exact solutions for practical situations. To overcome this issue, they proposed a heuristic method for finding near-optimal solutions for larger sizes. The authors extended a greedy heuristic where all stations and bike lanes to be open initially and computes the total costs of the network. Then it identifies the costly stations

and lanes one by one in subsequent steps, and close one of the most expensive stations or lanes in each step to make as large reduction as possible in the total costs. The algorithm repeats these steps until a maximum number of iterations is reached or improving the objective function becomes impossible. At the end, by doing a sensitivity analysis they realized the important parameters affecting the inventory holding decision and routing selections.

Liu et al. (2019) studied a bike network design by assuming the bicycle routes to be separated from the main roadways. They wanted to optimize the utilization of bike routes and study the behavior of cyclists' paths selection. They assumed that all roadways are appropriate to construct a bicycle path, the demand for bike is fixed and the users select the roads based on the travel conveniences (such as the existing facilities for bikes or road slopes). They also considered a limited budget for constructing the bike network and used a model to make the bicycle path selection behavior of customers more realistic.

The authors designed a mixed-integer nonlinear model and then solved it with a global optimization method and a novel math-heuristic. The idea of global optimization method was to linearize the main model using some techniques in order to guarantee an exact solution near the global optimum. Whereas the proposed math-heuristic method has been used to improve the efficiency of solutions for large scale instances. Actually, the procedure of this method was to embed a surrogate-model–based heuristic in the global optimization method and by updating the feasible regions, convert the original model and make some estimations. In this way, an approximation is provided in each iteration and then is evaluated in the first solution method. The computational results verified the performance of the proposed method as well.

Frade and Ribeiro (2015) introduced a bike sharing network that combines strategic and operational decisions. The strategic decisions include the location of stations and its capacity, and number of bikes, while the operational decisions include how bicycles should be relocated in the system. Their objective was to maximize the coverage of demands and to this end they considered a limited budget and the quality of service. Their model also considers the revenue and yearly costs of the system and helps the investor to build the network. The city of Coimbra in Portugal was selected to apply the model and the results of case study illustrated the performance of the system. It should be noted that, they tried to make the demand zones as small as possible (in terms of area) and the proposed model just allocates stations per each zone (it does not locate the stations accurately). They reminded that in order to find the exact location of stations in each demand zone, the proposed method should be coordinated with a model that minimizes the distance between demand zones and stations.

Nair and Miller-Hooks (2016) proposed a bicycle sharing network in Washington, D.C. which was focused on locating stations across the city and configuring the optimal fleet size in the system. By having fixed demand, a set of potential stations, an existing transit network, resource constraints and users' behavior as assumptions, their goal was to maximize the utilization of the system (the flow) and using a bi-level mixed-integer programming formulation with non-convex feasible area. In order to solve the model for large scales, they developed a Genetic Algorithm as solution method using the idea to decouple the lower-level flow variables from the upper-level design variables and solving the upper-level problem independently. The benefit of decoupling these two levels was that the problem could be broken down by origin-destination pairs or just destinations. The results verified that the configured network could integrate with existing transit system properly, and also travel times and system usage improved significantly. The analysis of results provided information of flow between stations as well.

Yan et al. (2017) designed four bike rental networks using deterministic and stochastic demands. In order to minimize the total cost of system and based on a time-space network, a deterministic and a stochastic bicycle location and allocation model (DBLAM and SBLAM) were formulated. Their focus was on locating the potential stations, fleet size and route allocations, and they proposed a mixed-integer programming model. The other two models were the conversion of DBLAM and SBLAM to a maximal service level objective (in order to increase the number of bicycle rental requests).

The two deterministic models were solved then within a reasonable time in CPLEX, however the stochastic models were solved using a threshold-accepting-based heuristic. The heuristic method used a two phase algorithm that first identifies the set of open stations, and then allocates bikes and routes to the open stations. The threshold value was defined as a factor multiplied by the fixed cost of opening stations, and was used to decrease the solution area and force the problem to search among the solutions having fitness values less than the threshold. The authors performed computational tests using the open data of New Taipei City bicycle program and the results verified the model and the performance of their solution method.

Alizadeh et al. (2019) proposed a stochastic model for a capacitated location-allocation problem and assumed the demand to be Bernoulli distributed. Their objective function was to minimize the total cost of facilities, customers' allocations, outsourcing and anticipated service, simultaneously, by finding the locations of facilities and allocation of users optimally. They assumed that the additional demands for facilities can be satisfied using the outsourcing recourses. They first solved small scale instances by normal distribution approximation. Then in order to solve medium to large

instances, and using the discrete colonial competitive algorithm (DCCA), they extended the DCCA and proposed a new solution approach called EDCCA. The main idea of EDCCA was based on a vectorization technique, which substitutes the scalar-oriented and loop-based code in DCCA by adding matrix and vector operations. At the end, the computational results showed the proficiency of proposed methods and verified that the EDCCA obtained better results in terms of time and accuracy, compared to DCCA method.

2.2. Service Level

One of the most important issues we are facing in a bike sharing network is service level. Service level in a bike sharing network can be defined in two ways: the probability of finding an available bike to pick up and the probability of finding an empty dock to return a bicycle. In this section we review papers that study formulation of these criteria and investigate the problematic bike stations. Problematic stations are those that users face a shortage either in egress side to find any slot to drop a bicycle off, or in access side to pick up a bike. In the literature researchers addressed the modeling of service level using the following methods: open queuing network, closed queuing network, mean-field method, and Markov decision process.

George and Xia (2011) studied a closed queuing vehicle network for finding the optimal fleet size and also they considered the availability of vehicles in stations in their model. Vehicles in their research could be bicycle or electric cars. The presented closed queuing network model was extended from the viewpoint of the vehicles. For achieving some principles in the system as network balancing methods, they presented a framework that considering the fleet size, could derive an asymptotic behavior of vehicle availability in an arbitrary station. They measured the quality of service by the vehicle availability which is defined by the percentage of passengers who find a vehicle upon their arrivals. They assumed the customer leaves the network without service if he finds no vehicles available upon arrival and also they assumed unlimited parking space at each station for dropping the vehicles off. In their study vehicles are waiting for arrival of customers to be served and view each station as a single server. Indeed, they considered each rental station as M/M/1/T queue with state-dependent Exponential arrival rate and service rate.

The class of network which they presented is BCMP network which has product-form solutions and according to the steady-state probability that they presented, they reach the queue length and actual throughput. They then presented a profit-based mathematical model to maximize the revenue per-unit-time and considered the maintenance cost and penalty cost of vehicles

unavailability in the objective function as well. There are two ways for obtaining the mean performance measures of network as convolution method and mean value analysis. George and Xia developed mean value analysis (MVA) because they wanted to direct the primary performance measures and also they used the Schweitzer–Bard MVA approximation method because the MVA was computationally expensive for the large number of fleet sizes. Kochel et al. (2003) used a closed queueing network model for fleet sizing and allocation problem as well. They presented a steady-state optimization formulation and their goal was to optimize the fleet size and vehicle repositioning jointly. In order to solve the joint problem, they developed a simulation optimization method and focused on an iterative method to reach appropriate solutions for the fleet-sizing problem.

Li et al. (2016) presented a unified framework for analyzing the closed queuing bike sharing network with multi-class of customers and defined some virtual costumers and nodes. They tried to achieve the product-form solution to the steady state joint probability of queue lengths and give performance analysis of the bicycle sharing network. To this end, they assumed that by having a bike available in the origin station, the user picks it up and chooses a road with a specific probability according to the routing matrix, otherwise the customer leaves the system. On the other side if there is an empty slot in the destination station, the bike is returned instantly, otherwise the user select another route with a specific probability from the routing matrix and this pattern could be repeated successively to finally find a space to drop the bike off. The traveling time and the road selections could be different from others.

Li et al. designed their bike-network with N different stations and the number of roads to be equal to N*(N-1) at most. The arrival of the customers is considered as a Poisson process with homogenous rate and the traveling time has exponential distribution function. Despite the physical attribute and functionality of routes and stations, they considered both of them as virtual nodes and by viewing the system from the bicycles perspective, they assumed the bicycles as virtual customers as well and divided them to two classes. The first class were those that riding in the system for the first time and the second group were those riding in the routes for at least two times successively. The service disciplines for stations and roads were FCFS (first come first service) and PS (processor sharing), respectively. In order to analyze the presented system, authors obtained a unified framework and by calculating the service rates, routing matrices and the relative rates of the closed queuing network, they provided a product-form solution to the steady state joint probability of N*(N-1) queue lengths.

Celebi and Isik (2018) designed an integrated bike sharing network to minimize the unsatisfied demands for pick up and drop off by locating bike stations and capacity allocation. The method that they used was combination of set-covering model to assign demands to stations with queue model for measuring the service level. They assumed a specific capacity for stations to address the uncertainty issue for demand satisfaction in picking and returning sides and applied their model in the area of Istanbul Technical University. To this end, they estimated the demand for bikes using an approximation method based on a survey prior to the design of the network in the university. They then claimed that according to their results, in practical the number of relocations would decrease. Their mathematical model minimizes the probability of having problematic stations by having capacity limitation and traffic intensity as some constraints. And by using queue theory and a dynamic calculation of bicycle pick-up and return rates, they estimated the unsatisfied demand in the university. The notable key point is that authors did not consider any cost factor in their analysis to avoid some misleading results caused by cost structures. Also it is remarkable that, as their model has a non-linear form and they used a dynamic programming-based algorithm to solve their model, they weren't able to apply it for larger sizes like big cities.

Li et al. (2017) designed a practical bicycle sharing network system to be able to calculate the probability of the full or empty stations (problematic stations). To this end, they assumed the users' arrival to follow Markovian process which illustrates that their arrival is heterogeneous in terms of time and space in practice. Also considering the geographical structure of the bike network, users ride in an irreducible path graph which is directed by N different stations and N-1 straight roads. In order to record the dynamic position of bikes, they used the concept of virtual nodes for roads and stations. Also they imagined each bike as a virtual customer in a multi-class closed queueing network. So by these assumptions passengers' arrival can be considered as the service time of station nodes and riding bikes on routes as the service time of road nodes. Then in order to make a routing matrix, they gathered data by observing the bike sharing system's physical behavior, and to calculate the relative rate of arrivals they introduced a nonlinear solution using the obtained routing matrix. The steady state probabilities of joint queue lengths in virtual nodes were computed using the product-form solution as well.

Fricker and Gast (2014) studied the effects of users' random choices on the number of problematic stations and presented a stochastic model with homogeneous scenario for bike sharing network. In their model the influence of stations' capacities is calculated, and in order to minimize the percentage of the problematic stations, the optimal fleet size is computed. In their assumption there are N stations in the network and each station has K bikes. So, the total fleet size is equal to N*K

and each bike can be served in a station or in a road between stations. They consider incentives and redistribution by trucks for the framework that they presented. Customers arrive to pick up a bike at each station with Poisson rate and the traveling time is distributed exponentially. For returning the bicycle, each customer chooses a destination randomly and rides towards there. In the case of finding an empty dock the bike is dropped off there, otherwise that station is called saturated and the user opts another station randomly and this procedure repeats until an unsaturated station be found.

At first authors modeled the simplest version of the problem without considering any incentives or redistributions and the system showed a poor performance. So then, they set an incentive for passengers to return their bikes to the least loaded station among two stations and the performance was improved with an exponential factor. They mentioned that even if a fraction of customers follows the incentive, the performance metric changes significantly. Also, they designed a situation that trucks redistribute a specific rate of bikes for insuring a given service quality and this redistribution rate was dependent to the fleet size and stations' capacities. For the verification of their model, they investigated different trip-time distributions and then simulated them and compared the results. They also added the geometry to their model and studied the influence of it on the proposed model. At the end the mean-field approximation methodology was used to gain the asymptotic behavior of the model when the system size grows up.

2.3. Conclusion

In this section we overview the main investigated articles in network design and service level, and then explain the features of our research and its differences with previous studies.

Table 1. Summary of main articles in network design

Authors	Problem	Objective	Solution Method
Lin et al. (2013)	Bicycle sharing network design	Minimizing the total cost	Heuristic
Liu et al. (2019)	Bicycle sharing network design	Maximizing the bike path utilization	Global optimization and Math-heuristic
Frade and Ribeiro (2015)	Bicycle sharing network design	Maximizing the demand coverage	Heuristic

Nair and Miller-Hooks (2016)	Bicycle sharing network design	Maximizing the utilization of system (the flow)	Genetic algorithm
Yan et al. (2017)	Bicycle sharing network design	Minimizing the total cost	Heuristic and Threshold-accepting-based method
Alizadeh et al. (2019)	Capacitated location-allocation problem	Minimizing the total cost	Normal approximation and EDCCA

Table 2. Summary of main articles in service level

Authors	Problem	Objective	Solution Method
George and Xia (2011)	Closed queuing vehicle network	Maximizing the vehicles availability	Mean value analysis
Li et al. (2016)	Closed queuing bike network	Minimizing queue length	Product-form solution
Celebi and Isik (2018)	Closed queuing bike network	Minimizing the unsatisfied demands	Programming-based algorithm
Li et al. (2017)	Closed queuing bike network	Minimizing the number of problematic stations	Product-form solution
Fricker and Gast (2014)	Closed queuing bike network	Minimizing the percentage of problematic stations	Mean-field approximation

Previous studies in bicycle sharing system are mostly considered as a bike sharing network design or closed queuing network independently, while we have integrated these two major problems together and have provided a mixed integer optimization model which not only propose a capacitated location-allocation problem, but also consider the service level of pick up and drop off. To the best of our knowledge, most articles with Markovian process, have concentrated on the service level of pick up or have studied a single bike station satisfying both service levels, where we have provided a model integrating both service levels in a bike sharing network problem.

In the next chapter, at first the problem is defined and the mathematical model is presented, then the service level function is described and the solution methodology is explained in detail.

Chapter 3 - Problem Statement and Methodology

In this study, we design a bike sharing network considering the service level of pick up and drop off bikes in all stations. Given a set of origin and destination points for travelers and the amount of demand in each origin point, the goal is to determine the locations and sizes of bike stations such that it minimizes the cost of opening stations and transportation costs. In this model it is crucial to know where to locate the bicycle stations, which routes should be selected considering the distances from demand zones to stations and the capacity level of each station to serve with a desired level of bike and dock availability.

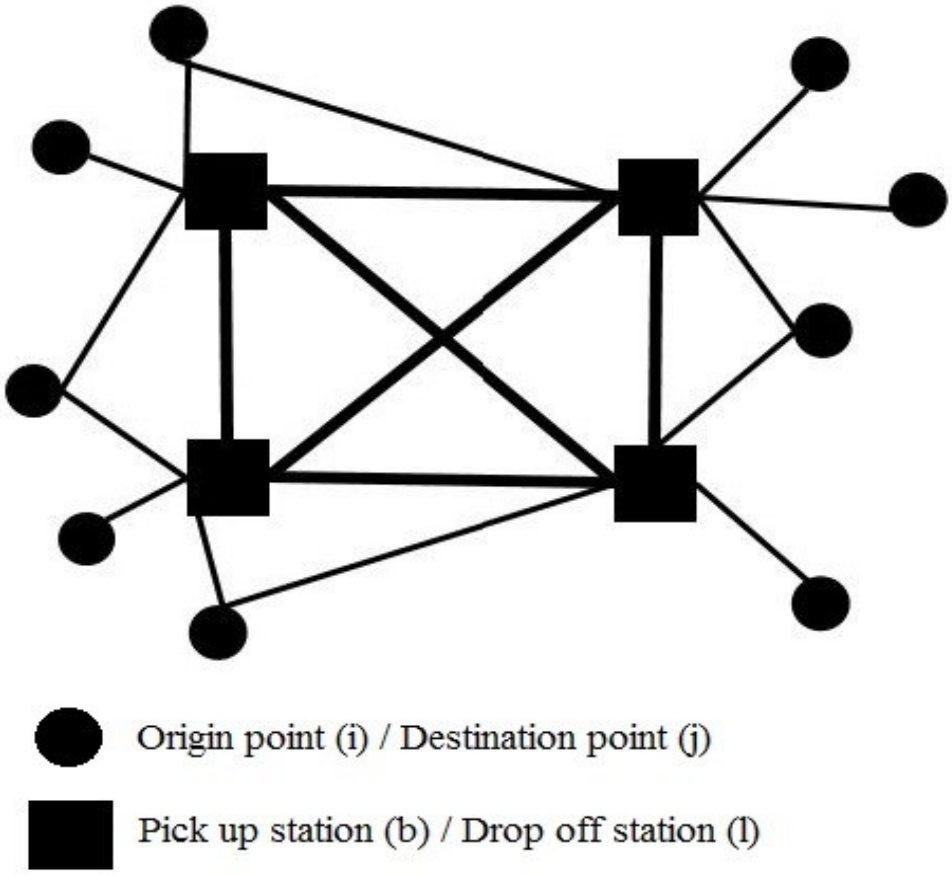


Figure 1. Bike Sharing Network

Building stations' cost is dependent on the number of stations to be opened and the level of capacity in each opened station, and the transportation cost consists of a walking trip from customer's origin to a station to pick up a bike and another walking trip from return station to the

final destination. In order to fulfill these walking trips the pedestrian needs to spend some time to travel the distances and this time is converted to cost using the average time value of each person.

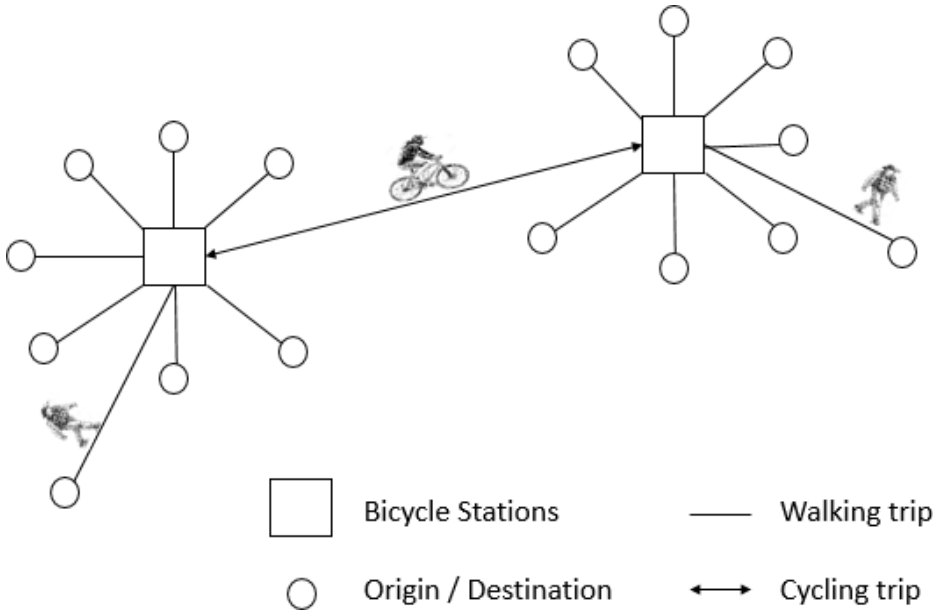


Figure 2. Bike Sharing Trips

There are two main decisions in this model. First the location decision which is affected by the cost of building station, demands and distances between demand zones and stations. The second one is the allocation decision which allocate the demand of each zone to those stations which are more cost efficient than others. Also, service level rates which are expressed by the availability of either bicycle or idle dock in each station, have important role in this model and affect the whole allocations in the network and the capacity and number of initial bikes in each station.

3.1. Assumptions

- The monthly demand follows exponential distribution and is allocated to stations without loss.
- The stations can be opened at different capacity levels.
- Each bicycle serves one person at a time, so during the travelling the bike is busy and would be available again, once it drops off in one station in the system.
- Balking is considered for users to pick up or drop off the bikes in all stations in the network.
- Fixed cost of maintaining a station depends on the location and the capacity level.

- The bike sharing network is closed, meaning that no bike leaves the system.
- Drop off service level is assumed to be greater than the service level of pick up all the times.
- The number of active days and hours are given as parameters to the model, so the network could be out of reach of users for some days or hours as the designer decides.

3.2. Input Parameters and Decision Variables

For the formulation of this problem, the following notation is used:

Sets:

$i.j \in I$	Index for origin and destination
$b.l \in B$	Index for potential bike stations
$k \in K$	Index for capacity level of a bike station

Parameters:

Λ_{ij}	Monthly customer demand from origin point i to destination point j
d_{ib}	Distance from origin point i to pick up station b (meters)
d_{bl}	Distance from pick up station b to drop off station l (meters)
d_{lj}	Distance from drop off station l to destination point j (meters)
c	Unit walking cost ($/meter)
f_{bk}	Monthly fixed cost of operating a station at b with capacity level k ($/station)
t	Number of active days per month
n	Number of active hours per day
p	Lower bound for μ_b/λ_b in each station to satisfy service level rates
r	Upper bound for μ_b/λ_b in each station to satisfy service level rates
e	Monthly fixed cost of providing each bike in the network ($/bike)
g	Riding speed of a passenger by bike (meter/hour)

Decision Variables:

X_{bk}	Binary variable that equals 1 if station b is opened with capacity level k and 0 otherwise
Y_{iblj}	Binary variable that equals 1 if customers travel from point i to j using stations b and l, and 0 otherwise
λ_b	Daily arrival rate of customers to pick up a bike from station b
μ_b	Daily arrival rate of customers to drop off a bike at station b
S_b	Number of initial bikes in station b

The unit walking cost is calculated based on the average time value of users ($/hour) and the average walking speed of users (meter/hour). The monthly fixed cost of operating a station depends on its location and capacity level. The monthly fixed cost of providing each bike in the network includes the capital cost and the maintenance cost per month and the riding speed is also the average speed of each passenger in the network (meter/hour). The upper bound and lower bound for μ_b/λ_b in each station are described in detail in the section of service level function.

3.3. Objective Function

A bicycle sharing network can be designed to optimize the objectives of users and system designers. The first one is viewpoint of customers which benefits the travelers most and tries to satisfy their needs of resources and minimize their cost simultaneously. Walking short distances before pick up and after drop off a bike would be users preference and they will be satisfied when they find resources available in the system. The second one is system designer's point of view, which imposes the highest revenue out of the system with lowest network building cost. This can be achieved by building the infrastructure with the lowest investment and the most possible successful bike renting. The following objective function combines both point of views by minimizing the building costs of investor and transportation costs of customers.

$$Min \sum_{i \in I} \sum_{b \in B} \sum_{l \in B \neq b} \sum_{j \in I \neq i} c\ d_{ib}\ Y_{iblj}\ \Lambda_{ij} + \sum_{i \in I} \sum_{b \in B} \sum_{l \in B \neq b} \sum_{j \in I \neq i} c\ d_{lj}\ Y_{iblj}\ \Lambda_{ij} + \sum_{b \in B} \sum_{k \in K} f_{bk}\ X_{bk} + \sum_{b \in B} e\ S_b \quad (1)$$

In the objective function (1), the first term represents walking cost from an origin point to a station to pick up a bike, the second term denotes walking cost from the return station to the destination, the third term is fixed cost of opening a station considering its location and with a specific level of docking capacity and the last term indicates cost of total bikes in the network.

3.4. Network Constraints

Based on the defined notation the network constraints are as follows:

$$\sum_{b \in B} \sum_{l \in B \neq b} \mathrm{Y}_{iblj} = 1 \,. \qquad \forall\, i.j \in I\,; i \neq j \tag{2}$$

$$\sum_{l \in B \neq b} \mathrm{Y}_{iblj} + \sum_{l \in B \neq b} \mathrm{Y}_{ilbj} \leq \sum_{k \in K} \mathrm{X}_{bk} \,. \qquad \forall\, i.j \in I\,; i \neq j \;.\forall\, b \in B \tag{3}$$

$$\sum_{k \in K} \mathrm{X}_{bk} \leq 1 \,. \qquad \forall\, b \in B \tag{4}$$

$$\lambda_b = (\frac{1}{t}) \sum_{i \in I} \sum_{l \in B \neq b} \sum_{j \in I \neq i} \mathrm{Y}_{iblj}\, \Lambda_{ij} \;. \qquad \forall\, b \in B \tag{5}$$

$$\mu_b = (\frac{1}{t}) \sum_{i \in I} \sum_{l \in B \neq b} \sum_{j \in I \neq i} \mathrm{Y}_{ilbj}\, \Lambda_{ij} \;. \qquad \forall\, b \in B \tag{6}$$

$$\sum_{b \in B} \mathrm{S}_b \geq \left(\frac{\sum_{i \in I} \sum_{b \in B} \sum_{l \in B \neq b} \sum_{j \in I \neq i} \mathrm{d}_{bl}\, \mathrm{Y}_{iblj}\, \Lambda_{ij}}{t\, n\, g} \right) \tag{7}$$

$$\mathrm{S}_b \geq \left(\frac{\sum_{k \in K} \mathrm{X}_{bk}\, k}{2} \right) + \sum_{k \in K} \mathrm{X}_{bk} - 0.5. \qquad \forall\, b \in B \tag{8}$$

$$\mathrm{S}_b \leq \left(\frac{\sum_{k \in K} \mathrm{X}_{bk}\, k}{2} \right) + \sum_{k \in K} \mathrm{X}_{bk}. \qquad \forall\, b \in B \tag{9}$$

$$\lambda_b \leq \mathrm{S}_b + \mu_b \;. \qquad \forall\, b \in B \tag{10}$$

$$\mu_b \leq \sum_{k \in K} \mathrm{X}_{bk}\, k - \mathrm{S}_b + \lambda_b \;. \qquad \forall\, b \in B \tag{11}$$

$$\lambda_b \geq \sum_{k \in K} X_{bk} \ . \qquad \forall\, b \in B \qquad (12)$$

$$\lambda_b . \mu_b \geq 0 \ . S_b \geq 0 \ \& \ integer \ . X_{bk} . Y_{iblj} \in \{0.1\} \qquad \forall\, i.j \in I.\ \forall\, b.l \in B.\ \forall\, k \in K \qquad (13)$$

Constraint (2) denotes that any demand between each pair of origin and destination points can travel through just one route (one pair of pick up and drop off stations), Constraint (3) ensures that there could be an allocation for pick up or drop off a bike at a station, if and only if that station is open. Constraint (4) denotes only one capacity level can be selected for an open station. Constraint (5) represents the total arrival rate of customers per day to pick up a bike from each station. Constraint (6) is the drop off rate of bikes to each returning station per day. Constraint (7) calculates the minimum number of bikes in the network (fleet size) needed to satisfy demand. The numerator is total distances should be traveled by bikes in the network to satisfy demand and the denominator computes the maximum distance each bike can cover in the network. Constraints (8) and (9) are calculating the initial number of bikes in each open station in the system considering the capacity of them, and in order to maximize the utilization of each station. The same sources were considered in C. Fricker and N. Gast (2014), where they developed a stochastic model to decrease the number of problematic stations (in a problematic station users face shortage in available bike for pick up or empty dock for drop off) and increase the performance of system to reach a given quality of service. In their study, they proved that for different cases considered, the number of bikes per stations that corresponds to the best performance of the network is half of the number of docks plus a very few more. Constraints (10) and (11) are bounding the rates of pick up and drop off in each open station logically, considering the capacity and initial number of bikes in that station. Constraint (12) denotes that there must be at least one passenger to pick up a bike for each open station. Constraint (13) demonstrates the type of decision variables.

3.5. Service Level Constraints

A bike sharing system usually provides two types of service to users. First is the availability level to pick up a bike from a specific station and second is the availability level of docks for users arriving to drop off a bike in the station. The service level of both distinct flows of users' arrival to the station is restricted by the capacity of the station (k) and number of bikes (h).

First type of users are the pickup users, who are pedestrians arrive to get a bicycle. Let's assume the arrival rate of this type is Poisson distributed with mean λ. Also, some users are willing to wait for pickup, when there is no bike available at the station. The probability of accepting to wait for pick up is defined with r which is called balking for pick up. Second type of users are return ones who have travelled toward the station and look for an idle dock to drop off their bikes. The arrival rate of this group of people is Poisson distributed with mean μ. Some travelers are also willing to wait for return when there is no idle dock available at the station. The probability of accepting to wait for drop off the bike is defined with s and is called balking for drop off (F. Laurent 2012).

Each station can be modeled with two random processes which are pedestrian arrivals and bicycle arrivals. All states of idle bike availability, empty dock availability, waiting of pedestrians or riders to pick up or drop off a bike are defined with state variable (h). For any given value of (h) the transition will occur to the neighboring values of (h + 1) or (h – 1) according to the following rules.

a) From h to h + 1, because of the arrival of return users and the transition rate would be μ if h < k (state of dock availability) or μs if h ≥ k (state of dock shortage).

b) From h to h -1, due to the arrival of users to pick up a bike and the transition rate is λ if h > 0 (state of bike availability) or λr if h ≤ 0 (state of bike shortage).

c) Any transition between h and h + m, for m ∉ {−1.1} has null rate.

Figure 3 shows the state transition diagram for a bi-sided waiting system;

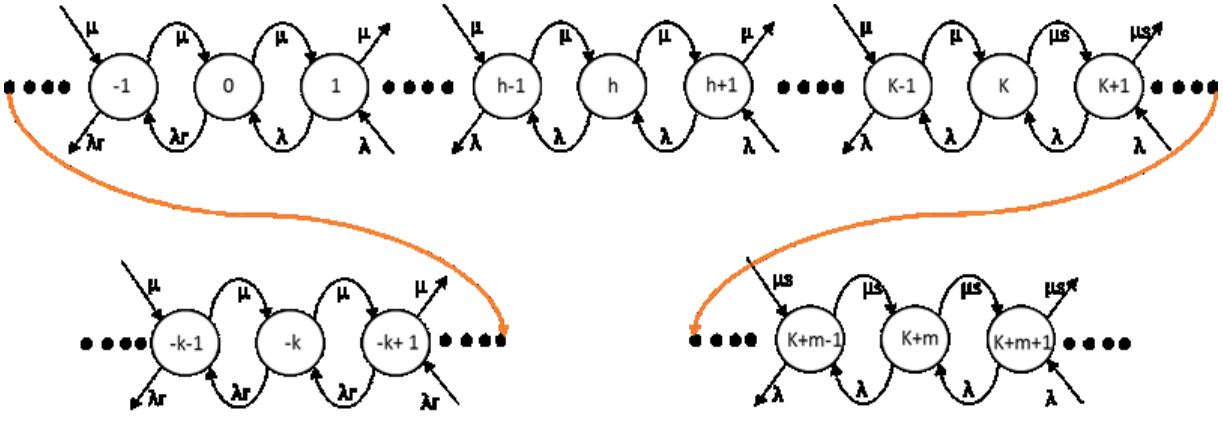


Figure 3. State Transition Diagram

Considering the arrival rates (λ, μ) and balking rates (r, s), the following parameters are represented to show the transition states and their probabilities;

$$\rho = \frac{\lambda r}{\mu} \qquad \sigma = \frac{\mu s}{\lambda} \qquad \phi = \frac{\mu}{\lambda}$$

Figure 4 depicts all possible states of a station and there would be three main subdomains;

A) When there is no bike available (state of bike shortage)

B) When both bike and dock are available (state of availability)

C) When there is no dock available (state of dock shortage)

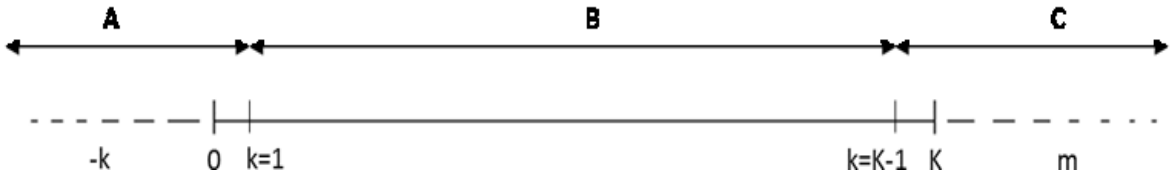


Figure 4. Subdomains according to the number of bikes at the station

Subdomain A; State of bike shortage

Subdomain A illustrates a situation when there is no bike available in the station to pick up and the user will decide to leave the station or wait to get service (F. Laurent 2012). The total probability of subdomain A is:

$$\sum_{k\geq 0} p_{-k} = \frac{p_0}{(1-\rho)} \text{ . where } \rho < 1$$

Subdomain B; State of availability

Subdomain B denotes a situation when there is at least one bicycle available for pickup and at least one empty dock to drop off a bike. The number of bicycles in the station may vary from 1 to k-1 and this condition ensures that any type of customer will be satisfied. The total probability of subdomain B is:

$$\sum_{k=1}^{k-1} p_k = \frac{p_0(\phi - \phi^k)}{(1-\phi)} \text{ . where } \phi \neq 1.$$

$$\sum_{k=1}^{k-1} p_k = (k-1)\, p_0 \,.\text{where } \phi = 1$$

Subdomain C; State of dock shortage

Subdomain C states a situation when there is no empty dock available in the station to drop off a bike and the rider will decide to leave the station or wait to get service. The total probability of subdomain C is:

$$\sum_{m\geq 0} p_{k+m} = \frac{p_0 \phi^k}{(1-\sigma)} \,.\text{where } \sigma < 1$$

Overall distributions:

The total probability function which contains all subdomains is:

$$\sum_{k\in z} p_k = \left(\sum_{k\geq 0} p_{-k}\right) + \left(\sum_{k=1}^{k-1} p_k\right) + \left(\sum_{m\geq 0} p_{k+m}\right) = \frac{p_0}{(1-\rho)} + \frac{p_0(\phi-\phi^k)}{(1-\phi)} + \frac{p_0\phi^k}{(1-\sigma)}$$

This function includes all possible scenarios so the total probability will be equal to 1, and the pivot probability p_0 (under the condition that $\frac{(\phi-\phi^k)}{(1-\phi)} = k-1$ where $\phi = 1$) would be:

$$p_0 = \frac{1}{\left[\frac{1}{(1-\rho)} + \frac{(\phi-\phi^k)}{(1-\phi)} + \frac{\phi^k}{(1-\sigma)}\right]}$$

3.5.1. Service level of Pick up and drop off

Evaluating the service level of each user is dependent on the type of resources that user is looking for in the bike station. A user coming to a station, will look for a bike to pick up and one's demand will be satisfied if there is at least one bicycle available. By contrary the demand of a rider would be an empty dock to drop off the bike. Therefore, the service levels of pick up and drop off users are different and will be extracted from the probability functions of aforementioned subdomains.

Service level of pick up (α) consists of subdomains B and C, where there would be at least one bike available for a user to pick up and it is the combination of these subdomains probabilities:

$$\frac{p_0(\phi - \phi^k)}{(1-\phi)} + \frac{p_0\phi^k}{(1-\sigma)} \geq \alpha$$

Or as the summation of all subdomains' probabilities is equal to 1, so the service level of pick up (α) can be calculated using the probability of subdomain A as follows:

$$\frac{p_0}{(1-\rho)} \leq 1 - \alpha$$

For a given set of arrival rates (λ, μ) and balking rates (r, s) for pick up and drop off, the graph of service level of pick up with respect to the capacity would be as shown in figure 5.

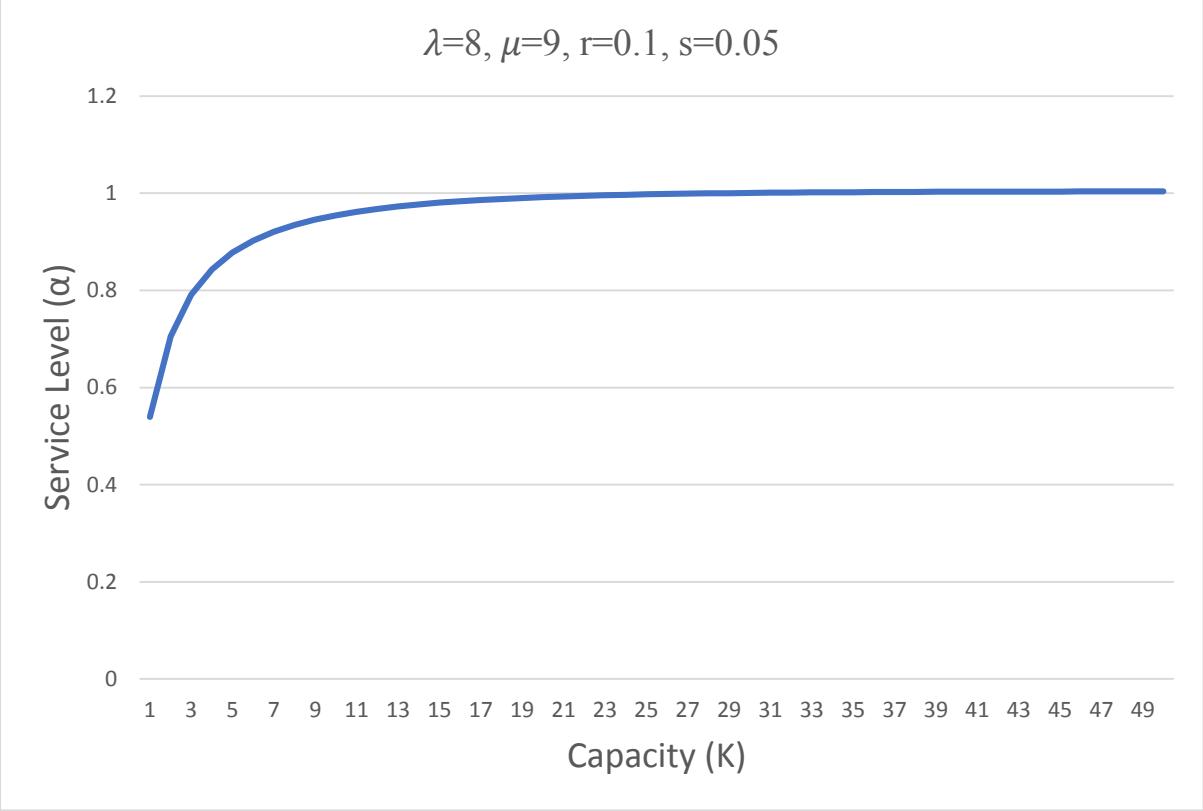


Figure 5. Graph of service level of pick up with respect to capacity

This function was proved to be concave as its second derivative was smaller than zero (for all possible given sets of parameters) and it is formulated as follows:

$$f(k) = \frac{p_0(\phi - \phi^k)}{(1-\phi)} + \frac{p_0\phi^k}{(1-\sigma)} \rightarrow$$

$$f(k) = \frac{(1-\rho)\,(\phi - \phi^k)\,(1-\sigma) + (1-\rho)\phi^k\,(1-\phi)}{(1-\phi)\,(1-\sigma) + (\phi - \phi^k)\,(1-\rho)\,(1-\sigma) + \phi^k\,(1-\rho)\,(1-\phi)}$$

$$f'(k) = \frac{(-1+\rho)\,(-1+\sigma)(\sigma-\phi)(-1+\phi)\,\phi^k\,Log[\phi]}{(-1+\sigma-\sigma\phi^k\ +\phi^{1+k} - \rho\,\phi\,(-1+\phi^k) + \rho\,\sigma\,(-\phi+\phi^k))^2}$$

$$f"(k) = \frac{(-1+\rho)\,(-1+\sigma)(\sigma-\phi)(-1+\phi)\,\phi^k\,(1+\phi^{1+k} - \rho\,\phi\,(1+\phi^k) + \sigma\,(-1-\phi^k+\rho\,(\phi+\phi^k)))Log[\phi]^2}{(-1+\sigma-\sigma\phi^k\ +\phi^{1+k} - \rho\,\phi\,(-1+\phi^k) + \rho\,\sigma\,(-\phi+\phi^k))^3}$$

On the other side, the service level of drop off (β) comprises subdomains A and B, where there would be at least one empty dock available for a user to drop a bike off and it is the combination of these subdomains' probabilities:

$$\frac{p_0(\phi - \phi^k)}{(1-\phi)} + \frac{p_0}{(1-\rho)} \geq \beta$$

Or as the summation of all subdomains' probabilities is equal to 1, so the service level of drop off (β) can be calculated using the probability of subdomain C as follows:

$$\frac{p_0\phi^k}{(1-\sigma)} \leq 1 - \beta$$

This function was also proved to be concave as its second derivative was always smaller than zero and for a given set of parameters, the graph of service level of drop off with respect to the capacity is as follows:

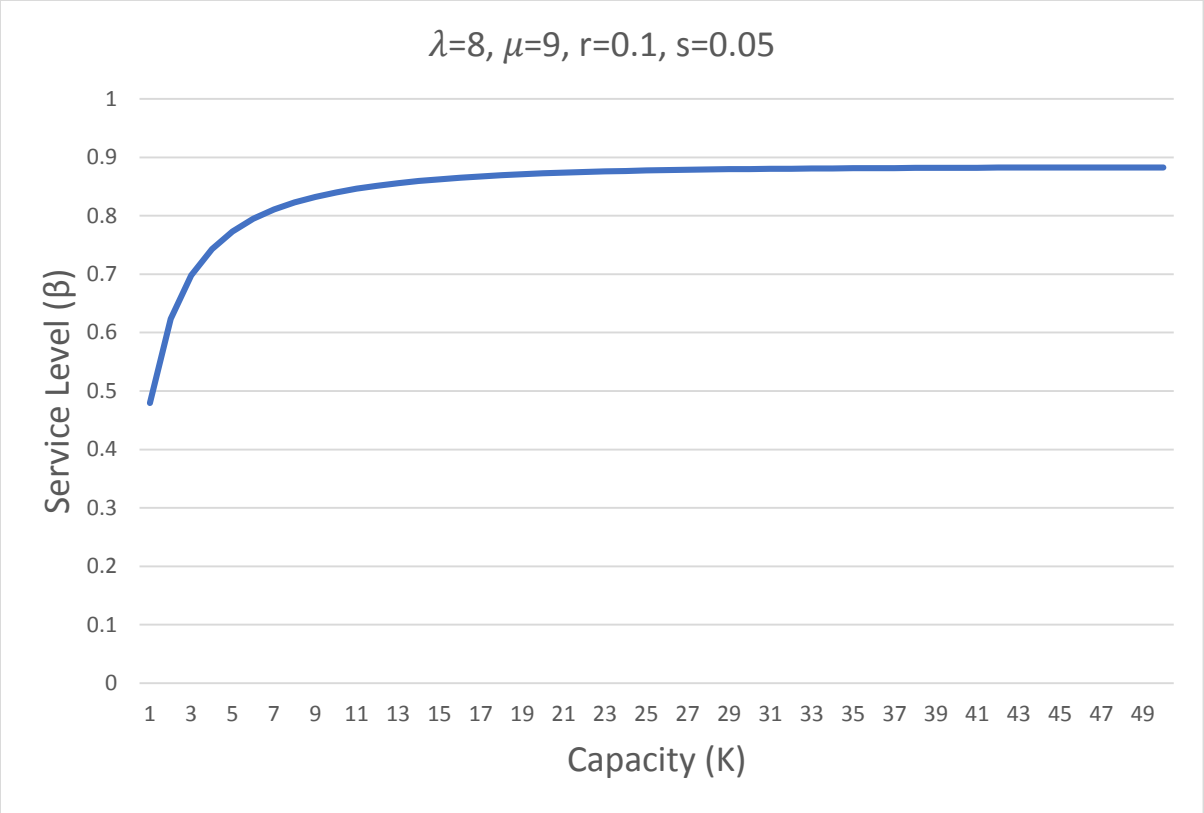


Figure 6. Graph of service level of drop off with respect to capacity

and it is formulated as follows:

$$f(k) = \frac{p_0(\phi - \phi^k)}{(1-\phi)} + \frac{p_0}{(1-\rho)} \rightarrow$$

$$f(k) = \frac{(1-\sigma)\,(\phi - \phi^k)\,(1-\rho) + (1-\sigma)\,(1-\phi)}{(1-\sigma)\,(1-\phi) + \,(1-\sigma)(\phi - \phi^k)\,(1-\rho) + \phi^k\,(1-\rho)\,(1-\phi)}$$

$$f'(k) = -\frac{(-1+\rho)\,(-1+\sigma)(-1+\phi)\,\phi^k(-1+\rho\phi)\,Log[\phi]}{(-1+\sigma-\sigma\phi^k\, + \phi^{1+k} - \rho\,\phi\,(-1+\phi^k) + \rho\,\sigma\,(-\phi+\phi^k))^2}$$

$$f"(k) = \frac{(-1+\rho)\,(-1+\sigma)(-1+\phi)\,\phi^k(-1+\rho\phi)\,(1+\phi^{1+k} - \rho\,\phi\,(1+\phi^k) + \sigma\,(-1-\phi^k+\rho\,(\phi+\phi^k)))Log[\phi]^2}{(-1+\sigma-\sigma\phi^k\, + \phi^{1+k} - \rho\,\phi\,(-1+\phi^k) + \rho\,\sigma\,(-\phi+\phi^k))^3}$$

3.5.2. Linearization of Service Level Expression

Since the nonlinear service levels formula can't be used in the optimization model, for any given set of parameters $(\alpha.\beta.r.s)$, we approximate them based on possible ranges of phi ($\phi = \frac{\mu}{\lambda}$) and capacity (k). To this end, we solve two separate optimization problems using the excel nonlinear programming solver to find the minimum and maximum acceptable values for phi. In addition, the exact value for the minimum possible capacity (k to satisfy a specified service level for pick up and drop off can be extracted approximately from the aforementioned graph of service levels as well, it should be noted that the graphs are just some examples).

For the formulation of these two problems, the following notation is used:

Parameters:

- α Service level of pick up (%)
- β Service level of drop off (%)
- r Probability of acceptance to wait for pick up (balking for pick up) (%)
- s Probability of acceptance to wait for drop off (balking for drop off) (%)
- k Minimum possible capacity

Decision Variables:

- λ Arrival rate of users to pick up a bike (daily)
- μ Arrival rate of users to drop off a bike (daily)

Based on the defined notation the optimization models can be formulated as follows: (as the constraints of both models are the same, we have merged them together)

$$Min\ \frac{\mu}{\lambda} \quad (1)$$

$$Max\ \frac{\mu}{\lambda} \quad (2)$$

Subject to:

$$\lambda \geq 0 \tag{3}$$

$$\mu \geq 0 \tag{4}$$

$$\left(\frac{\left(\frac{\mu}{\lambda}\right)-\left(\frac{\mu}{\lambda}\right)^{k}}{1-\left(\frac{\mu}{\lambda}\right)}+\frac{\left(\frac{\mu}{\lambda}\right)^{k}}{1-s\left(\frac{\mu}{\lambda}\right)}\right)/\left(\frac{1}{1-\frac{r}{\left(\frac{\mu}{\lambda}\right)}}+\frac{\left(\frac{\mu}{\lambda}\right)-\left(\frac{\mu}{\lambda}\right)^{k}}{1-\left(\frac{\mu}{\lambda}\right)}+\frac{\left(\frac{\mu}{\lambda}\right)^{k}}{1-s\left(\frac{\mu}{\lambda}\right)}\right) \geq \alpha \tag{5}$$

$$\left(\frac{\left(\frac{\mu}{\lambda}\right)-\left(\frac{\mu}{\lambda}\right)^{k}}{1-\left(\frac{\mu}{\lambda}\right)}+\frac{1}{1-\frac{r}{\left(\frac{\mu}{\lambda}\right)}}\right)/\left(\frac{1}{1-\frac{r}{\left(\frac{\mu}{\lambda}\right)}}+\frac{\left(\frac{\mu}{\lambda}\right)-\left(\frac{\mu}{\lambda}\right)^{k}}{1-\left(\frac{\mu}{\lambda}\right)}+\frac{\left(\frac{\mu}{\lambda}\right)^{k}}{1-s\left(\frac{\mu}{\lambda}\right)}\right) \geq \beta \tag{6}$$

In the objective function, in the first model (1) it represents the minimization of phi and in the second model (2) it represents the maximization of phi. Constraints (3) and (4) denote that the rate of pick up and drop off in each station cannot take a negative value. Constraint (5) denotes the service level of pick up must be satisfied in an open station. The left side of the formula shows the probability of having bike available in an open station and the right side is the given service level of pick up to the model that must be satisfied. Constraint (6) indicates the drop off service level must be satisfied in an open station. The left side of the formula illustrates the probability of having empty dock available in an open station and the right side is the given service level of drop off to the model that must be satisfied.

So for any given set of parameters $(\alpha.\beta.r.s)$, the range of phi and the minimum capacity per each open station (minimum k) are calculated and then are used in the main linear programming model. The minimum and maximum possible values for phi are named r and p respectively, and are given as parameters to the main model. Thus the minimum number of docks per each open station along with the following constraints are added to the main bike sharing model to satisfy the service level of pick up and drop off per each open station in the network.

$$\mu_b \geq p\,\lambda_b \ . \qquad \forall\, b \in B \tag{14}$$

$$\mu_b \leq r\,\lambda_b \ . \qquad \forall\, b \in B \qquad (15)$$

Constraints (14) and (15) together are bounding the range of μ/λ for each open station in the bike sharing network.

3.6. The MILP Model

The proposed MILP model is represented in its finalized form as follows

Objective Function:

$$Min \sum_{i\,\epsilon\,I}\sum_{b\,\epsilon\,B}\sum_{l\,\epsilon\,B\neq b}\sum_{j\,\epsilon\,I\neq i} c\ \mathrm{d}_{ib}\ \mathrm{Y}_{iblj}\ \Lambda_{ij} + \sum_{i\,\epsilon\,I}\sum_{b\,\epsilon\,B}\sum_{l\,\epsilon\,B\neq b}\sum_{j\,\epsilon\,I\neq i} c\ \mathrm{d}_{lj}\ \mathrm{Y}_{iblj}\ \Lambda_{ij} + \sum_{b\,\epsilon\,B}\sum_{k\,\epsilon\,K} \mathrm{f}_{bk}\ \mathrm{X}_{bk} + \sum_{b\,\epsilon\,B} e\ \mathrm{S}_b \qquad (1)$$

Subject to:

$$\sum_{b\,\epsilon\,B}\sum_{l\,\epsilon\,B\neq b} \mathrm{Y}_{iblj} = 1\ . \qquad \forall\, i.j \in I\,;\, i \neq j \qquad (2)$$

$$\sum_{l\,\epsilon\,B\neq b} \mathrm{Y}_{iblj} + \sum_{l\,\epsilon\,B\neq b} \mathrm{Y}_{ilbj} \leq \sum_{k\,\epsilon\,K} \mathrm{X}_{bk}\ . \qquad \forall\, i.j \in I\,;\, i \neq j\ .\forall\, b \in B \qquad (3)$$

$$\sum_{k\,\epsilon\,K} \mathrm{X}_{bk}\ \leq 1\ . \qquad \forall\, b \in B \qquad (4)$$

$$\lambda_b = \left(\frac{1}{t}\right)\sum_{i\,\epsilon\,I}\sum_{l\,\epsilon\,B\neq b}\sum_{j\,\epsilon\,I\neq i} \mathrm{Y}_{iblj}\ \Lambda_{ij}\ . \qquad \forall\, b \in B \qquad (5)$$

$$\mu_b = \left(\frac{1}{t}\right)\sum_{i\,\epsilon\,I}\sum_{l\,\epsilon\,B\neq b}\sum_{j\,\epsilon\,I\neq i} \mathrm{Y}_{ilbj}\ \Lambda_{ij}\ . \qquad \forall\, b \in B \qquad (6)$$

$$\sum_{b\,\epsilon\,B} \mathrm{S}_b \geq \left(\frac{\sum_{i\,\epsilon\,I}\sum_{b\,\epsilon\,B}\sum_{l\,\epsilon\,B\neq b}\sum_{j\,\epsilon\,I\neq i} \mathrm{d}_{bl}\ \mathrm{Y}_{iblj}\ \Lambda_{ij}}{t\ n\ g}\right) \qquad (7)$$

$$S_b \geq \left(\frac{\sum_{k \in K} X_{bk}\, k}{2}\right) + \sum_{k \in K} X_{bk} - 0.5. \qquad \forall\, b \in B \qquad (8)$$

$$S_b \leq \left(\frac{\sum_{k \in K} X_{bk}\, k}{2}\right) + \sum_{k \in K} X_{bk}. \qquad \forall\, b \in B \qquad (9)$$

$$\lambda_b \leq S_b + \mu_b \ . \qquad \forall\, b \in B \qquad (10)$$

$$\mu_b \leq \sum_{k \in K} X_{bk}\, k - S_b + \lambda_b \ . \qquad \forall\, b \in B \qquad (11)$$

$$\lambda_b \geq \sum_{k \in K} X_{bk} \ . \qquad \forall\, b \in B \qquad (12)$$

$$\mu_b \geq p\, \lambda_b \ . \qquad \forall\, b \in B \qquad (13)$$

$$\mu_b \leq r\, \lambda_b \ . \qquad \forall\, b \in B \qquad (14)$$

$$\lambda_b . \mu_b \geq 0 \ . S_b \geq 0 \ \&\ integer \ . X_{bk} . Y_{iblj} \in \{0.1\} \qquad \forall\, i.j \in I.\ \forall\, b.l \in B.\ \forall\, k \in K \qquad (15)$$

3.7. Proposed Genetic Algorithm (Solution Method)

According to the mentioned contents in the second chapter and the researches done by the researchers, the degree of difficulty of such facility location problems, especially when the number of demand zones and potential stations increases, is NP hard and the accuracy of this theory is verified by (R. J. Fowler *et al*. 1981, N. Megiddo *et al.* 1982 and T. Gonzalez *et al.* 1985) who have proved that the different scenarios of facility location problem are NP hard. Thus, we resort to meta-heuristic methods in order to achieve an acceptable solution in a reasonable time for larger size instances of the problem.

According to our studies on meta-heuristic algorithms and among several methods that were tried to be applied on the model under this study, the genetic algorithm was selected and used to solve our model in this research. Its acceptable results and compliance with our model ensured us that this method is appropriate to be applied for large scale instances as well. Defining the chromosome

in a way to apply the steps of genetic algorithm practically and search the solution area properly, was the most difficult part of our job. So in order to encode the problem, matrix string was selected and designed in such a way that using the crossover and mutation operators, it be possible to search the solution area appropriately (M. Zandieh and N. Karimi 2010).

The steps of this algorithm are as follows:

3.7.1. Step 1: Coding and defining the chromosome

Firstly, in order to make a chromosome, two matrices presenting the pick-up and drop off stations are defined. In both matrices the rows are representing the origin zones and columns the destination ones. Then in the pick-up matrix each cell shows the selected station to pick up the bike and ride among each pair of origin and destination zones. Similarly, in the drop off matrix each cell depicts the opted station to drop off the bike after riding among each pair of origin and destination zones.

An example of coding and defining a chromosome for a problem with 4 demand zones and 3 possible stations are as follows:

Pick up	1	2	3	4
1	0	3	3	2
2	1	0	2	1
3	2	2	0	2
4	1	1	3	0

Drop off	1	2	3	4
1	0	1	2	1
2	3	0	1	3
3	3	1	0	1
4	3	2	2	0

Figure 7. Coding a problem with 4 demand zones and 3 possible stations

In this example in order to satisfy the demand among 4 zones and using 3 possible stations there are 12 routes as follows:

Origin zone	Pick up station	Drop off station	Destination zone
1	3	1	2
1	3	2	3
1	2	1	4
2	1	3	1
2	2	1	3
2	1	3	4
3	2	3	1
3	2	1	2
3	2	1	4
4	1	3	1
4	1	2	2
4	3	2	3

3.7.2. Step 2: Creating the initial population

In order to create the initial population, different scenarios are used in this algorithm. A part of initial solutions is made randomly using all possible stations to be allocated to demand zones. Another part of initial solutions is made using the Roulette Wheel strategy which gives a direction to make the initial population. To this end, by knowing the distances of all possible stations to each demand zone in the network, we give more chance to some closest stations to be selected to satisfy the demand. In order to calculate the probability of stations' selection for each demand zone, we define a table where each cell shows the distance of a demand zone to a station. We first take the inverse of the value of cells (distances) followed by computing the probability of selecting each station which is equal to the ratio of its inversed value over the summation of inversed values of all stations to a specific demand zone. To clarify the mentioned calculation, an example with a table of distances, inversed values and probabilities are presented in the following.

Distances (D) (meters)	Stations (B,L)		
Zones (I,J)	1	2	3
1	905.873	2609.643	2787.243

2	57.692	1769.180	1960.178
3	797.601	917.009	1124.910
4	1656.541	190.604	453.186
5	2535.227	886.679	856.133

Inversed values (1/D)	Stations (B,L)			Sum
Zones (I,J)	1	2	3	
1	0.00110	0.00038	0.00036	0.00185
2	0.01733	0.00057	0.00051	0.01841
3	0.00125	0.00109	0.00089	0.00323
4	0.00060	0.00525	0.00221	0.00806
5	0.00039	0.00113	0.00117	0.00269

Probabilities (1/D/Sum)	Stations (B,L)			Sum
Zones (I,J)	1	2	3	
1	0.598039	0.207594	0.194367	1
2	0.941583	0.030704	0.027713	1
3	0.387774	0.33728	0.274946	1
4	0.074927	0.651191	0.273882	1
5	0.146617	0.419213	0.43417	1

Figure 8. An example of roulette wheel selection method

The selection probability distribution can be generated using the following equation:

$$P_{(b)} = \frac{R_{(b)}}{\sum_{i=1}^{n} R_i}$$

In this equation the $P_{(b)}$ is the probability of selecting station b for a specific zone to satisfy its demand, $R_{(b)}$ is the inversed distance of station b to that demand zone and $\sum_{i=1}^{n} R_i$ is the summation of inversed distances of all possible stations to the demand zone.

In fact, in this method, the stations are placed on the roulette wheel and each station occupies a part of the wheel according to its probability. Then the wheel is rotated and after stopping, the station indicated by the indicator is selected. The selection process turns the roulette wheel enough times, and it is logical that each time a station with more space on the wheel has more chance of being selected (S. N. Sivanandam and S. N. Deepa 2008).

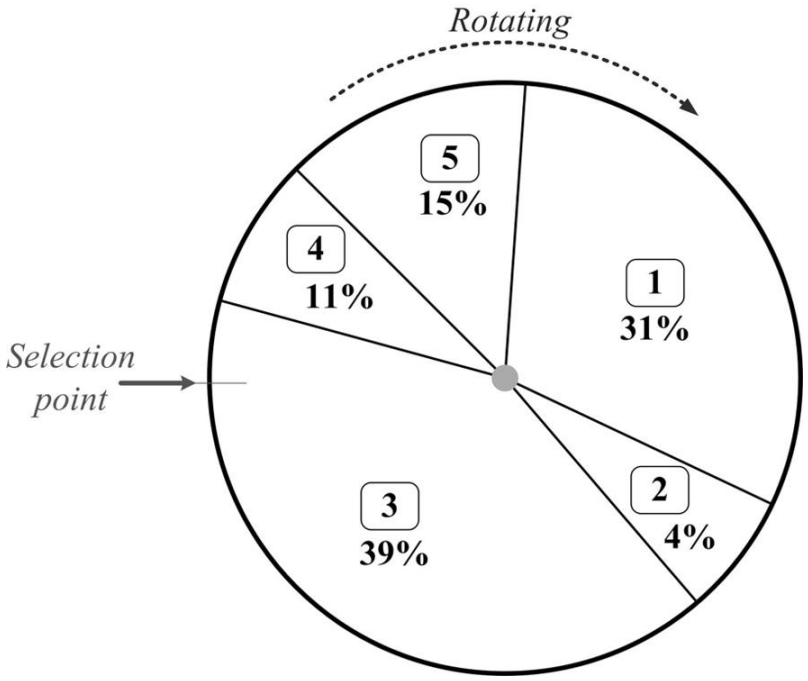


Figure 9. Roulette Wheel Selection Method

In order to help the algorithm to find the best solutions as fast as possible, an engineered solution can be added to other solutions initially, which specify the closest station to each demand zone at first and then provide us with a solution using the closest station to each zone for pick up and drop off. This solution can be infeasible but going through the crossover and mutation operators in the next steps and by the help of other solutions produces better offspring in next generations. It can be noted that, the number of initial solutions is given to the algorithm as a parameter, so it is obvious that whatever this value (the number of initial solutions) decreases, the effect of engineered solution increases.

3.7.3. Step 3: Crossover

The crossover operator combines the characteristics of parents to make offspring and to create better chromosomes. There are a variety of ways to do this step and in this study, according to the specific form of coding and the formation of the chromosomes, three methods are used. Which among all parents, one third of them are combined using the first method, one third using the second one and the rest of parents follow the third way to be mixed.

The first method of crossover is done in such a way that after selecting one third of chromosomes as parents, one demand zone of each chromosome is randomly selected, and a single point

crossover operation is performed on this zone. This is done by selecting a point on the bits of this zone and then the genes before this point on the first chromosome and the genes after this point on the second chromosome are used to form the first child. To form the second child, this operation is done inversely (B. Zarei, M. R. Meybodi and M. Abbaszadeh 2007).

An example of a single-point crossover and the formation of offspring chromosomes for a problem with 4 demand zones and 3 possible stations is as follows.

Parent 1

Pick up	1	2	3	4
1	**0**	**3**	**3**	**2**
2	1	0	2	1
3	2	2	0	2
4	1	1	3	0

Drop off	1	2	3	4
1	**0**	**1**	**2**	**1**
2	3	0	1	3
3	3	1	0	1
4	3	2	2	0

offspring 1

Pick up	1	2	3	4
1	**0**	**3**	**2**	**1**
2	1	0	2	1
3	2	2	0	2
4	1	1	3	0

Drop off	1	2	3	4
1	**0**	**1**	**1**	**3**
2	3	0	1	3
3	3	1	0	1
4	3	2	2	0

Parent 2

Pick up	1	2	3	4
1	**0**	**1**	**2**	**1**
2	2	0	2	1
3	1	1	0	2
4	3	3	3	0

Drop off	1	2	3	4
1	**0**	**2**	**1**	**3**
2	1	0	1	3
3	2	2	0	3
4	1	2	1	0

offspring 2

Pick up	1	2	3	4
1	**0**	**1**	**3**	**2**
2	2	0	2	1
3	1	1	0	2
4	3	3	3	0

Drop off	1	2	3	4
1	**0**	**2**	**2**	**1**
2	1	0	1	3
3	2	2	0	3
4	1	2	1	0

Figure 10. Sample of a single-point crossover

In figure 10, the first demand zone was randomly selected and according to the identified breakpoint, the combination of these two chromosomes was performed.

The second method of crossover is done in such a way that after selecting the second part of chromosomes as parents (one third of total), one demand zone of each chromosome is randomly selected, and a two-point crossover operation is implemented on this zone. This is done by

selecting two points on the bits of this demand zone and then the genes between these points are exchanged with each other. (B. Zarei, M. R. Meybodi and M. Abbaszadeh 2007).

An example of a two-point crossover and the formation of offspring chromosomes for a problem with 4 demand zones and 3 possible stations is as follows.

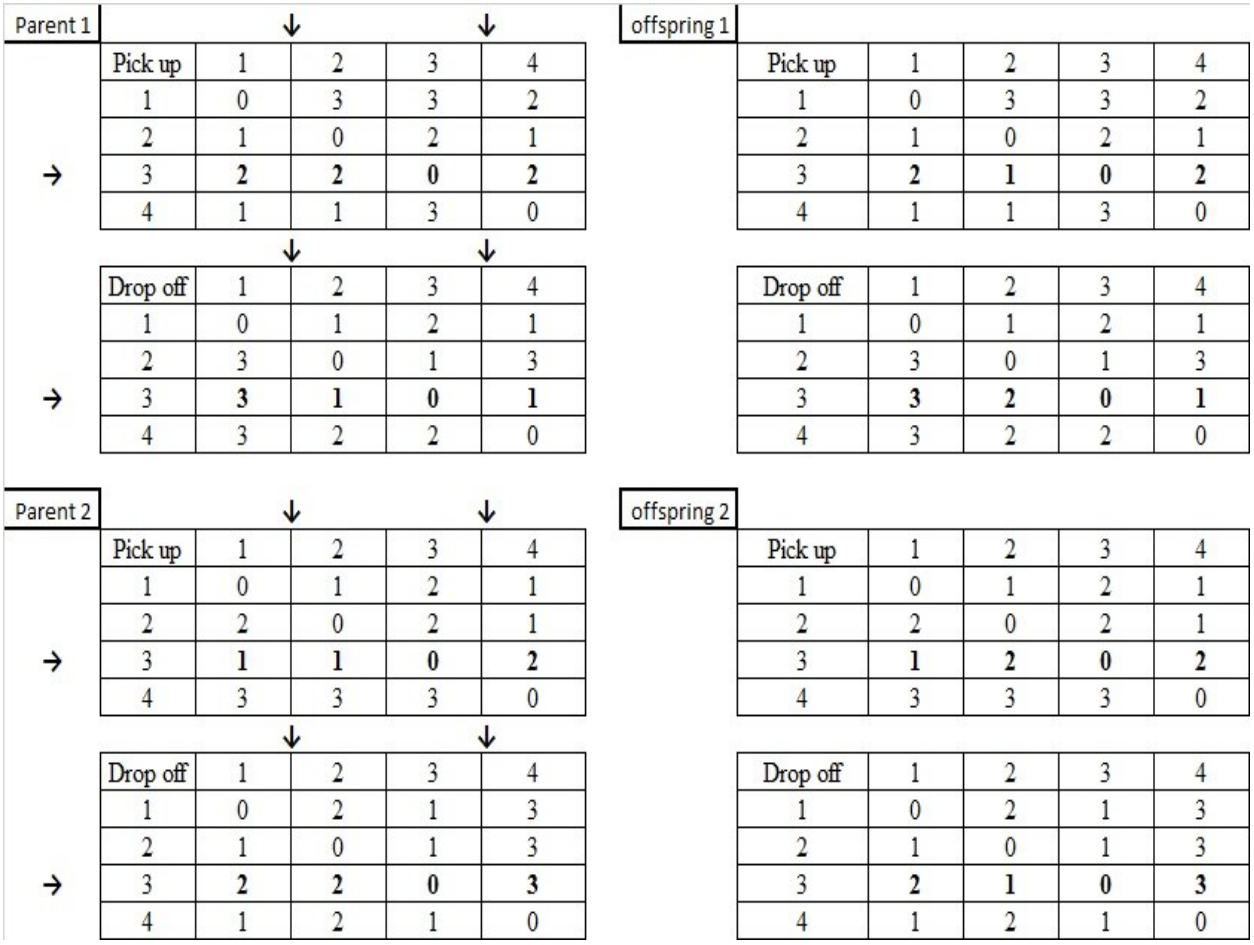


Figure 11. Sample of a two-point crossover

In figure 11, the third demand zone was randomly selected and according to the identified breakpoints, the combination of these two chromosomes was performed.

The third method of crossover is done in such a way that after selecting the last part of the chromosomes as parents, a demand zone is randomly selected. Then some random numbers are generated between 0 and 1 (corresponding to the bits of this demand zone), and the crossover operation is performed on the selected zone using the random values. In this way that, in order to form the first child, if the random value is less than a specific amount like 0.6, the corresponding

gene from the first chromosome will be transferred to the child chromosome, otherwise the gene from the second chromosome is selected to form the child chromosome. In order to make the second child, an inverse operation will be done (number 0.6 has been selected experimentally) (J. C. Bean 1994).

An example of a random number crossover and the formation of offspring chromosomes for a problem with 4 demand zones and 3 possible stations is as follows:

Parent 1

	Pick up	1	2	3	4
	1	0	3	3	2
	2	1	0	2	1
	3	2	2	0	2
→	4	**1**	**1**	**3**	**0**

	Drop off	1	2	3	4
	1	0	1	2	1
	2	3	0	1	3
	3	3	1	0	1
→	4	**3**	**2**	**2**	**0**

offspring 1

Pick up	1	2	3	4
1	0	3	3	2
2	1	0	2	1
3	2	2	0	2
4	**3**	**1**	**3**	**0**

Drop off	1	2	3	4
1	0	1	2	1
2	3	0	1	3
3	3	1	0	1
4	**1**	**2**	**1**	**0**

Parent 2

	Pick up	1	2	3	4
	1	0	1	2	1
	2	2	0	2	1
	3	1	1	0	2
→	4	**3**	**3**	**3**	**0**

	Drop off	1	2	3	4
	1	0	2	1	3
	2	1	0	1	3
	3	2	2	0	3
→	4	**1**	**2**	**1**	**0**

Random numbers	0.62	0.21	0.7	0.13

offspring 2

Pick up	1	2	3	4
1	0	1	2	1
2	2	0	2	1
3	1	1	0	2
4	**1**	**3**	**3**	**0**

Drop off	1	2	3	4
1	0	2	1	3
2	1	0	1	3
3	2	2	0	3
4	**3**	**2**	**2**	**0**

Figure 12. Sample of a random number crossover

In figure 12, the fourth demand zone was randomly selected and according to the generated random numbers, the combination of these two chromosomes was performed.

3.7.4. Step 4: Mutation

3.7.4.1. Light Mutation

At this stage, a certain number of new generation chromosomes are selected not on the basis of their fitness value, but randomly and with the same chance of mutation which is given as a parameter. Then, on each selected chromosome, one bit (gene) of a demand zone is randomly selected and its value is replaced by another possible value (among all possible stations). The light mutation operation is more random than the crossover operation and causes a change in the new generation at a lower rate. However, it can improve the optimization process by providing essential features that are not available in the current generation (S. Olariu and A. Y. Zomaya 2005).

An example of a light mutation and the formation of offspring chromosomes for a problem with 4 demand zones and 3 possible stations is as follows.

before

Pick up	1	2	3	4
1	0	3	3	2
2	1	0	**2**	1
3	2	2	0	2
4	1	1	3	0

Drop off	1	2	3	4
1	0	1	2	1
2	3	0	**1**	3
3	3	1	0	1
4	3	2	2	0

after

Pick up	1	2	3	4
1	0	3	3	2
2	1	0	**1**	1
3	2	2	0	2
4	1	1	3	0

Drop off	1	2	3	4
1	0	1	2	1
2	3	0	**3**	3
3	3	1	0	1
4	3	2	2	0

Figure 13. Sample of a light mutation

In figure 13, the third genes of pick up and drop off from the second demand zone were randomly selected and replaced by another possible values (among all possible stations).

3.7.4.2. Heavy Mutation

In this algorithm an offspring can only be selected to transfer to the next generation if it is at least better than one of its parents. So after light mutation, the fitness value of the solutions is calculated according to the objective function of the mathematical model, and a process called heavy mutation takes place in the case that during one reproduction no child is nominated to go to the next generation (in other words, no child is found to be better than at least one of its parents and the population remains unchanged). Under such condition, which happens rarely and indicate that the chromosomes converge to a specific part of the solution area, all chromosomes except the best chromosome are replaced with completely random solutions (among possible solutions) by performing a heavy mutation, so that the algorithm retaining the best chromosome found so far, move randomly to another part of the solution area and continue searching to find better solutions.

3.7.5. Step 5: Evaluation and Selection

At this stage, after the completion of a new generation, the fitness value of new solutions is calculated, and the selection process takes place among the chromosomes of the previous generation and the new generation. In this method, any solution with higher fitness value has more chance of being selected and the best solutions are selected to be transferred to the next generation proportional to the population size. For example, if we start with 100 initial solutions and set the rate of crossover 1.5, for the next generation the algorithm reproduces 150 (100*1.5) offspring and calculates the fitness value of whole solutions after mutation operation. Then during the selection process the algorithm chooses the best 100 solutions (population size) out of the 250 solutions (100 parents + 150 offspring) to make the next generation.

It should be noted that, during the evaluation process and in order to avoid using a same station for both pick up and drop off bike among two specific origin and destination zones, and totally in order to find good feasible solutions, a penalty is added to the objective function of infeasible solutions. In this way, as time goes on, using the crossover and mutation operations and during the evolution process to make new generations, infeasible solutions get replaced by feasible ones with appropriate fitness values.

3.7.6. Step 6: Termination Condition

In order to stop the algorithm, a criterion is considered as the termination condition. This criterion can be a certain number of iterations (generations) in general or a certain number of constant

consecutive iterations (consecutive iterations without improvement). In the proposed algorithm in this research, the criterion for termination is determined by a certain number of consecutive iterations without improving the objective function.

An overview of the solution methodology is as follows.

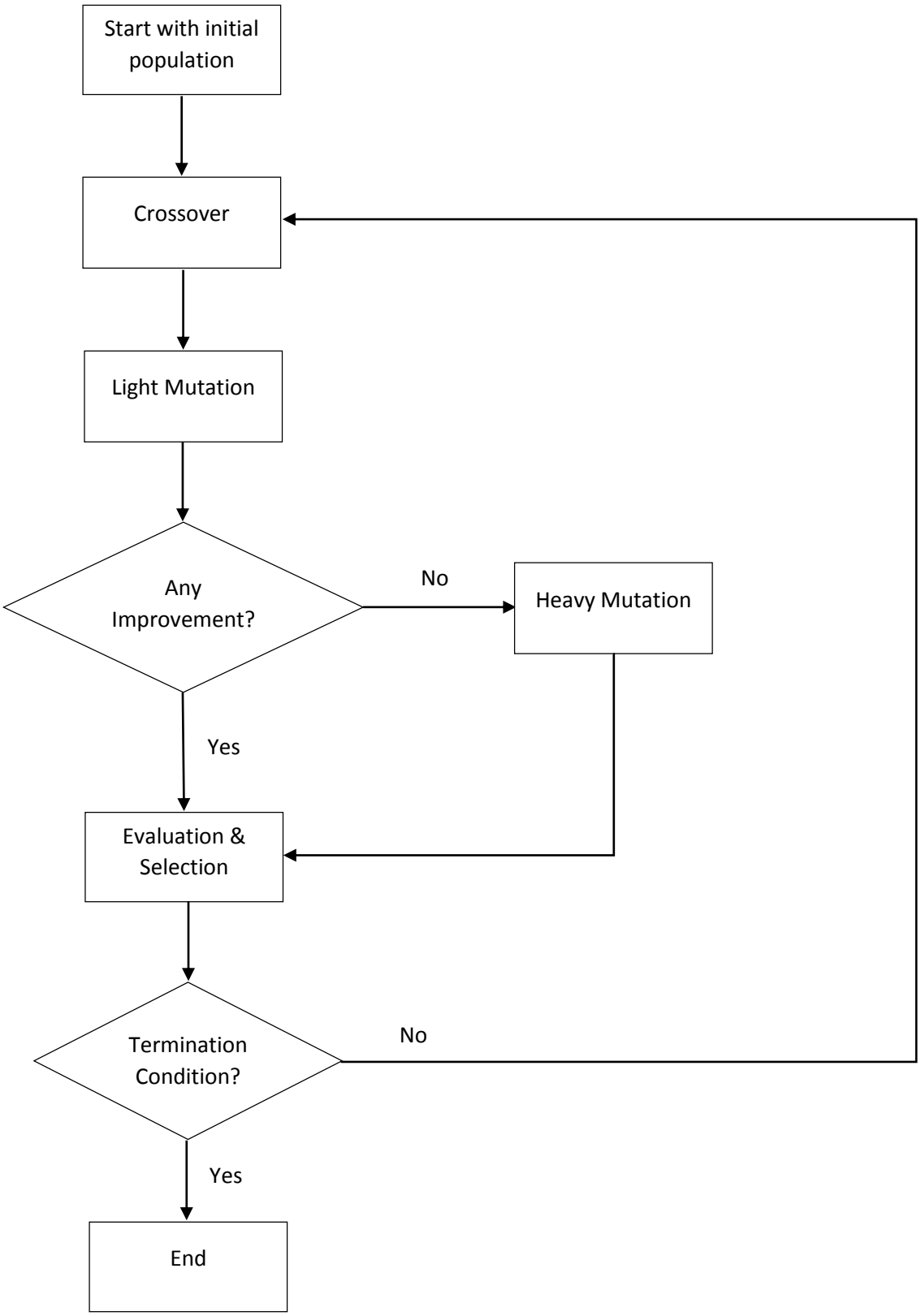


Figure 14. Genetic Algorithm Flow Diagram

In this chapter, at first the problem and its features were defined. Then by describing the assumptions, parameters and decision variables of the problem, a mathematical model was presented, and the objective function and constraints of the model were explained. Then, the service level function was described in detail. Finally, after implying the difficulty degree of the problem, the genetic algorithm was examined and all its steps were explained.

Chapter 4 - Numerical Experiments

In this chapter, we first describe the way our data have been gathered and analyze the sensitivity of the main parameters in detail. We then validate the design solutions with simulation and study the performance of the proposed genetic algorithm by presenting the computational results in small to large scale instances. The mathematical model has been coded using OPL and solved by IBM ILOG CPLEX Optimization Studio Version 12.8.0.0. Also, the genetic algorithm has been encoded using C++ in Microsoft Visual Studio Professional 2013 Version 12.0.21005.1 REL and ran on a computer with a Core i5 – 6500 CPU 3.2 GHz Processor and 16 GB of RAM.

4.1. Input Data

Since the city of Montreal was considered as the case study, the input data had to be consistent with the population, area and geography of the city. The raw information of Bixi company was accessed through their open-source data in their website (https://www.bixi.com/en/open-data). Using the statistics from the previous years, the following information was extracted: The number of daily and monthly trips between all stations, Stations of origin and destination of each trip, Time spent for each trip to be completed and also the number, address and location of stations (latitude and longitude of each station).

To structure the data, the city of Montreal was divided into different zones and the amount of monthly travel demands between the identified zones were extracted from the bixi open data (according to the number and location of stations in different zones). All required distances from origin zone (i) to pick up station (b), from pick up station (b) to drop off station (l), and from drop off station (l) to destination zone (j) were extracted and calculated in meters. The average time value of users was assumed to be $25 per hour and the average walking speed of users 4700 meters per hour, so the unit walking cost was calculated as 25/4700=$0.00532 per meter. The monthly fixed cost of opening a station was considered based on its capacity level starting $125 per month for each dock, where the maximum capacity level was set as 30 docks. The monthly fixed cost of providing bikes in the network, including the capital cost and maintenance cost was measured $128 per month for each bike. The average riding speed of each passenger by bike in the network was considered to be 16000 meters per hour. The minimum and maximum possible values for μ_b/λ_b in each station were calculated based on the different set of given parameters (α, β, r, s) for service levels (as described in the service level section in chapter 3). For example for a set of

α=0.7, β=0.8, r=0.1, s=0.2, the minimum possible value for μ_b/λ_b in each station were calculated as 0.76938 and the maximum as 1.0551. Also the number of operational days per month was set as 30 days and the number of active hours per day as 12 hours.

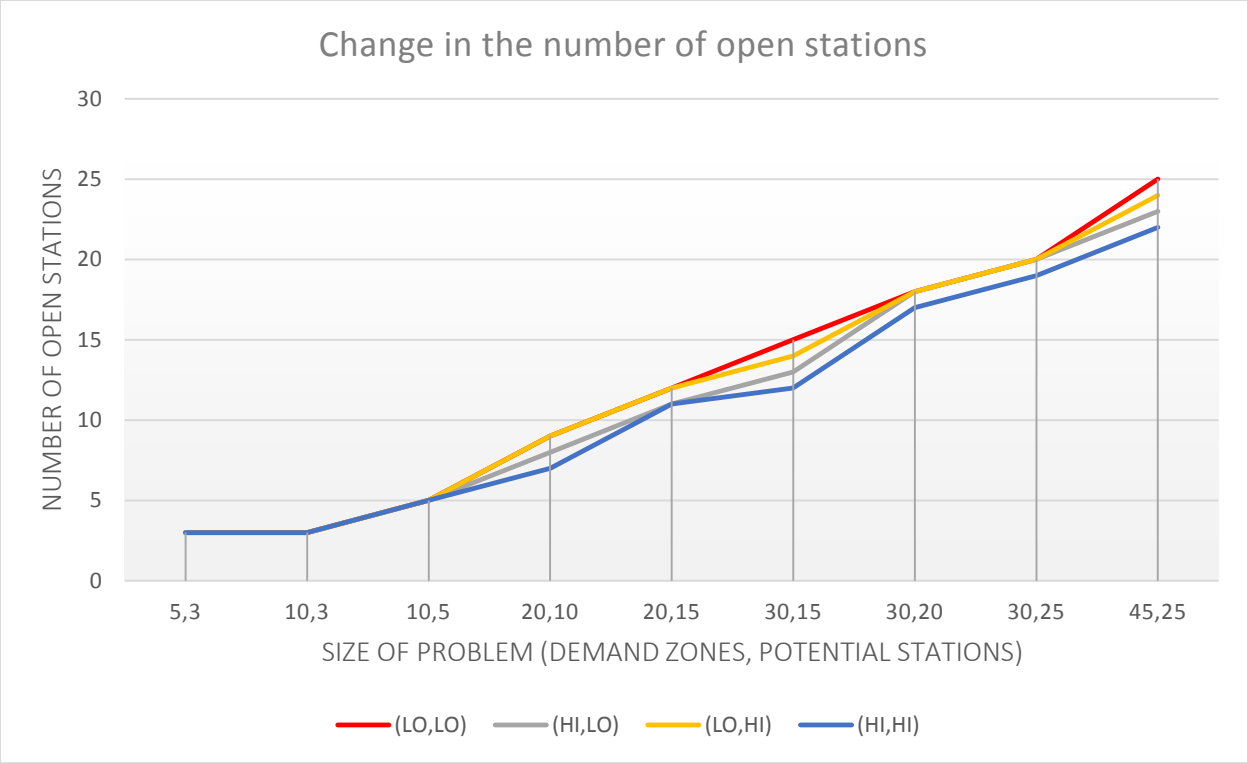
4.2. Sensitivity analysis of the main parameters

In this section, we study the effects of the main parameters (α, β, r, s) on the different components of the proposed bike sharing network. We observe the changes in the value of objective function and decision variables of the model, by changing the value of main parameters. To this end, four different sets of α, β, r, s are considered and in each instance the couple (α, β) (or (r, s)) has been kept constant while changing the values of (r, s) (or (α, β)). The different sets of parameters are as follows:

Table 3. Different sets of parameters

Four different sets of parameters	First term: Couple (α, β)	Second term: Couple (r, s)
(LO,LO)	α=0.7, β=0.8	r=0.1, s=0.2
(LO,HI)	α=0.7, β=0.8	r=0.2, s=0.35
(HI,LO)	α=0.8, β=0.9	r=0.1, s=0.2
(HI,HI)	α=0.8, β=0.9	r=0.2, s=0.35

Then the model is solved in different sizes from small to large scales and the results are analyzed in detail. In what follows, we examine the effects of these parameters on each component of the designed network.



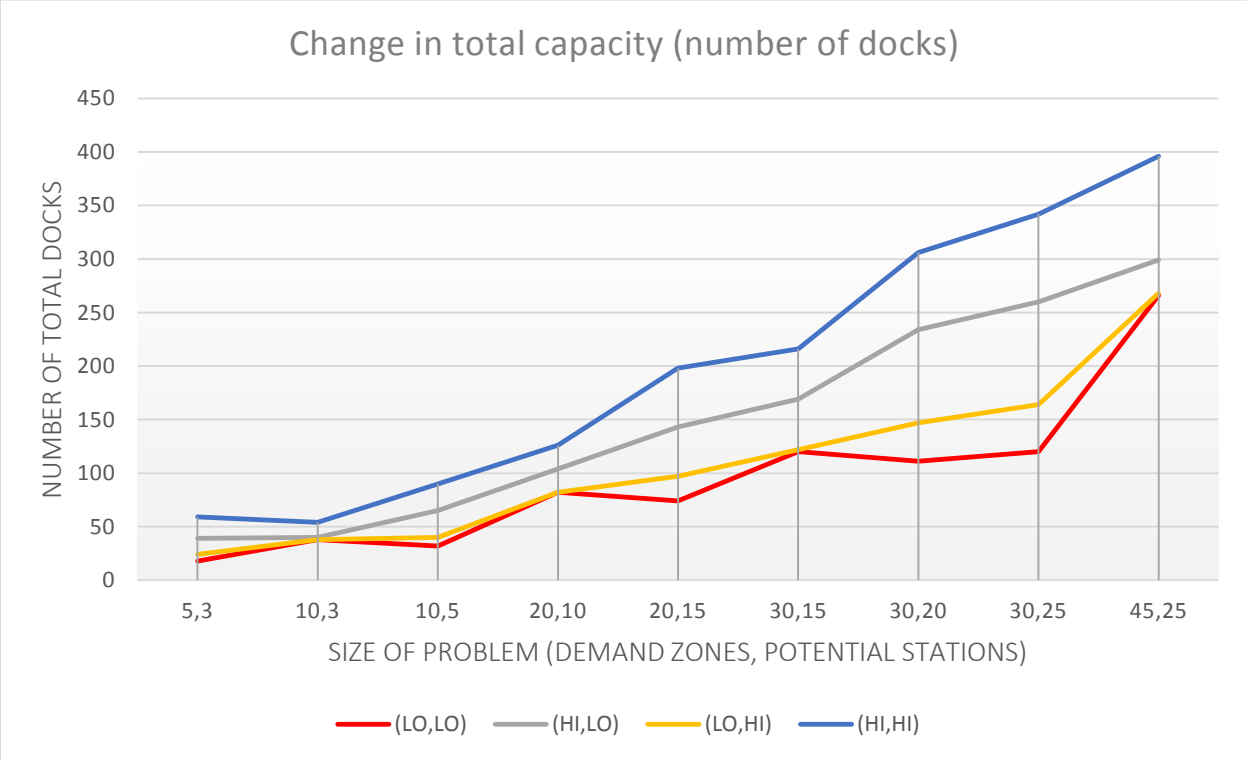


Figure 15a and 15b. Change in the number of open stations and total capacity

The very first thing to be studied in a designed bike sharing network is the number of open stations and their total capacities by having different sets of parameters and for different sizes. Obviously as the size of problem increases, the number of opened stations and total docks grows regardless of the parameter settings. what stands out in this figure is that, for the same sizes of the problem, having higher service level rates and balking for pick up and drop off results in fewer open stations and higher capacities and the effect of these parameters on capacities are more than their effect on the number of open stations. Also the trends (especially in figure 15b) show that the effect of service level rates (α, β) is more than the effect of balking rates (r, s) on both the number of open stations and total docks.

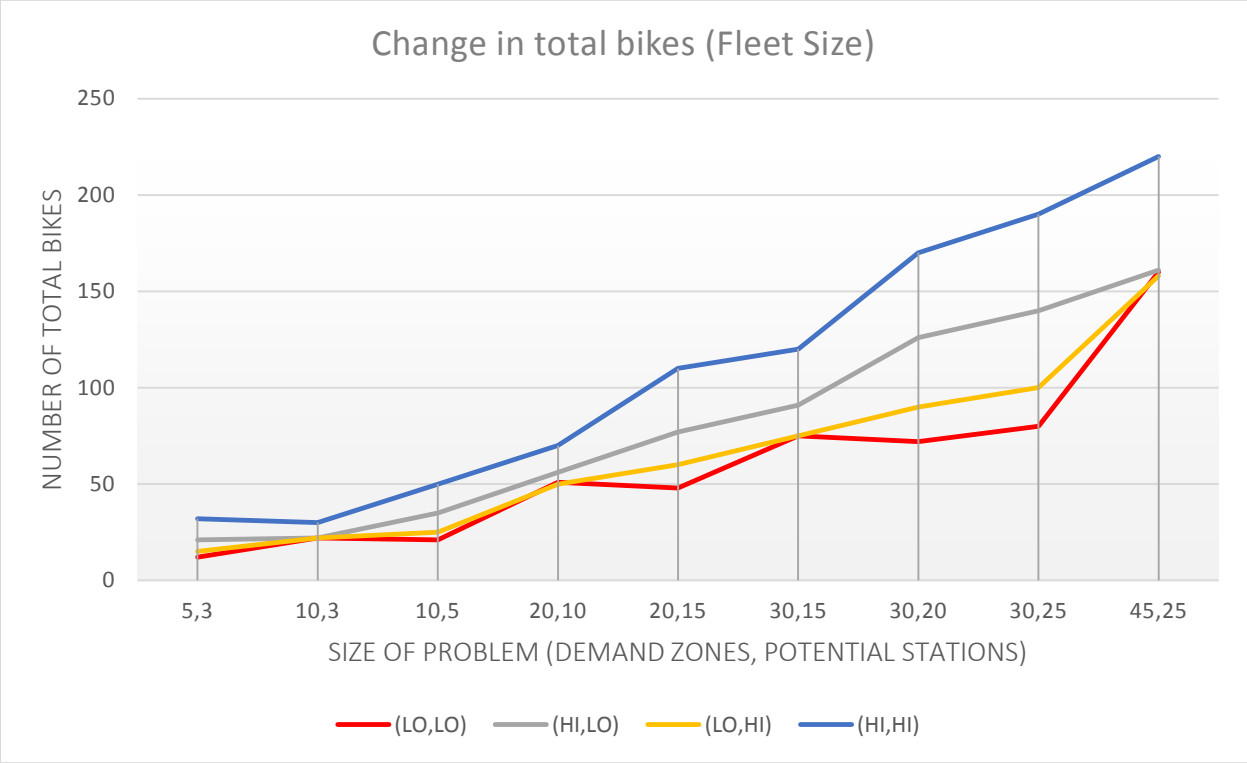


Figure 16. Change in the number of total bikes

The graph of total bikes is almost similar to the graph of total capacity as the fleet size is proportional to the number of docks. The fleet size in all instances has increased considerably as the size of problem has enlarged, and this increase rate is faster when it comes to the trend with highest service level and balking rates (the blue line). Also for the same sizes instances, as the service level and balking rates increases, the fleet size increases as well. The other important takeaway from this graph is that the effect of service level rates (α, β) is more than the effect of balking rates (r, s) on the number of total bikes. It is because the amount of increase in fleet size of gray line (with higher service level rates) compared to

the red line, is more than the increase in fleet size of orange line (with higher balking rates) compared to the red line for all examined sizes.

With regard to the fixed cost of designing the bike sharing network, as it consists of the cost of open stations, capacities and fleet size, the changes of the fixed cost are proportional to the changes of total capacities and fleet size. Thus, the main takeaways from the graph of change in total capacity and total bikes are applied for fixed cost as well.

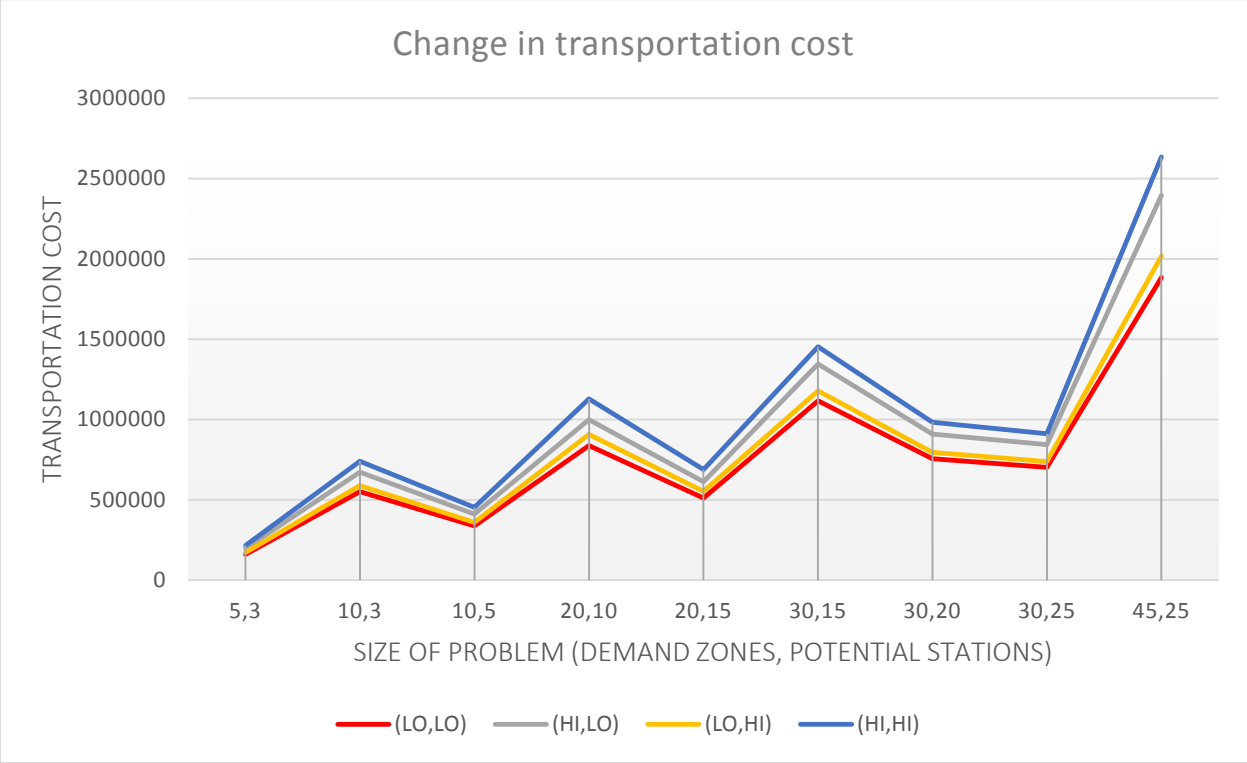


Figure 17. Change in transportation cost

Transportation cost is dependent on the way customers have been allocated to the open stations in the network, and the routes which have been used to satisfy the demand. It is notable that the changes of transportation cost for all different sets of given parameters have almost a same shape with different values. It is because the model has tried to minimize the walking distances all the time and the closest stations to each demand zone have been used to satisfy the demand of that zone for all examined sizes of the problem, and this fact is not a matter of service level or balking rates.

It is also notable that although all trends have had an increasing orientation (as the size of problem has increased), they have fluctuated considerably when the number of demand zones are the same and the

number of potential stations have increased. Actually by having more potential stations for the same number of demand zones, the transportation costs have decreased as the model has opened more stations enabling shorter routing allocations. Also for the same sizes of the problem, having higher service level and balking rates have resulted in higher transportation costs which is due to an increase in the number of satisfied customers and the effect of service level rates has been more than the effect of balking rates on transportation costs as well. It is because the amount of increase in transportation cost of gray line (with higher service level rates) compared to the red line, is more than the increase in transportation cost of orange line (with higher balking rates) compared to the red line for all instances.

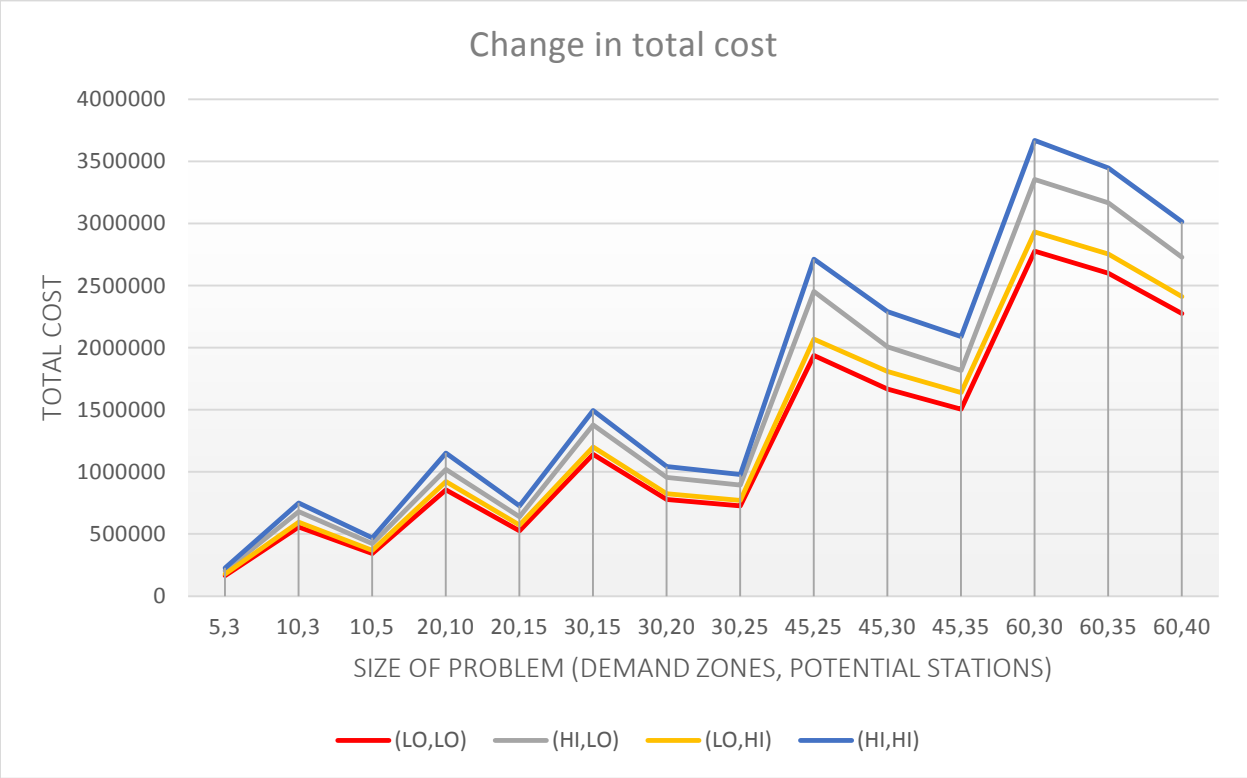


Figure 18. Change in total cost

As total cost is the value of objective function of the model, the results of five larger instances have been added to figure 18. It should be noted that for these larger instances the CPLEX was able to provide us just with the value of objective function and not the rest of results in detail (all decision variables), so the results of these five largest instances have been obtained by genetic algorithm and just added to this graph. It is notable that for the large size instances, the variance of changes in total costs increases significantly. Also as the amount of transportation costs are more than the fixed costs for all examined

instances, the graph of total cost is more similar to the graph of transportation cost and the main takeaways of figure 17 are applied here as well.

In addition, in order to visualize the bike sharing network, an instance with 20 demand zones and 10 potential stations are followed. The average monthly demand of these 20 zones (out of 100 zones) are as follows:

Zone	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84
65	0	2767	1563	576	231	51	4	315	810	1916	1562	1536	887	376	76	40	0	257	60	226
66	3075	0	4049	1371	453	87	1	125	301	905	817	1213	4157	837	154	61	0	124	52	181
67	1340	3630	0	3398	887	131	2	46	184	570	646	671	2738	2344	305	94	0	95	47	130
68	525	1337	3520	0	1354	203	1	19	111	279	306	326	1209	1399	466	130	0	56	29	79
69	75	174	403	509	0	699	1	4	18	51	52	49	221	285	900	407	0	5	3	22
70	15	36	73	85	631	0	1	5	3	22	12	17	36	60	123	344	0	1	1	11
71	3	0	0	0	0	0	0	21	4	3	1	0	0	0	0	0	0	29	2	0
72	279	113	56	20	12	7	13	0	597	562	139	37	39	22	14	3	0	623	75	125
73	790	287	195	105	34	5	2	735	0	1569	441	160	104	59	14	5	0	777	238	599
74	1802	933	661	346	128	32	4	564	1795	0	1585	599	372	207	54	29	0	572	257	1047
75	1431	829	720	318	112	22	2	143	513	1601	0	757	362	189	62	19	0	211	129	469
76	1502	1170	756	390	154	44	1	52	180	558	717	0	444	238	67	22	0	104	59	202
77	801	2969	2392	1097	508	71	1	30	104	310	315	371	0	899	217	92	0	48	20	84
78	298	664	2114	1233	756	116	0	16	52	162	158	179	947	0	607	88	0	20	10	56
79	43	78	200	228	811	127	0	20	6	28	25	33	115	248	0	560	0	3	3	4
80	11	23	48	57	347	346	0	2	2	9	9	11	40	51	366	0	0	2	0	2
81	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
82	233	92	85	56	6	4	29	646	483	403	138	72	41	21	10	9	0	0	98	143
83	58	38	44	27	6	2	1	76	264	252	100	58	21	11	3	3	0	125	0	112
84	219	143	149	94	42	26	1	121	623	960	423	156	103	69	20	8	0	164	94	0

Figure 19. Average monthly demand of 20 zones (zone 65 to 84)

Out of the results of CPLEX for different given sets of parameters, the map of Montreal with the location of 20 demand zones and 10 potential stations are followed. It should be noted that the numbers in the icon of demand zones show the zone number based on our data set, and the numbers in the parenthesis next to the icon of stations show the amount of capacity and initial bikes of each open station, respectively.

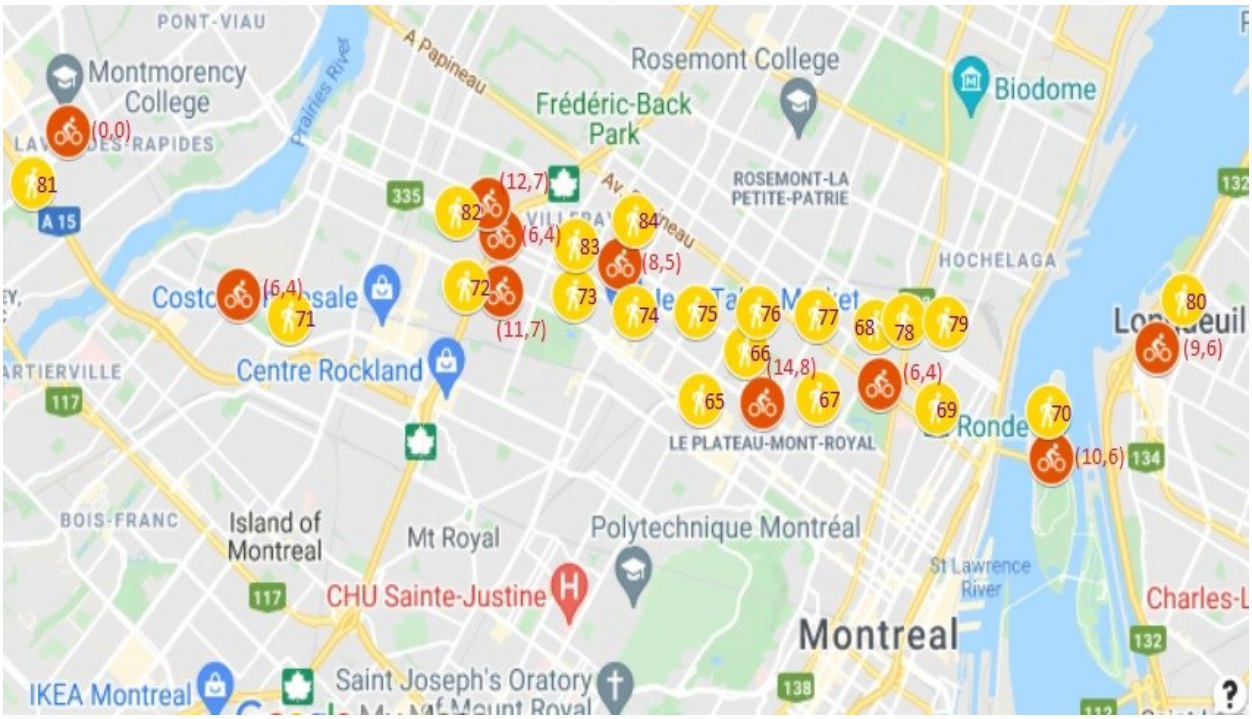


Figure 20. Bike sharing network with α=0.7, β=0.8, r=0.1, s=0.2 (LO,LO)

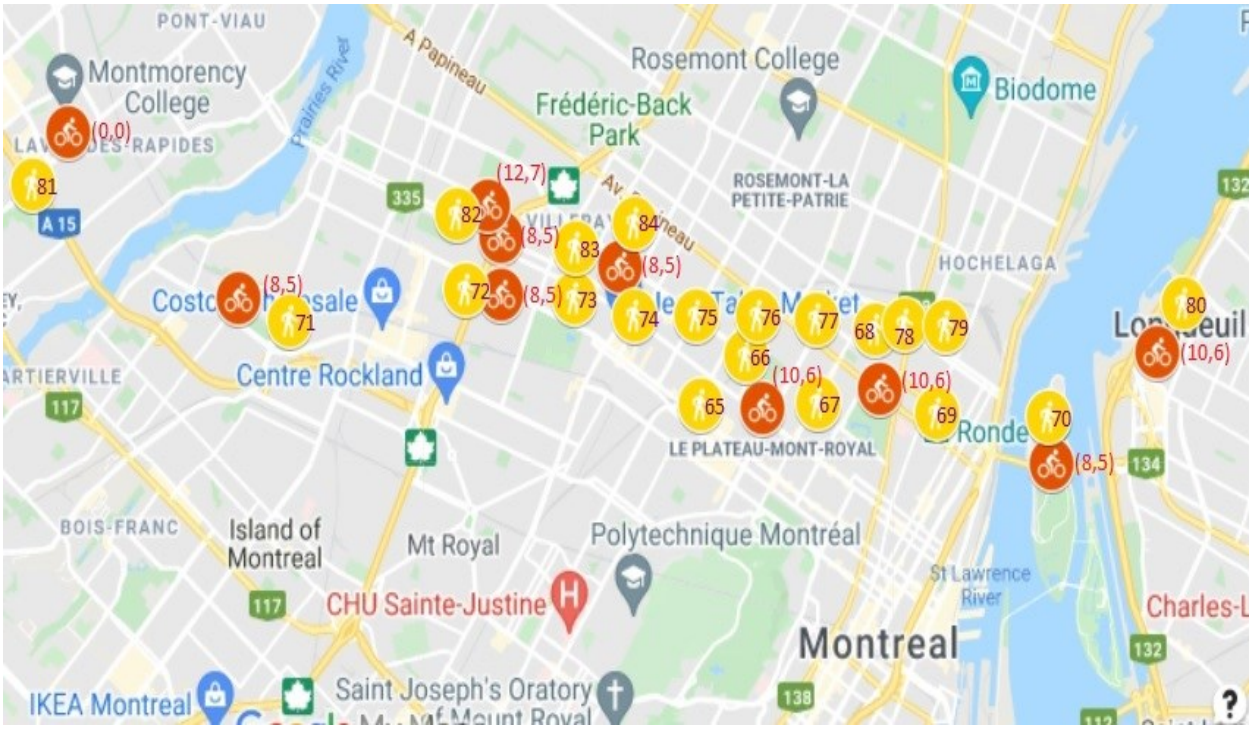


Figure 21. Bike sharing network with α=0.7, β=0.8, r=0.25, s=0.35 (LO,HI)

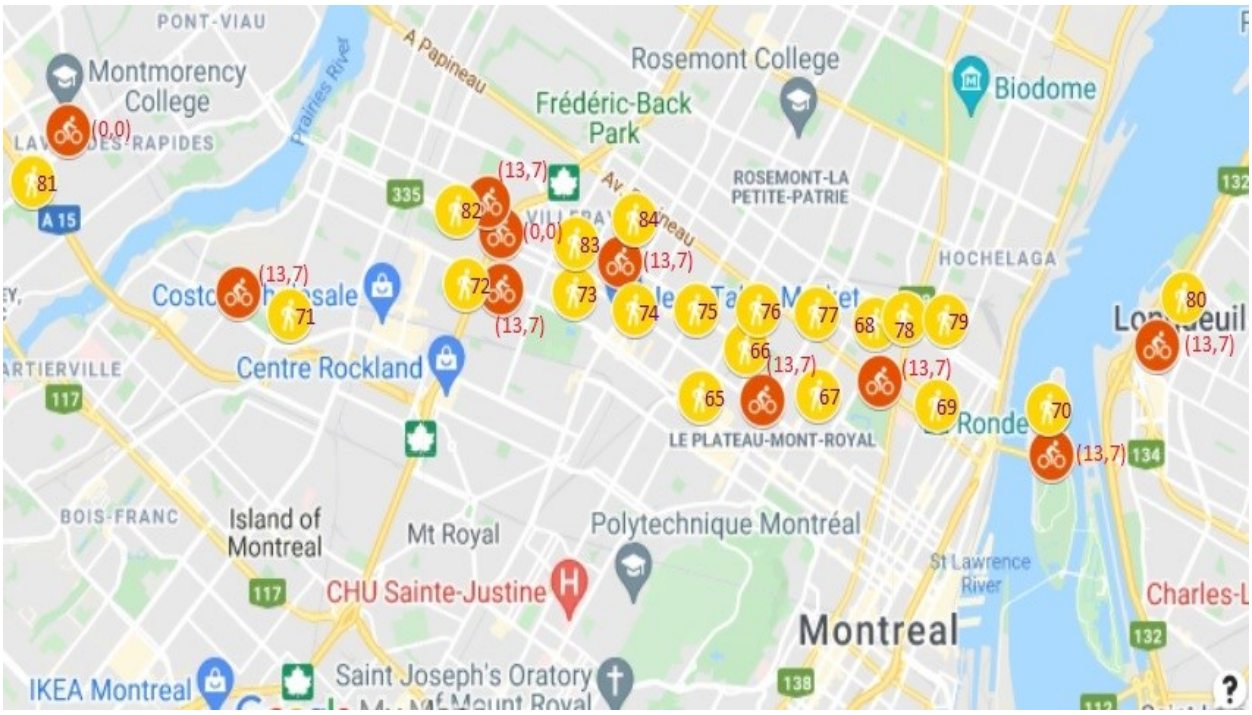


Figure 22. Bike sharing network with α=0.8, β=0.9, r=0.1, s=0.2 (HI,LO)

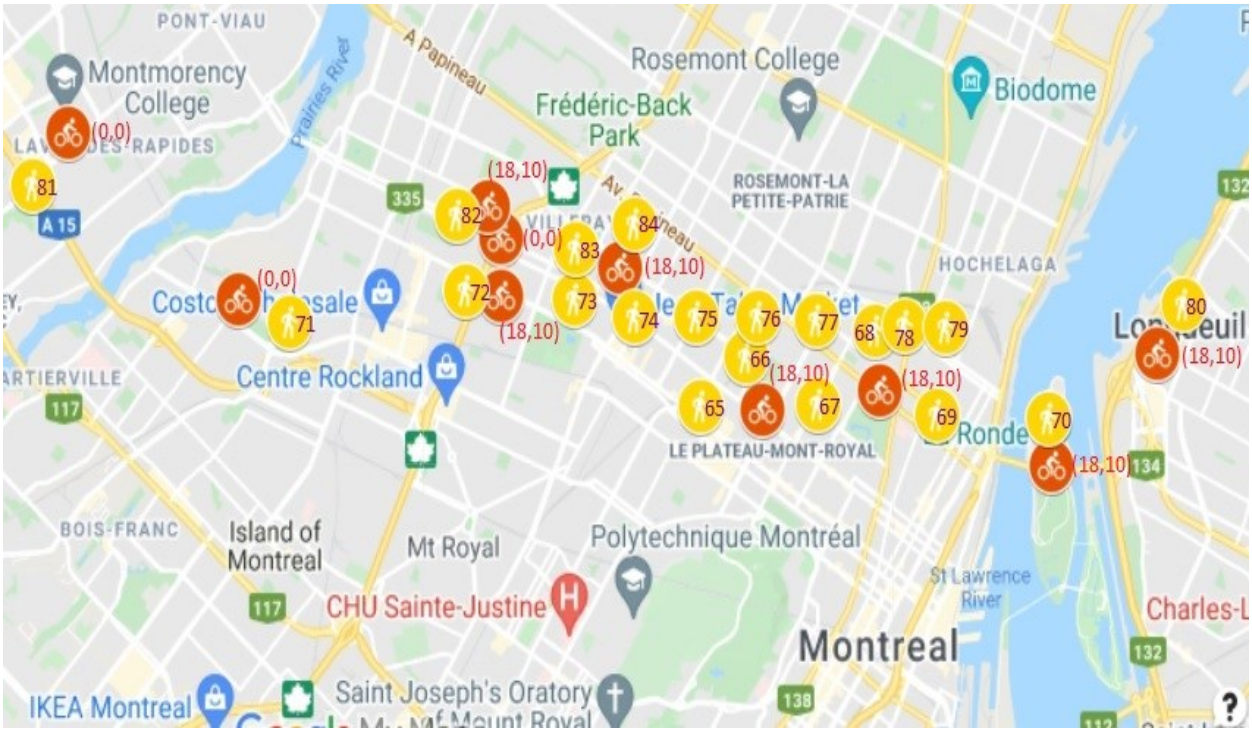


Figure 23. Bike sharing network with α=0.8, β=0.9, r=0.25, s=0.35 (HI,HI)

The main observations from figures 20 to 23 are summarized as follows:

- The number of open stations has decreased marginally as the service level and balking rates have increased. In figures 20 and 21 with α=0.7, β=0.8 (lower service level rates), 9 stations out of 10 potential stations have been opened while in figure 22 (HI,LO), the number of open stations has decreased to 8 out of 10. In figure 23 (HI,HI), we have another decrease and 7 stations out of 10 potential stations have been opened.
- The number of total docks and bikes have increased as the service level and balking rates have increased. It is notable that in figure 20, the minimum capacity and initial bikes per each open station are 6 and 4 respectively, while they have a marginal increase to 8 docks and 5 bikes, by having the same service level rates and increasing the balking rates in figure 21. They have increased to 13 docks and 7 bikes by increasing the service level rates in figure 22 and by increasing the balking rates in figure 23, they have surged to 18 and 10 respectively.

4.3. Validation of design solutions with simulation

In order to validate the network design solutions and by the help of ARENA Simulation Software Version 15.00, we have simulated the result of some experiments in small to medium sizes out of CPLEX. To this end, we have first set the ARENA run setup as following: Number of replications: 300, Time unit: minute, Replication length: 21600 minutes (30 days per month and 12 hours per each day).

Next by having the number of total pick up of bikes per each station per day (λ_b) (the result of CPLEX), and in order to set the arrival distributions in ARENA, we have converted the arrival rates to calculate them per minute ($\lambda_b/(12 * 60)$). It should be noted that the number of passengers arriving at a fixed interval of time (minute) follows the Poisson distribution. Then as these events (arrival of passengers) occur independently but continuously at a constant average rate, and as the time between events in a Poisson process follows Exponential distribution, we have set the distribution of arrivals to be Exponential with mean $\frac{1}{\lambda_b/(12*60)}$ (Haight, Frank A. 1967, Johnson, Kotz and Balakrishnan, 1994).

A simulated bike sharing network in ARENA is as following.

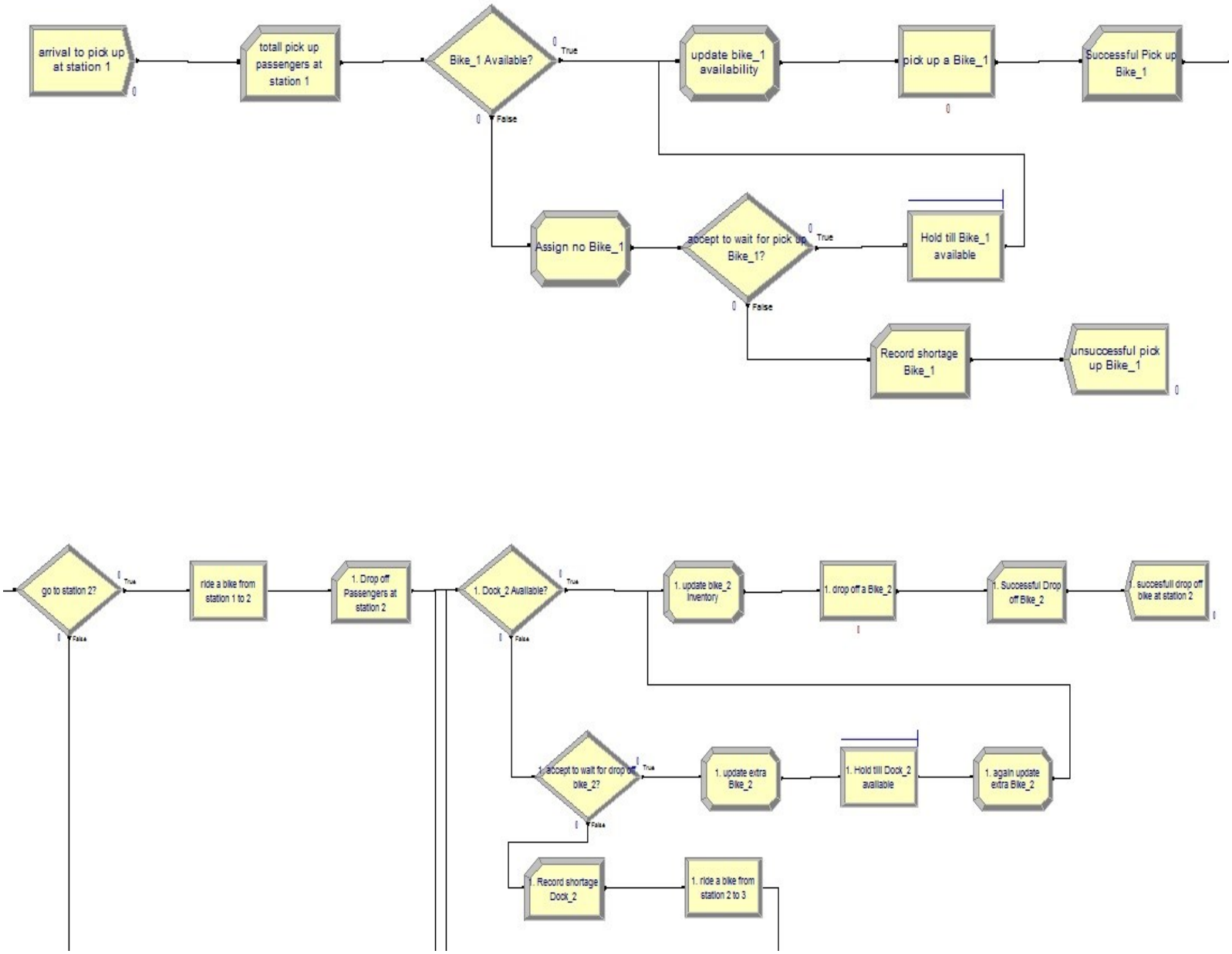


Figure 24. A simulated bike sharing network

In this simulated bike sharing network, a user arrives to station 1 and looks for a bike to pick it up. If there are any bikes available, the user picks a bike up and is counted as successful pick up, otherwise the user either waits for a bike availability by a chance (which is given as balking rate for pick up) or leaves the system and is considered as unsatisfied user in pick up side.

A satisfied user in pick up side, rides the bike towards station 2. If there are any empty docks there, the user drops the bike off and is counted as successful drop off, otherwise the user either waits for an empty dock to become available by a chance (which is given as balking rate for drop off) or rides towards the next close station which is station 3 and is considered as unsatisfied user in drop off side.

Using the decisions obtained from the proposed mixed integer programming model we have simulated the bike sharing network and compared the statistic of simulated problems with the results of CPLEX. The output statistic of all simulated samples verifies the solution of implemented experiments. The results of CPLEX and ARENA for a simulated problem with 10 demand zones and 3 possible stations and with α=0.7, β=0.8, r=0.1, s=0.2 (LO,LO) are as follows.

Output of CPLEX						
(LO,LO)	Number of docks	Number of bikes	Total allocation for pick up (monthly)	Route allocation (out of total pick up)		Total allocation for drop off (monthly)
Station 1	22	12	34607	station 1 to 2	24432	34782
				station 1 to 3	10175	
Station 2	10	6	31128	station 2 to 1	23792	31000
				station 2 to 3	7336	
Station 3	6	4	17558	station 3 to 1	10990	17511
				station 3 to 2	6568	

Output of ARENA (pick up)							
(LO,LO)	Total pick up (monthly)	Successful pick up (monthly)	Successful rate for pick up	Average waiting time for pick up (min)	Half Width of waiting time for pick up	Average number of users waiting for pick up	Half Width of number of users waiting for pick up
Station 1	34601	25317	0.732	0.86	< 0.00	0.04	< 0.00
Station 2	31102	22858	0.735	0.95	< 0.00	0.04	< 0.00
Station 3	17562	12516	0.713	1.76	< 0.01	0.05	< 0.00

Output of ARENA (drop off)							
(LO,LO)	Total drop off (monthly)	Successful drop off (monthly)	Successful rate for drop off	Average waiting time for drop off (min)	Half Width of waiting time for drop off	Average number of users waiting for drop off	Half Width of number of users waiting for drop off
Station 1	25307	25307	1.000	0	< 0.00	0.00	< 0.00
Station 2	22540	22492	0.998	0.68	< 0.05	0.0002	< 0.00
Station 3	12828	12466	0.972	1.31	< 0.025	0.0028	< 0.00

Figure 25. Result of a simulated problem

In this example, the successful rates for pick up and drop off in all stations show that the service levels have been satisfied. Also, the average waiting times of passengers in queues and the number of users waiting in the network in queues shows the consistency of the designed network of bike sharing. The average waiting times to pick up or drop off a bike are less than 2 minutes in all stations. Also the average number of people waiting in queues is very small, and the half width of both waiting time and number of customers in queues (for pick up and drop off), proves the confidence level of the designed network and this fact that the number of replications is big enough to trust the statistic of the simulated network. The result of ARENA (in detail) for this instance (with 10 demand zones and 3 potential stations) has been added to the appendix as well.

4.4. Computational Results

In this section, by presenting numerical experiments, we evaluate the performance of the proposed genetic algorithm. The criterion used to measure the performance of the proposed meta-heuristic algorithm is the GAP factor, which is calculated using the following equation. In this regard, O_{MH} means the value of the objective function of the metaheuristic method and O_{opt} means the optimal value of the exact method.

$$\left(\frac{(O_{MH} - O_{opt})}{O_{opt}}\right) \times 100$$

Numerical results for different sizes of the problem are presented in the following tables. It should be noted that for each of these dimensions, four problems (with different values of genetic algorithm parameters) have been evaluated and solved.

Table 4. Comparison of the performance of optimum method and genetic algorithm; (LO,LO)

$\alpha = 0.7$, $\beta = 0.8$ $r = 0.1$, $s = 0.2$		Visual Studio, C++ (Genetic Algorithm)						CPLEX (Optimum Method)		
Number of Zones	Number of Stations	Minimum Gap (%)	Maximum Gap (%)	Average Gap (%)	Minimum Time (Sec)	Maximum Time (Sec)	Average Time (Sec)	Minimum Time (Sec)	Maximum Time (Sec)	Average Time (Sec)
5	3	0.4	0.98	0.7	1.00	2.12	1.6	2.16	4.73	3.4
10	3	0.48	1.7	1.1	1.83	3.74	2.8	2.31	5.02	3.7
10	5	0.9	2.2	1.6	5.92	8.32	7.1	2.61	5.16	3.9
20	10	1.9	2.7	2.3	11.38	37.93	24.7	17.95	22.38	20.2
20	15	2.1	3.5	2.8	33.01	39.57	36.3	30.53	36.82	33.7
30	15	2.9	4.3	3.6	24.67	61.80	43.2	116.66	124.72	120.7
30	20	4.0	4.3	4.2	39.81	66.78	53.3	593.09	619.76	606.4
30	25	3.2	3.6	3.4	40.34	64.62	52.5	727.36	781.49	754.4
45	25	3.8	4.2	4.0	68.05	86.44	77.2	3512.77	3755.46	3634.1
45	30	4.9	5.2	5.1	125.8	153.69	139.7	14212	14862	14537
45	35	5.0	5.9	5.5	142.37	176.62	159.5	30157	36125	33141
60	30	5.9	6.1	6.0	801	1274	1037.5	43834	45218	44526
60	35	4.9	5.3	5.1	560	1740	1150.0	59696	61521	60608
60	40	6.0	6.3	6.2	674	2129	1401.5	100315	108413	104364
Average		3.7 %			310 Secs			18740 Secs		

Table 5. Comparison of the performance of optimum method and genetic algorithm; (HI,LO)

$\alpha = 0.8$, $\beta = 0.9$ $r = 0.1$, $s = 0.2$		Visual Studio, C++ (Genetic Algorithm)						CPLEX (Optimum Method)		
Number of Zones	Number of Stations	Minimum Gap (%)	Maximum Gap (%)	Average Gap (%)	Minimum Time (Sec)	Maximum Time (Sec)	Average Time (Sec)	Minimum Time (Sec)	Maximum Time (Sec)	Average Time (Sec)
5	3	0.7	1.0	0.9	1.21	2.57	1.9	2.04	4.2	3.1
10	3	0.9	1.2	1.1	2.98	4.13	3.6	2.24	5.22	3.7
10	5	1.7	1.9	1.8	4.39	8.63	6.5	2.35	5.31	3.8
20	10	2.3	3.5	2.9	12.47	55.38	33.9	7.05	11.42	9.2
20	15	2.9	3.2	3.1	59.74	71.13	65.4	19.54	25.73	22.6
30	15	3.9	4.2	4.1	34.00	73.67	53.8	67.11	75.48	71.3
30	20	4.3	4.6	4.5	51.91	101.15	76.5	373.52	381.88	377.7
30	25	2.9	3.1	3.0	47.02	72.19	59.6	2688.26	2716.55	2702.4
45	25	4.3	4.4	4.4	60.07	87.62	73.8	2868.87	2912.46	2890.7
45	30	4.9	5.1	5.0	160.09	192.55	176.3	35080	39178	37129
45	35	4.5	4.8	4.7	423.92	520.44	472.2	48538	52955	50746
60	30	5.7	6.0	5.9	391.43	895.54	643.5	75960	83670	79815
60	35	5.7	6.2	6.0	725.77	1563.22	1144.5	93960	97388	95674
60	40	5.7	6.4	6.1	1327.45	1713.32	1520.4	118419	125099	121759
Average		3.8 %			309 Secs			27943 Secs		

Table 6. Comparison of the performance of optimum method and genetic algorithm; (LO, HI)

$\alpha = 0.7$, $\beta = 0.8$ $r = 0.25$, $s = 0.35$		Visual Studio, C++ (Genetic Algorithm)						CPLEX (Optimum Method)		
Number of Zones	Number of Stations	Minimum Gap (%)	Maximum Gap (%)	Average Gap (%)	Minimum Time (Sec)	Maximum Time (Sec)	Average Time (Sec)	Minimum Time (Sec)	Maximum Time (Sec)	Average Time (Sec)
5	3	0.3	0.6	0.5	2.63	3.31	3.0	2.16	3.92	3.0
10	3	0.9	1.2	1.1	3.92	5.11	4.5	2.27	5.01	3.6
10	5	1.0	1.4	1.2	9.26	13.11	11.2	2.81	5.84	4.3
20	10	3.1	3.3	3.2	32.36	64.61	48.5	18.44	23.61	21.0
20	15	3.7	4.1	3.9	50.16	102.48	76.3	29.03	34.83	31.9
30	15	2.6	2.9	2.8	61.39	82.12	71.8	110.31	120.79	115.6
30	20	4.2	4.4	4.3	78.44	121.78	100.1	572.71	580.12	576.4
30	25	3.3	3.7	3.5	49.87	125.22	87.5	964.95	972.74	968.8
45	25	3.8	4.4	4.1	105.66	246.33	176.0	6071.42	6202.76	6137.1
45	30	5.1	5.3	5.2	193.89	278.67	236.3	25657	28031	26844
45	35	5.4	5.8	5.6	258.54	304.56	281.6	52101	54878	53489
60	30	6.0	6.3	6.2	487.31	499.56	493.4	67575	70230	68902
60	35	5.3	6.1	5.7	983.35	1380.49	1181.9	83389	89951	86670
60	40	6.4	6.7	6.6	453.31	605.31	529.3	107930	114866	111398
Average		3.8 %			236 Secs			25369 Secs		

Table 7. Comparison of the performance of optimum method and genetic algorithm; (HI,HI)

$\alpha = 0.8$, $\beta = 0.9$ $r = 0.25$, $s = 0.35$		Visual Studio, C++ (Genetic Algorithm)						CPLEX (Optimum Method)		
Number of Zones	Number of Stations	Minimum Gap (%)	Maximum Gap (%)	Average Gap (%)	Minimum Time (Sec)	Maximum Time (Sec)	Average Time (Sec)	Minimum Time (Sec)	Maximum Time (Sec)	Average Time (Sec)
5	3	0.6	0.7	0.7	1.75	2.43	2.1	2.19	4.08	3.1
10	3	0.9	1.1	1.0	3.91	6.22	5.1	2.4	5.16	3.8
10	5	1.6	1.7	1.7	4.51	7.48	6.0	2.52	5.28	3.9
20	10	3.2	3.4	3.3	33.77	45.10	39.4	6.99	10.43	8.7
20	15	3.5	3.9	3.7	50.67	76.49	63.6	31.31	36.57	33.9
30	15	2.2	2.6	2.4	61.33	82.62	72.0	68.86	76.94	72.9
30	20	3.4	3.6	3.5	80.02	101.12	90.6	430.17	437.56	433.9
30	25	4.4	5.0	4.7	110.47	225.96	168.2	721.8	735.91	728.9
45	25	4.9	5.2	5.1	187.77	309.77	248.8	2309.88	2412.63	2361.3
45	30	5.1	5.3	5.2	253.24	326.21	289.7	27268	29813	28540
45	35	5.4	5.6	5.5	438.68	1024.54	731.6	39133	44541	41837
60	30	6.4	6.8	6.6	791.99	1080.12	936.1	58466	66112	62289
60	35	6.0	6.2	6.1	372.13	1009.06	690.6	74820	77131	75975
60	40	5.8	6.4	6.1	667.95	1041.43	854.7	96887	102233	99560
Average		4.0 %			300 Secs			22275 Secs		

The results of all instances show that the time required to solve the problem and obtain an optimal solution by CPLEX increases dramatically as the size of problem increases. However, the time needed to solve the problem using the genetic algorithm does not change significantly with the higher dimensions of the problem. A comparison of the average time needed to solve the problem using both methods is presented in the following graph (out of tables 4-7).

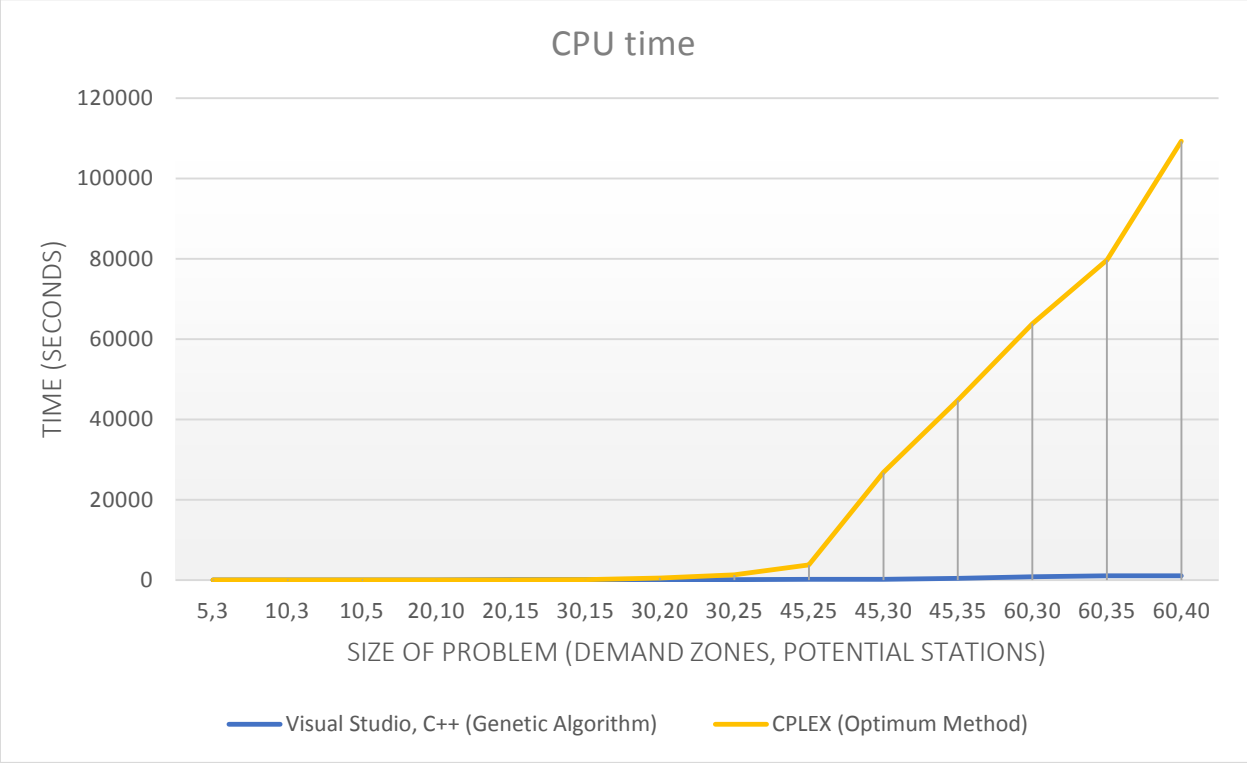


Figure 26. The average CPU time for solving the problem

The results indicate the proposed genetic algorithm finds appropriate feasible solutions in a reasonable amount of time. It is notable that, for small to medium sizes of the problem the average CPU time of both methods are almost the same, however with larger dimensions of the problem (starting 45 demand zones and 25 potential stations), the time needed to solve the problem by CPLEX has increased remarkably. The average time needed to solve any instance with genetic algorithm is 289 seconds while the average time that CPLEX required to find the optimal solution is 23582 seconds. So, the proposed genetic algorithm is about 82 times faster than the optimum method (CPLEX) on average for all reported instances. It should be noted that even for very large instances that CPLEX is not able to build the problem or reach any result, the proposed genetic algorithm provides us with feasible solutions in a reasonable time.

The following graph shows the average gap between the genetic algorithm solutions and optimal solutions (out of tables 4-7).

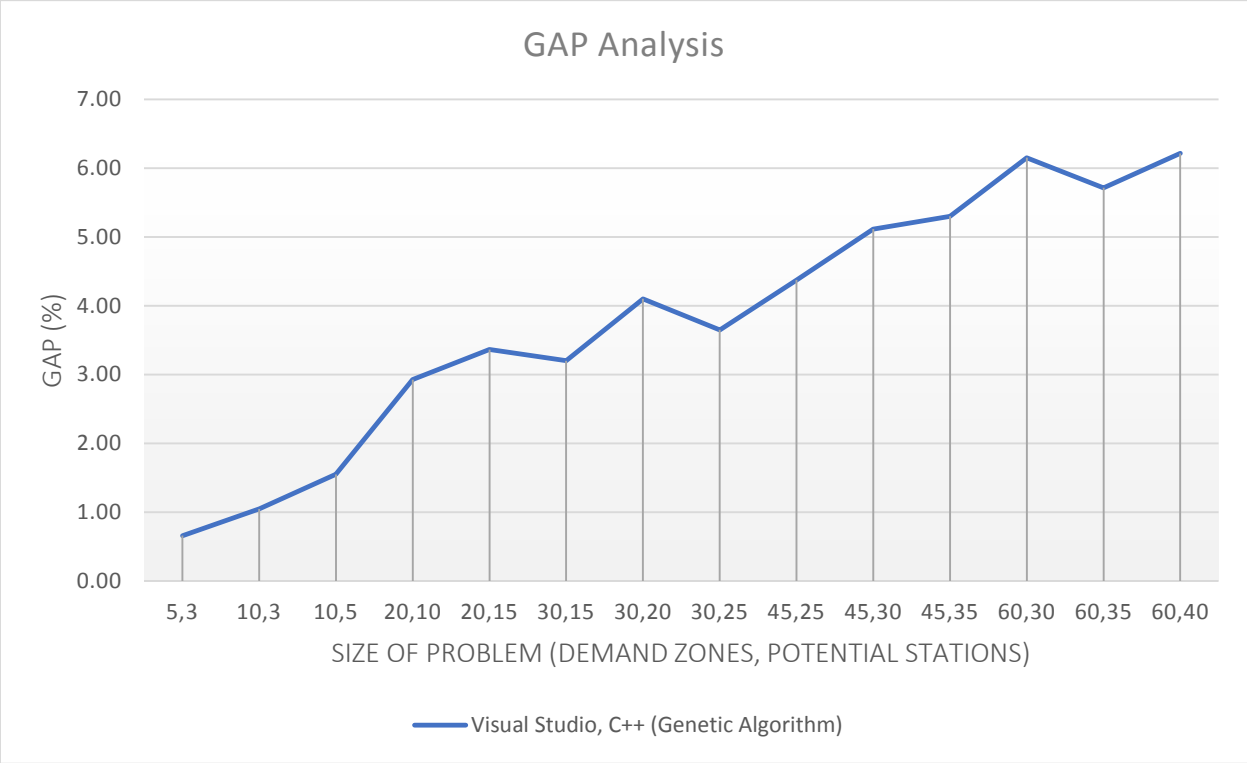


Figure 27. Gap analysis of genetic algorithm solutions

It is notable that, the average gap between the genetic algorithm solutions and optimal solutions has increased slightly as the dimension of the problem has increased. The average gap is about 3.8% for all reported instances and it has started from 0.66% for the smallest size with 5 demand zones and 3 potential stations and has ended to 6.21% for the largest size with 60 demand zones and 40 potential stations.

In summary, the proposed genetic algorithm performs very well for all given sets of parameters and is able to gain suitable solutions in a reasonable amount of time. Its performance is remarkable specially when the size of problem increases and it is about 82 times faster than the optimum method (CPLEX). We also show that our method provides solutions with acceptable gaps even for the large size instances of problem.

Chapter 5 - Conclusion and Future Research Directions

The objective of this study was to design a bike sharing network considering the service level of pick up and drop off bikes in all stations, which had not been simultaneously studied in previous works. For a given set of origin and destination points for travelers and the amount of demand in each origin point, a mixed integer programming model was developed in order to determine the location of bike stations and their capacities so as to minimize the total cost of opening stations and passengers' transportation. In this model it was crucial to know which routes should be selected considering the distances from demand zones to stations and the capacity level of each station to serve with a desired level of bike and dock availability. We also considered balking (probability of waiting) of users for an available bike or an empty dock in all stations, and investigated their impact on the capacity and number of bicycles in the opened stations and selected routes in the network.

To overcome the computational complexity of the problem, we developed a genetic algorithm method to solve the problem in large size instances and provided illustrative examples to examine the performance of our proposed solution method. By presenting the numerical results for different sizes of the problem with various instance sets of parameters, we compared the performance of our meta-heuristic method and the one resulted by CPLEX, in terms of the computation time and optimality gap.

In order to validate the network design solutions, we simulated the network design obtained in small to medium sizes using Arena simulation software. All simulated instances provided equivalent performance levels compared to the conditions stated in the optimization model, achieving acceptable pick up and drop off service levels. In addition, the average waiting times of passengers in queues and the number of users waiting in the network in queues indicated the consistency of the designed network of bike sharing.

We considered the city of Montreal as our case study and in order to conduct our numerical experiments, we obtained needed data from Bixi's website (current bike sharing network of Montreal). In our experiments, the optimal solutions indicated that, as the size of problem increases, the number of opened stations, capacities, fleet size and fixed costs increases as well. However, although transportation costs trends had an increasing orientation (as the size of problem increased), by having more potential stations for the same number of demand zones, the transportation costs decreased as the model opened more stations with better routing allocations.

We also studied the effects of the main parameters on the model and the main insights from the experiments are summarized below:

(i) For the same sizes of the problem, having higher service level and balking rates resulted in fewer open stations.

(ii) For the same sizes of the problem, having higher service level and balking resulted in higher capacities, fleet sizes and costs.

(iii) The effect of service level rates was more than the effect of balking rates.

The results of all instances showed that the time required to obtain an optimal solution by CPLEX increased dramatically as the size of problem increased, however, the needed time to solve the problem using the genetic algorithm did not change significantly with increasing the dimensions of the problem. The proposed genetic algorithm was about 82 times faster than the optimum method (CPLEX) on average for all reported instances. It was also notable that for very large instances, which CPLEX was not able to reach any result within the 30 hours' time limit, the proposed genetic algorithm provided us with feasible solutions in less than half an hour. Regarding the gap between the genetic algorithm solutions and optimal solutions, it increased marginally with increasing the dimensions of the problem and the average optimality gap was about 3.8% for all instances.

This research can be expanded in many ways and future studies may consider different goals, limitations or scenarios to apply service levels. Bicycle path construction and its cost would be a suggestion to be added to the model in the future. Considering CO2 emission savings as a key factor which can be studied in bike sharing network design and may be added to the proposed model as a component in the objective function or as a constraint. Taking the operational functions like relocation of bikes between opened stations in the network and its cost, into account and combining them with the proposed model under this study can be another interesting aspect in the future works. In terms of solution methodology, using other solution methodologies like considering other heuristic or meta-heuristic methods to solve the problem and comparing the performance of them with the genetic algorithm used in this study deserves for further investigation.

Bibliography

Alizadeh M., Ma J., Mahdavi-Amiri N., Marufuzzaman M. and Jaradat R., "A stochastic programming model for a capacitated location-allocation problem with heterogeneous demands", *Computers & Industrial Engineering*, Volume 137, 106055, 2019.

Bean J. C., "Genetic Algorithms and Random Keys for Sequencing and Optimization.," *ORSA Journal on Computing,* vol. 6, no. 2, pp. 154-160, 1994.

Bonnette, B. "The implementation of a public-use bicycle program in Philadelphia". *Urban Studies Program*. Senior Seminar Papers. Pennsylvania University, 2007.

Broach, J., Dill, J., Gliebe, J., "Where do cyclists ride? A route choice model developed with revealed preference GPS data". *Transp. Res. Part A*, 46 (10) 1730–1740, 2012.

Büttner, J., Mlasowsky, H., Birkholz, T., Groper, D., Fernandez, A. C., Emberger and Banfi, M. "Optimizing Bike Sharing in European Cities: A Handbook", *Intelligent Energy Europe program*, 2011.

Caggiani, L. and Ottomanelli, M. "A dynamic simulation based model for optimal fleet repositioning in bike-sharing systems". *Procedia - Social and Behavioral Sciences.* Vol. 87, No. 1, pp. 203-210, 2013.

Çelebi D., Yörüsün A. and Işık H., "Bicycle sharing system design with capacity allocations", *Transportation Research Part B: Methodological*, Volume 114, Pages 86-98, 2018.

David C., An Introduction to Genetic Algorithms for Scientists and Engineers, World Scientific, 1999.

Fowler, R. J., Paterson, M. S., Tanimoto, S. L. "Optimal packing and covering in the plane are NP-complete", *Information Processing Letters*, **12** (3): 133–137, 1981.

Frade I. and Ribeiro A., "Bike-sharing stations: A maximal covering location approach", *Transportation Research Part A: Policy and Practice*, Volume 82, Pages 216-227, 2015.

Fricker C. and Gast N., "Incentives and redistribution in homogeneous bike-sharing systems with stations of finite capacity", *Springer-Verlag Berlin Heidelberg and EURO J Transp Logist*, pp. 261–291, 2014.

García-Palomares, J. C., Gutiérrez, J. and Latorre, M. "Optimizing the location of stations in bike sharing programs: A GIS approach". Applied Geography, Vol. 35, No. 1, pp. 235-246, 2012.

George, D.K and Xia, C.H,. "Fleet-sizing and service availability for a vehicle rental system via closed queueing networks", *European Journal of Operational Research*, Volume 211, Issue 1, Pages 198-207, 2011.

Gonzalez, T., "Clustering to minimize the maximum intercluster distance", *Theoretical Computer Science*, 38: 293–306, 1985.

Haight, Frank A., *Handbook of the Poisson Distribution*, New York, NY, USA: John Wiley & Sons, 1967.

Haupt R. L. and Haupt S. E., Practical Genetic Algorithms, 2nd Edition, John Wiley & Sons Inc, 2004.

Heinen, E., van Wee, B. and Maat, K. "Commuting by bicycle: An overview of the literature", *Transport Reviews*, Vol. 30, No. 1, pp. 59-96, 2010.

Holland J. H., Adaptation in Natural and Artificial Systems., Cambridge, MA: MIT Press., 1975.

https://www.bixi.com/en/open-data

Jenn-Rong L., Ta-Hui Y. and Yu-Chung C., "A hub location inventory model for bicycle sharing system design: Formulation and solution", *Computers & Industrial Engineering*, Volume 65, Issue 1, Pages 77-86, 2013.

Johnson, Kotz and Balakrishnan, *Continuous Univariate Distributions,* Volumes I and II, 2nd. Ed., John Wiley and Sons, 1994.Kochel, P., Kunze, S. and Nielander, U., "Optimal control of a distributed service system with moving resources: Application to the fleet sizing and allocation problem", *International Journal of Production Economics*, 81–82, 443–459, 2003.

Leurent F., "Modelling a vehicle-sharing station as a dual waiting system: stochastic framework and stationary analysis.," 19 pages. 2012. <hal-00757228>

Lin, J. R. and Yang, T. H. “Strategic design of public bicycle sharing systems with service level constraints”, *Transportation Research - Part E*: Logistics and Transportation Review. Vol. 47, No. 2, pp. 284-294, 2011.

Liu H., Szeto W.Y., Long J., “Bike network design problem with a path-size logit-based equilibrium constraint: Formulation, global optimization, and matheuristic”, *Transportation Research Part E: Logistics and Transportation Review*, Volume 127, Pages 284-307, 2019.

Martens, K. “Promoting bike and ride: The Dutch experience”, *Transportation Research - Part A*: Policy and Practice. Vol. 41, No. 4, pp. 326-338, 2007.

Megiddo N. and Arie T., "On the complexity of locating linear facilities in the plane", *Operations Research Letters*, 1 (5): 194–197, 1982.

Midgley, P. “Bicycle-sharing schemes: Enhancing sustainable mobility in urban areas”, Background Paper No. 8, CSD19/2011/BP8, *Commission on Sustainable Development*, United Nations, 2011.

Nair R. and Miller-Hooks E., “Equilibrium design of bicycle sharing systems: the case of Washington D.C.”, *EURO Journal on Transportation and Logistics*, Volume 5, Issue 3, Pages 321-344, 2016.

Olariu S. and Zomaya A. Y., Handbook of Bioinspired Algorithms and Applications, *Chapman and Hall/CRC*, 2005.

Pucher, J., Komanoff, Ch., Schimek, P., “Bicycling Renaissance in North America Recent Trends and Alternative Policies to Promote Bicycling”, *Transportation Research Part A*, 33: 625, 1999.

Quan L.L, Rui N.F, and Jing Y.M, “A Unified Framework for Analyzing Closed Queueing Networks in Bike Sharing Systems”, *Information Technologies and Mathematical Modelling,* Queueing Theory and Applications, Volume 638, 2016.

Quan L.L, Rui N.F, and Zhi-Yong Q., “A Nonlinear Solution to Closed Queueing Networks for Bike Sharing Systems with Markovian Arrival Processes and Under an Irreducible Path Graph”, *Queueing Theory and Network Applications*, Volume 10591, Pages 118-140, 2017.

Shu, J., Chou, M., Liu, Q., Teo, C. P. and Wang, I. L. “Bicycle-sharing system: Deployment, utilization and the value of re-distribution”. Singapore: National University of Singapore - NUS Business School, 2010.

Sivanandam S. N. and Deepa S. N., Introduction to genetic algorithms., *Springer Berlin Heidelberg*, 2008.

Stinson, M.A., Bhat, C.R. “A comparison of the route preferences of experienced and inexperienced bicycle commuters”, *Transportation Research Board*: 110-121, 2005.

Yan S., Lin J., Chen Y. and Xie F., “Rental bike location and allocation under stochastic demands”, *Computers & Industrial Engineering*, Volume 107, Pages 1-11, 2017.

Zahedian-Tejenaki, Z. and Tavakkoli-Moghaddam, R. “A Fuzzy bi-objective mathematical model for sustainable hazmat transportation”. *International Journal of Transportation Engineering*, Vol. 2, No. 3, pp. 231-243, 2015.

Zandieh M. and Karimi N., "An adaptive multi-population genetic algorithm to solve the multi-objective group scheduling problem in hybrid flexible flowshop with sequence-dependent setup times," *Intelligent Manufacturing Systems,* vol. 22, pp. 979-989, 2010.

Appendices

CPLEX Code:

.mod

```
/* Sets */
int i = ...;     /*number of rider zones*/
range I = 1..i;  /*origin of rider*/
int j = ...;     /*number of rider zones*/
range J = 1..j;  /*destination of rider*/
int b = ...;     /*number of potential bike stations*/
range B = 1..b;  /*pick up station*/
int l = ...;     /*number of potential bike stations*/
range L = 1..l;  /*drop off station*/
int k = ...;     /*maximum possibility of capacity level*/
range K = 6..k;  /*capacity level of a specific bike station*/

/* Parameters */
int capital_lamda[I][J] = ...;      /*monthly customer demand from origin point i to destination point j*/
float d[I][B] = ...;                /*distance from origin point i to pick up station b*/
float d_prime[B][L] = ...;          /*distance from pick up station b to drop off station l*/
float d_double_prime[L][J] = ...;   /*distance from drop off station l to destination point j*/
float c = ...;                      /*unit walking cost from each zone (i or j) to each station (b or l) (per meter per trip)*/
int f[B][K] = ...;                  /*monthly fixed cost of locating any station at b with capacity level k*/
```

```
int t = ...;                    /*number of days per month*/

int n = ...;                    /*number of active hours per day*/

float p = ...;                  /*minimum possible value for Mu[b]/Lamda[b] in each station to
satisfy service level constraints*/

float r = ...;                  /*maximum possible value for Mu[b]/Lamda[b] in each station to
satisfy service level constraints*/

int e = ...;                    /*monthly fixed cost of providing each bike in the network*/

int g = ...;                    /*riding speed of a passenger by bike (meters per hour)*/


/* Decision Variables */

dvar boolean X[B][K];           /*equals 1 if station b is opened with capacity level k and 0
otherwise*/

dvar boolean Y[I][B][L][J];     /*equals 1 if customers travel from point i to j using stations b
and l, and 0 otherwise*/

dvar float+ Lamda[B];           /*daily arrival rate of customers to pick up bike from station b*/

dvar float+ Mu[B];              /*daily arrival rate of customers to drop off bike at station b*/

dvar int S[B];                  /*number of initial bikes in each station*/



/* Mathematical Model */

minimize sum(i in I, b in B, l in L, j in J) c*d[i][b]*Y[i][b][l][j]*capital_lamda[i][j] + sum(i in I,
b in B, l in L, j in J) c*d_double_prime[l][j]*Y[i][b][l][j]*capital_lamda[i][j] + sum(b in B, k in
K) f[b][k]*X[b][k] + sum(b in B) e*S[b];


subject to {


        forall(i in I, j in J : i!=j)
```

```
        Const2: sum(b in B, l in L : b!=l) Y[i][b][l][j] == 1;

    forall(i in I, j in J : i!=j, b in B)

        Const3: sum(l in L : b!=l) Y[i][b][l][j] + sum(l in L : b!=l) Y[i][l][b][j] <= sum(k
in K) X[b][k];


    forall(b in B)

        Const4: sum(k in K) X[b][k] <= 1;


    forall(b in B)

       Const5: Lamda[b] == (1/t)*sum(i in I, j in J : i!=j, l in L : b!=l)
Y[i][b][l][j]*capital_lamda[i][j];


    forall(b in B)

       Const6: Mu[b] == (1/t)*sum(i in I, j in J : i!=j, l in L : b!=l)
Y[i][l][b][j]*capital_lamda[i][j];


       Const7: sum(b in B) S[b] >= sum(i in I, b in B, l in L, j in J)
(d_prime[b][l]*Y[i][b][l][j]*capital_lamda[i][j]) / (t*n*g);


    forall(b in B)

       Const8: S[b] >= ((sum(k in K) (X[b][k]*k))/2 + sum(k in K) X[b][k] - 0.5);


    forall(b in B)

       Const9: S[b] <= ((sum(k in K) (X[b][k]*k))/2 + sum(k in K) X[b][k]);


    forall(b in B)

       Const10: Lamda[b] <= S[b] + Mu[b];
```

```
        forall(b in B)
           Const11: Mu[b] <= sum(k in K) (X[b][k]*k) - S[b] + Lamda[b];

        forall(b in B)
           Const12: Lamda[b] >= sum(k in K) X[b][k];

        forall(b in B)
           Const14: Mu[b] >= p * Lamda[b];

        forall(b in B)
           Const15: Mu[b] <= r * Lamda[b];


}
```

.dat

```
i=30;
j=30;
b=15;
l=15;
k=30;
SheetConnection sheet("Data_Set.xlsx");
capital_lamda from SheetRead(sheet,"Demand!BN365:EY454");
```

```
d from SheetRead(sheet,"Distance!HW1319:KI1408");

d_prime from SheetRead(sheet,"Distance!HW1413:KI1477");

d_double_prime from SheetRead(sheet,"Distance!HW1483:LH1547");

c = 0.00532; /*Assumptions: Walking speed 4700 meters per hour, Time value of users $25 per hour, So 25/4700=0.00532 (dollars/meter)*/

f from SheetRead(sheet,"Fixed_Cost!H3:AF67");

t=30;

n=12;

p=0.76938;

r=1.0551;

e=128;

g=16000;  /*Assumption: riding speed of a passenger by bike is 16000 meters per hour*/
```

C++ Code (Genetic Algorithm):

```
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
using std::fixed;
using std::ios;
using std::cerr;
#include <deque>
using namespace std;
#include <vector>
using namespace std;
#include <cmath>
#include <cstdlib>              // Contains function prototype for rand
using std::rand;

#include <time.h>
#include <iomanip>
using std::setprecision;
using std::setw;

#include <string>
using std::string;
using std::getline;
```

```
#include <fstream>                // file stream
using std::ifstream;     // input file stream
using std::ofstream;     // output file stream



#define RC_EPS 1.0e-6
#define BIG 1.0e7

ifstream DATA("DATA.txt", ios::in);
ofstream RESULT("RESULT.txt", ios::out);

//Global Variables
bool *Open_Station;
int NO_Zone, NO_Station, Min_Cap, Max_Cap, NO_Day, NO_Hour, Sum_Open_Station;
int **Demand, *Cap;
int *Init_Bicycle, *Near1, *Near2, *Near3, *Near4, *Near5;
double **SS_Dis, **ZS_Dis, UWC, *FC, Min_p, Max_r, FCB, Speed, Sum_Initial_Bike,
*Lambda, *Mu, **Pr_Dis;

template< typename T >
T Max (T x,T y) {
        T maximum = x;
        if (y > maximum)
                maximum = y;
        return maximum;
}
```

```
void DefineVariables();
void FinalFree();
double Genetic();
double Fitness(int ** , int **);
int main() {

	cout.precision(6);
	double Total_Cost;
	//time_t start, end;
	//time (&start);

	try {
		cout << "Please Enter the Value of the Following Parameters:\nNumber of Excecution: ";
		int NoE = 1; cin >> NoE;
		for (int problem = 1; problem <= NoE; problem++) {
			//RESULT << "Problem " << problem  << " : ";
			DefineVariables();
			Total_Cost = Genetic();
			FinalFree();
			//time (&end);
			//RESULT << "time = " << difftime(end, start) << endl;
			RESULT << "Total Cost = " << fixed << Total_Cost << endl;
			RESULT << "---------------------------------" << endl;
			DATA.clear();
```

```
                DATA.seekg(0);
            }
        }

        catch (...) {
            cerr << "Error" << endl;
        }

        cout << "\n\t\t\"Please See the RESULT File.\"\n\n";
        return 0;
}


void DefineVariables() {

        register int i, j;
        DATA >> NO_Zone >> NO_Station >> Min_Cap >> Max_Cap;

        Demand = new int*[NO_Zone + 1];
        for(i = 1; i <= NO_Zone; i++){
            Demand[i] = new int[NO_Zone + 1];
            for(j = 1; j <= NO_Zone; j++)
                DATA >> Demand[i][j];
        }
```

```
ZS_Dis = new double*[NO_Zone + 1];
for(i = 1; i <= NO_Zone; i++){
        ZS_Dis[i] = new double[NO_Station + 1];
        for(j = 1; j <= NO_Station; j++)
                DATA >> ZS_Dis[i][j];
}

SS_Dis = new double*[NO_Station + 1];
for(i = 1; i <= NO_Station; i++){
        SS_Dis[i] = new double[NO_Station + 1];
        for(j = 1; j <= NO_Station; j++)
                DATA >> SS_Dis[i][j];
}

DATA >> UWC;
FC = new double[Max_Cap + 1];
FC[0] = 0;
for(i = Min_Cap; i <= Max_Cap; i++)
        DATA >> FC[i];

DATA >> NO_Day >> NO_Hour >> Min_p >> Max_r >> FCB >> Speed;

Near1 = new int[NO_Zone + 1];
Near2 = new int[NO_Zone + 1];
Near3 = new int[NO_Zone + 1];
```

```
Near4 = new int[NO_Zone + 1];
Near5 = new int[NO_Zone + 1];
Cap = new int[NO_Station + 1];
Open_Station = new bool[NO_Station + 1];
Init_Bicycle = new int[NO_Station + 1];
Lambda = new double[NO_Station + 1];
Mu = new double[NO_Station + 1];



for(i = 1; i <= NO_Zone; i++){
        Near1[i] = Near2[i] = 1;
        for(j = 2; j <= NO_Station; j++){
                if(ZS_Dis[i][j] < ZS_Dis[i][Near1[i]]){
                        Near2[i] = Near1[i];
                        Near1[i] = j;
                }
        }
        if(Near1[i] != 1){
        for(j = Near1[i] + 1; j <= NO_Station; j++)
                if(ZS_Dis[i][j] < ZS_Dis[i][Near2[i]])
                        Near2[i] = j;
        }
        else{
                Near2[i] = 2;
```

```
                for(j = 3; j <= NO_Station; j++)
                        if(ZS_Dis[i][j] < ZS_Dis[i][Near2[i]])
                                Near2[i] = j;
        }
    }

    for(i = 1; i <= NO_Zone; i++){
        Near3[i] = 0;
        for(j = 1; j <= NO_Station; j++){
            if (ZS_Dis[i][j] > ZS_Dis[i][Near2[i]] && (Near3[i] == 0 || ZS_Dis[i][j] <
ZS_Dis[i][Near3[i]]))
                Near3[i] = j;
        }
        if (Near3[i] == 0){
            for(j = 1; j <= NO_Station; j++){
                if (ZS_Dis[i][j] == ZS_Dis[i][Near2[i]] && j != Near2[i] && j !=
Near1[i]){
                    Near3[i] = j;
                    break;
                }
            }
        }
    }
    for(i = 1; i <= NO_Zone; i++){
        Near4[i] = 0;
        for(j = 1; j <= NO_Station; j++){
```

```
                    if (ZS_Dis[i][j] > ZS_Dis[i][Near3[i]] && (Near4[i] == 0 || ZS_Dis[i][j] <
ZS_Dis[i][Near4[i]]))
                        Near4[i] = j;
                }
                if (Near4[i] == 0){
                    for(j = 1; j <= NO_Station; j++){
                        if (ZS_Dis[i][j] == ZS_Dis[i][Near3[i]] && j != Near3[i] && j !=
Near2[i] && j != Near1[i]){
                            Near4[i] = j;
                            break;
                        }
                    }
                }
        }
        for(i = 1; i <= NO_Zone; i++){
            Near5[i] = 0;
            for(j = 1; j <= NO_Station; j++){
                if (ZS_Dis[i][j] > ZS_Dis[i][Near4[i]] && (Near5[i] == 0 || ZS_Dis[i][j] <
ZS_Dis[i][Near5[i]]))
                    Near5[i] = j;
            }
            if (Near5[i] == 0){
                for(j = 1; j <= NO_Station; j++){
                    if (ZS_Dis[i][j] == ZS_Dis[i][Near4[i]] && j != Near4[i] && j !=
Near3[i] && j != Near2[i] && j != Near1[i]){
                        Near5[i] = j;
                        break;
```

```
                        }
                    }
                }
            }

            Pr_Dis = new double*[NO_Zone + 1];
            for(i = 1; i <= NO_Zone; i++)
                Pr_Dis[i] = new double[6];

            for(i = 1; i <= NO_Zone; i++){
                Pr_Dis[i][0] = 1 / ZS_Dis[i][Near1[i]] + 1 / ZS_Dis[i][Near2[i]] + 1 /
ZS_Dis[i][Near3[i]] + 1 / ZS_Dis[i][Near4[i]] + 1 / ZS_Dis[i][Near5[i]];
                Pr_Dis[i][1] = (1 / ZS_Dis[i][Near1[i]]) / Pr_Dis[i][0]; Pr_Dis[i][2] = (1 /
ZS_Dis[i][Near2[i]]) / Pr_Dis[i][0];
                Pr_Dis[i][3] = (1 / ZS_Dis[i][Near3[i]]) / Pr_Dis[i][0]; Pr_Dis[i][4] = (1 /
ZS_Dis[i][Near4[i]]) / Pr_Dis[i][0];
                Pr_Dis[i][5] = (1 / ZS_Dis[i][Near5[i]]) / Pr_Dis[i][0];
                for(j = 2; j <= 5; j++)
                    Pr_Dis[i][j] = Pr_Dis[i][j - 1] + Pr_Dis[i][j];
            }
}

double Genetic() {
    register int counter, g, h, j, i;

    /*const*/ int Population_Size = 30; cout << "Population Size: "; cin >> Population_Size;
```

```
	/*const*/ double Crossover_Rate = 1.5; cout << "Crossover Rate: "; cin >>
Crossover_Rate;

	/*const*/ double Mutation_Rate = 0.1; cout << "Mutation Rate (between 0 and 1): "; cin
>> Mutation_Rate;

	int NO_Offspring = (int) floor(Population_Size * Crossover_Rate);


	time_t T;

	int TEMP, THE_BEST, Generation = 1, start, end, parent1, parent2, candid,
Generation_Without_Improvement = 0;

	double THE_BEST_Fitness, Before, After, TEMPD;

	bool Heavy_Mutation_flag;


	int ***P_Chromosome = new int**[Population_Size + 1];

	for(counter = 1; counter <= Population_Size; counter++){

		P_Chromosome[counter] = new int*[NO_Zone + 1];

		for(g = 1; g <= NO_Zone; g++)

			P_Chromosome[counter][g] = new int[NO_Zone + 1];

	}

	int ***D_Chromosome = new int**[Population_Size + 1];

	for(counter = 1; counter <= Population_Size; counter++){

		D_Chromosome[counter] = new int*[NO_Zone + 1];

		for(g = 1; g <= NO_Zone; g++)

			D_Chromosome[counter][g] = new int[NO_Zone + 1];

	}


	int ***New_P_Chromosome = new int**[NO_Offspring + 1];
```

```
for(counter = 1; counter <= NO_Offspring; counter++){
        New_P_Chromosome[counter] = new int*[NO_Zone + 1];
        for(g = 1; g <= NO_Zone; g++)
                New_P_Chromosome[counter][g] = new int[NO_Zone + 1];
}
int ***New_D_Chromosome = new int**[NO_Offspring + 1];
for(counter = 1; counter <= NO_Offspring; counter++){
        New_D_Chromosome[counter] = new int*[NO_Zone + 1];
        for(g = 1; g <= NO_Zone; g++)
                New_D_Chromosome[counter][g] = new int[NO_Zone + 1];
}

double *Chromosome_Fitness = new double[Population_Size + 1];

int **P_Offspring = new int*[NO_Zone + 1];
for(g = 1; g <= NO_Zone; g++)
        P_Offspring[g] = new int[NO_Zone + 1];

int **D_Offspring = new int*[NO_Zone + 1];
for(g = 1; g <= NO_Zone; g++)
        D_Offspring[g] = new int[NO_Zone + 1];

double *Candid_Fitness = new double[NO_Offspring + 1];
int *Candid_List = new int[NO_Offspring + 1];
```

```
T = time(0);
srand( (unsigned int) T );

/************************************************************
*
                *
*                                       Initial Population
*
*
                *
************************************************************/

for(i = 1; i <= NO_Zone; i++){
        for(j = 1; j < i; j++){
                if(Near1[i] != Near1[j]){
                        P_Chromosome[1][i][j] = Near1[i];
                        D_Chromosome[1][i][j] = Near1[j];
                }
                else if (ZS_Dis[i][Near1[i]] + ZS_Dis[j][Near2[j]] < ZS_Dis[i][Near2[i]]
+ ZS_Dis[j][Near1[j]]){
                        P_Chromosome[1][i][j] = Near1[i];
                        D_Chromosome[1][i][j] = Near2[j];
                }
                else{
                        P_Chromosome[1][i][j] = Near2[i];
                        D_Chromosome[1][i][j] = Near1[j];
                }
```

```
            }
            for(j = i + 1; j <= NO_Zone; j++){
                if(Near1[i] != Near1[j]){
                    P_Chromosome[1][i][j] = Near1[i];
                    D_Chromosome[1][i][j] = Near1[j];
                }
                else if (ZS_Dis[i][Near1[i]] + ZS_Dis[j][Near2[j]] < ZS_Dis[i][Near2[i]]
+ ZS_Dis[j][Near1[j]]){
                    P_Chromosome[1][i][j] = Near1[i];
                    D_Chromosome[1][i][j] = Near2[j];
                }
                else{
                    P_Chromosome[1][i][j] = Near2[i];
                    D_Chromosome[1][i][j] = Near1[j];
                }
            }
        }

        /*************************************************/

        for (counter = 2; counter <= Population_Size; counter++){
            for (i = 1; i <= NO_Zone; i++){
                for (j = 1; j <= NO_Zone; j++){
                    if (i == j) continue;
                    TEMPD = (double)rand()/(double)(RAND_MAX) - RC_EPS;
                    g = 1;
```

```
                    while(Pr_Dis[i][g] < TEMPD) g++;
                    switch(g){
                        case 1: P_Chromosome[counter][i][j] = Near1[i]; break;
                        case 2: P_Chromosome[counter][i][j] = Near2[i]; break;
                        case 3: P_Chromosome[counter][i][j] = Near3[i]; break;
                        case 4: P_Chromosome[counter][i][j] = Near4[i]; break;
                        case 5: P_Chromosome[counter][i][j] = Near5[i]; break;
                    }
                    do {
                        TEMPD = (double)rand()/(double)(RAND_MAX) -
RC_EPS;
                        g = 1;
                        while(Pr_Dis[j][g] < TEMPD) g++;
                        switch(g){
                            case 1: D_Chromosome[counter][i][j] = Near1[j];
break;
                            case 2: D_Chromosome[counter][i][j] = Near2[j];
break;
                            case 3: D_Chromosome[counter][i][j] = Near3[j];
break;
                            case 4: D_Chromosome[counter][i][j] = Near4[j];
break;
                            case 5: D_Chromosome[counter][i][j] = Near5[j];
break;
                        }
                    }
                    while (D_Chromosome[counter][i][j] ==
P_Chromosome[counter][i][j]);
```

```
                }
            }
        }


        for(counter = 1; counter <= Population_Size; counter++){
            Chromosome_Fitness[counter] = Fitness(P_Chromosome[counter],
D_Chromosome[counter]);
        }


        THE_BEST = 1;
        THE_BEST_Fitness = Chromosome_Fitness[1];
        for (counter = 2; counter <= Population_Size; counter++) {
            if (Chromosome_Fitness[counter] < THE_BEST_Fitness) {
                THE_BEST = counter;
                THE_BEST_Fitness = Chromosome_Fitness[counter];
            }
        }


        /*********************************************************
                *
                        *
                *                                   start of MA algorithm
            *
                *
                        *
```

```
**********************************************************/

int Stopping_Criteria = 100;
cout << "Stopping Criteria (Number of Generations without Improvement): "; cin >>
Stopping_Criteria;
while /*stopping criteria not satisfied*//* (time(0) - T <=
60*/(Generation_Without_Improvement <= Stopping_Criteria/*&&(THE_BEST_Fitness < 0)*/)
{
        Heavy_Mutation_flag = true;
        candid = 0;
        Before = THE_BEST_Fitness;
        for (counter = 1; counter <= Population_Size * Crossover_Rate; counter++) {


/***********************************************************
        *
                *
        *                                       Selection
                *
        *
                *
***********************************************************/

        parent1 = rand() %(Population_Size) + 1;
        parent2 = rand() %(Population_Size) + 1;
        if (parent2 == parent1) parent2 = Population_Size + 1 - parent1;
```

```
            if (Chromosome_Fitness[parent2] < Chromosome_Fitness[parent1]) {
                TEMP = parent1; parent1 = parent2; parent2 = TEMP;
            }


/*************************************************************
            *
                    *
            *                                       Cross over
                *
            *
                    *
*************************************************************/

            if(counter < NO_Offspring / 3){
                start = rand() %NO_Zone + 1;
                end = rand() %NO_Zone + 1;
                if (start > end) {
                    TEMP = start; start = end; end = TEMP;
                }

                for (g = 1; g < start; g++){
                    for(i = 1; i <= NO_Zone; i++){
                        P_Offspring[i][g] = P_Chromosome[parent2][i][g];
                        D_Offspring[g][i] = D_Chromosome[parent2][g][i];
                    }
```

```
        }

        for (g = start; g <= end; g++){
            for(i = 1; i <= NO_Zone; i++){
                P_Offspring[i][g] = P_Chromosome[parent1][i][g];
                D_Offspring[g][i] = D_Chromosome[parent1][g][i];
            }
        }

        for (g = end + 1; g <= NO_Zone; g++){
            for(i = 1; i <= NO_Zone; i++){
                P_Offspring[i][g] = P_Chromosome[parent2][i][g];
                D_Offspring[g][i] = D_Chromosome[parent2][g][i];
            }
        }
    }
    else if(counter < 2 * NO_Offspring / 3){
        start = rand() %NO_Zone + 1;
        for (g = 1; g < start; g++)
            for(i = 1; i <= NO_Zone; i++)
                P_Offspring[i][g] = P_Chromosome[parent1][i][g];

        for (g = start; g <= NO_Zone; g++)
            for(i = 1; i <= NO_Zone; i++)
                P_Offspring[i][g] = P_Chromosome[parent2][i][g];
```

```
                start = rand() %NO_Zone + 1;
                for (g = 1; g < start; g++)
                        for(i = 1; i <= NO_Zone; i++)
                                D_Offspring[g][i] = D_Chromosome[parent1][g][i];

                for (g = start; g <= NO_Zone; g++)
                        for(i = 1; i <= NO_Zone; i++)
                                D_Offspring[g][i] = D_Chromosome[parent2][g][i];
        }
        else{
                for (g = 1; g <= NO_Zone; g++){
                        for(i = 1; i <= NO_Zone; i++){
                                TEMPD = (double)rand()/(double)(RAND_MAX);
                                if (TEMPD <= 0.5){
                                        P_Offspring[i][g] =
P_Chromosome[parent1][i][g];
                                }
                                else{
                                        P_Offspring[i][g] =
P_Chromosome[parent2][i][g];
                                }
                                TEMPD = (double)rand()/(double)(RAND_MAX);
                                if (TEMPD <= 0.5){
                                        D_Offspring[g][i] =
D_Chromosome[parent1][g][i];
```

```
                                        }
                                        else{
                                                D_Offspring[g][i] =
D_Chromosome[parent2][g][i];
                                        }
                                }
                        }
                }


        /*************************************************************
                *
                        *
                *                                               Light Mutation
                        *
                *
                        *

        *************************************************************/


                if ((double)rand()/(double)(RAND_MAX) <= Mutation_Rate) {
                        g = rand() %(NO_Zone) + 1;
                        h = rand() %(NO_Zone) + 1;
                        if (g == h) h = NO_Zone + 1 - g;
                        i = rand() %(5) + 1;
                        switch(i){
                                case 1: P_Offspring[g][h] = Near1[g]; break;
```

```
                    case 2: P_Offspring[g][h] = Near2[g]; break;
                    case 3: P_Offspring[g][h] = Near3[g]; break;
                    case 4: P_Offspring[g][h] = Near4[g]; break;
                    case 5: P_Offspring[g][h] = Near5[g]; break;
                }
                i = rand() %(5) + 1;
                switch(i){
                    case 1: D_Offspring[g][h] = Near1[h]; break;
                    case 2: D_Offspring[g][h] = Near2[h]; break;
                    case 3: D_Offspring[g][h] = Near3[h]; break;
                    case 4: D_Offspring[g][h] = Near4[h]; break;
                    case 5: D_Offspring[g][h] = Near5[h]; break;
                }
            }

            Candid_Fitness[candid + 1] = Fitness(P_Offspring, D_Offspring);


/*************************************************************
            *
                *
            *                           Insertion new individuals policy
*
            *
                *
*************************************************************/
```

```
                if (Candid_Fitness[candid + 1] < Chromosome_Fitness[parent2]) {
                        candid++;
                        Heavy_Mutation_flag = false;
                        for (i = 1; i <= NO_Zone; i++){
                                for (j = 1; j <= NO_Zone; j++){
                                        New_P_Chromosome[candid][i][j] =
P_Offspring[i][j];
                                        New_D_Chromosome[candid][i][j] =
D_Offspring[i][j];
                                }
                        }
                        Candid_List[candid] = parent2;
                }
        }//End of Crossover counter


        /*************************************************************
        *
                *
        *                                       Heavy Mutation
                *
        *
                *
        *************************************************************/

        if (Heavy_Mutation_flag == true) {
```

```
for (counter = 1; counter < THE_BEST; counter++) {
    for (i = 1; i <= NO_Zone; i++){
        for (j = 1; j <= NO_Zone; j++){
            if (i == j) continue;
            TEMPD = (double)rand()/(double)(RAND_MAX) - RC_EPS;
            g = 1;
            while(Pr_Dis[i][g] < TEMPD) g++;
            switch(g){
                case 1: P_Chromosome[counter][i][j] = Near1[i]; break;
                case 2: P_Chromosome[counter][i][j] = Near2[i]; break;
                case 3: P_Chromosome[counter][i][j] = Near3[i]; break;
                case 4: P_Chromosome[counter][i][j] = Near4[i]; break;
                case 5: P_Chromosome[counter][i][j] = Near5[i]; break;
            }
            do {
                TEMPD = (double)rand()/(double)(RAND_MAX) - RC_EPS;
                g = 1;
                while(Pr_Dis[j][g] < TEMPD) g++;
                switch(g){
                    case 1: D_Chromosome[counter][i][j] = Near1[j]; break;
```

```
                                                    case 2:
D_Chromosome[counter][i][j] = Near2[j]; break;
                                                    case 3:
D_Chromosome[counter][i][j] = Near3[j]; break;
                                                    case 4:
D_Chromosome[counter][i][j] = Near4[j]; break;
                                                    case 5:
D_Chromosome[counter][i][j] = Near5[j]; break;
                                                }
                                            }
                                            while (D_Chromosome[counter][i][j] ==
P_Chromosome[counter][i][j]);
                                        }
                                    }
                                    Chromosome_Fitness[counter] = Fitness(P_Chromosome[counter],
D_Chromosome[counter]);
                                    if (Chromosome_Fitness[counter] < THE_BEST_Fitness) {
                                        THE_BEST = counter;
                                        THE_BEST_Fitness = Chromosome_Fitness[counter];
                                    }
                                }
                                for (counter = THE_BEST + 1; counter <= Population_Size; counter++) {
                                    for (i = 1; i <= NO_Zone; i++){
                                        for (j = 1; j <= NO_Zone; j++){
                                            TEMPD = (double)rand()/(double)(RAND_MAX) -
RC_EPS;
                                            g = 1;
                                            while(Pr_Dis[i][g] < TEMPD) g++;
```

```
                switch(g){
                    case 1: P_Chromosome[counter][i][j] =
Near1[i]; break;
                    case 2: P_Chromosome[counter][i][j] =
Near2[i]; break;
                    case 3: P_Chromosome[counter][i][j] =
Near3[i]; break;
                    case 4: P_Chromosome[counter][i][j] =
Near4[i]; break;
                    case 5: P_Chromosome[counter][i][j] =
Near5[i]; break;
                }
                do {
                    TEMPD =
(double)rand()/(double)(RAND_MAX) - RC_EPS;
                    g = 1;
                    while(Pr_Dis[j][g] < TEMPD) g++;
                    switch(g){
                        case 1:
D_Chromosome[counter][i][j] = Near1[j]; break;
                        case 2:
D_Chromosome[counter][i][j] = Near2[j]; break;
                        case 3:
D_Chromosome[counter][i][j] = Near3[j]; break;
                        case 4:
D_Chromosome[counter][i][j] = Near4[j]; break;
                        case 5:
D_Chromosome[counter][i][j] = Near5[j]; break;
                    }
```

```
                                        }
                                        while (D_Chromosome[counter][i][j] ==
P_Chromosome[counter][i][j]);
                                }
                        }
                        Chromosome_Fitness[counter] = Fitness(P_Chromosome[counter],
D_Chromosome[counter]);
                        if (Chromosome_Fitness[counter] < THE_BEST_Fitness) {
                                THE_BEST = counter;
                                THE_BEST_Fitness = Chromosome_Fitness[counter];
                        }
                }
        }



        /************************************************************
        *
                *
        *                                       Updating Population
        *
        *
                *
        ************************************************************/


        for (counter = 1; counter <= candid; counter++) {
                if (Chromosome_Fitness[Candid_List[counter]] >
Candid_Fitness[counter]) {
```

```
                        Chromosome_Fitness[Candid_List[counter]] =
Candid_Fitness[counter];
                        if (Candid_Fitness[counter] < THE_BEST_Fitness) {
                                THE_BEST = Candid_List[counter];
                                THE_BEST_Fitness = Candid_Fitness[counter];
                        }
                        for (g = 1; g <= NO_Zone; g++){
                                for (j = 1; j <= NO_Zone; j++){
                                        P_Chromosome[Candid_List[counter]][g][j] =
New_P_Chromosome[counter][g][j];
                                        D_Chromosome[Candid_List[counter]][g][j] =
New_D_Chromosome[counter][g][j];
                                }
                        }
                }
        }

        After = THE_BEST_Fitness;
        if ((After - Before) >= -RC_EPS){
                Generation_Without_Improvement += 1;
        }
        else {
                Generation_Without_Improvement = 0;
        }

        //cout << Generation <<"\t"<< THE_BEST_Fitness <<"\t"<< time(0) - T <<"\n";
```

```
		//RESULT << Generation <<"\t"<< THE_BEST_Fitness <<"\t"<< time(0) - T
<<"\n";


		Generation++;
	}//End Of While


	TEMPD = Fitness(P_Chromosome[THE_BEST], D_Chromosome[THE_BEST]);


	RESULT << "Zone" <<"\t"<< "Pickup" <<"\t"<< "Drop" << "\t" << "Zone" << "\n";
	for(i = 1; i <= NO_Zone; i++){
		for(j = 1; j < i; j++)
			RESULT << i <<"\t\t"<< P_Chromosome[THE_BEST][i][j] <<"\t\t"<<
D_Chromosome[THE_BEST][i][j] << "\t\t" << j << "\n";
		for(j = i + 1; j <= NO_Zone; j++)
			RESULT << i <<"\t\t"<< P_Chromosome[THE_BEST][i][j] <<"\t\t"<<
D_Chromosome[THE_BEST][i][j] << "\t\t" << j << "\n";
	}
	RESULT << "Lambda" << "\n";
	for(i = 1; i <= NO_Station; i++)
		RESULT << Lambda[i] <<"\t";
	RESULT << "\n" << "Mu" << "\n";
	for(i = 1; i <= NO_Station; i++)
		RESULT << Mu[i] <<"\t";
	RESULT << "\n" << "Capacity" << "\n";
	for(i = 1; i <= NO_Station; i++)
		RESULT << Cap[i] <<"\t";
```

```
RESULT << "\n" << "Initial Bicycle" << "\n";
for(i = 1; i <= NO_Station; i++)
        RESULT << Init_Bicycle[i] <<"\t";
RESULT << "\n";


for(counter = 1; counter <= Population_Size; counter++){
        for(g = 1; g <= NO_Station; g++){
                delete [] P_Chromosome[counter][g];
                delete [] D_Chromosome[counter][g];
        }
        delete [] P_Chromosome[counter];
        delete [] D_Chromosome[counter];
}
delete [] P_Chromosome; delete [] D_Chromosome;


for(counter = 1; counter <= NO_Offspring; counter++){
        for(g = 1; g <= NO_Zone; g++){
                delete [] New_P_Chromosome[counter][g];
                delete [] New_D_Chromosome[counter][g];
        }
        delete [] New_P_Chromosome[counter];
        delete [] New_D_Chromosome[counter];
}
delete [] New_P_Chromosome;
delete [] New_D_Chromosome;
```

```
	delete [] Chromosome_Fitness;

	for(g = 1; g <= NO_Zone; g++){
		delete [] P_Offspring[g];
		delete [] D_Offspring[g];
	}
	delete [] P_Offspring;
	delete [] D_Offspring;

	delete [] Candid_Fitness;
	delete [] Candid_List;

	return THE_BEST_Fitness;
}
double Fitness(int **Pickup , int **Drop){

	register int i, j;
	double FitValue = 0;

	for(i = 1; i <= NO_Station; i++){
		Lambda[i] = 0;
		Mu[i] = 0;
	}
```

```
	for(i = 1; i <= NO_Zone; i++){
		for(j = 1; j < i; j++){
			Lambda[Pickup[i][j]] = Lambda[Pickup[i][j]] + Demand[i][j];
			Mu[Drop[i][j]] = Mu[Drop[i][j]] + Demand[i][j];
		}
		for(j = i + 1; j <= NO_Zone; j++){
			Lambda[Pickup[i][j]] = Lambda[Pickup[i][j]] + Demand[i][j];
			Mu[Drop[i][j]] = Mu[Drop[i][j]] + Demand[i][j];
		}
	}
	for(i = 1; i <= NO_Station; i++){
		Lambda[i] = Lambda[i]/NO_Day;
		Mu[i] = Mu[i]/NO_Day;
	}

	for(i = 1; i <= NO_Station; i++)
		Open_Station[i] = 0;

	Sum_Initial_Bike = 0;
	Sum_Open_Station = 0;
	for(i = 1; i <= NO_Zone; i++) {
		for(j = 1; j < i; j++) {
			Sum_Initial_Bike = Sum_Initial_Bike + Demand[i][j] *
SS_Dis[Pickup[i][j]][Drop[i][j]];
			Open_Station[Pickup[i][j]] = 1;
			Open_Station[Drop[i][j]] = 1;
```

```
            }
            for(j = i + 1; j <= NO_Zone; j++) {
                Sum_Initial_Bike = Sum_Initial_Bike + Demand[i][j] *
SS_Dis[Pickup[i][j]][Drop[i][j]];
                Open_Station[Pickup[i][j]] = 1;
                Open_Station[Drop[i][j]] = 1;
            }
        }
        for(i = 1; i <= NO_Station; i++)
            if(Lambda[i] == 0 && Mu[i] == 0) Open_Station[i] = 0;

        for(i = 1; i <= NO_Station; i++)
            Sum_Open_Station = Sum_Open_Station + Open_Station[i];
        Sum_Initial_Bike = Sum_Initial_Bike/(NO_Day * NO_Hour * Speed);

        Cap[0] = Max(Min_Cap, (int) ceil(Sum_Initial_Bike/Sum_Open_Station));

        for(i = 1; i <= NO_Station; i++){
            if (Open_Station[i] && (Lambda[i] - Mu[i]) > 0){
                Init_Bicycle[i] = (int) ceil(Lambda[i] - Mu[i]);
                Cap[i] = 2 * Init_Bicycle[i] - 1;
                if (Cap[i] < Cap[0]){
                    Cap[i] = Cap[0];
                    Init_Bicycle[i] = (int) ceil((double) (Cap[i] + 1)/2);
                }
            }
```

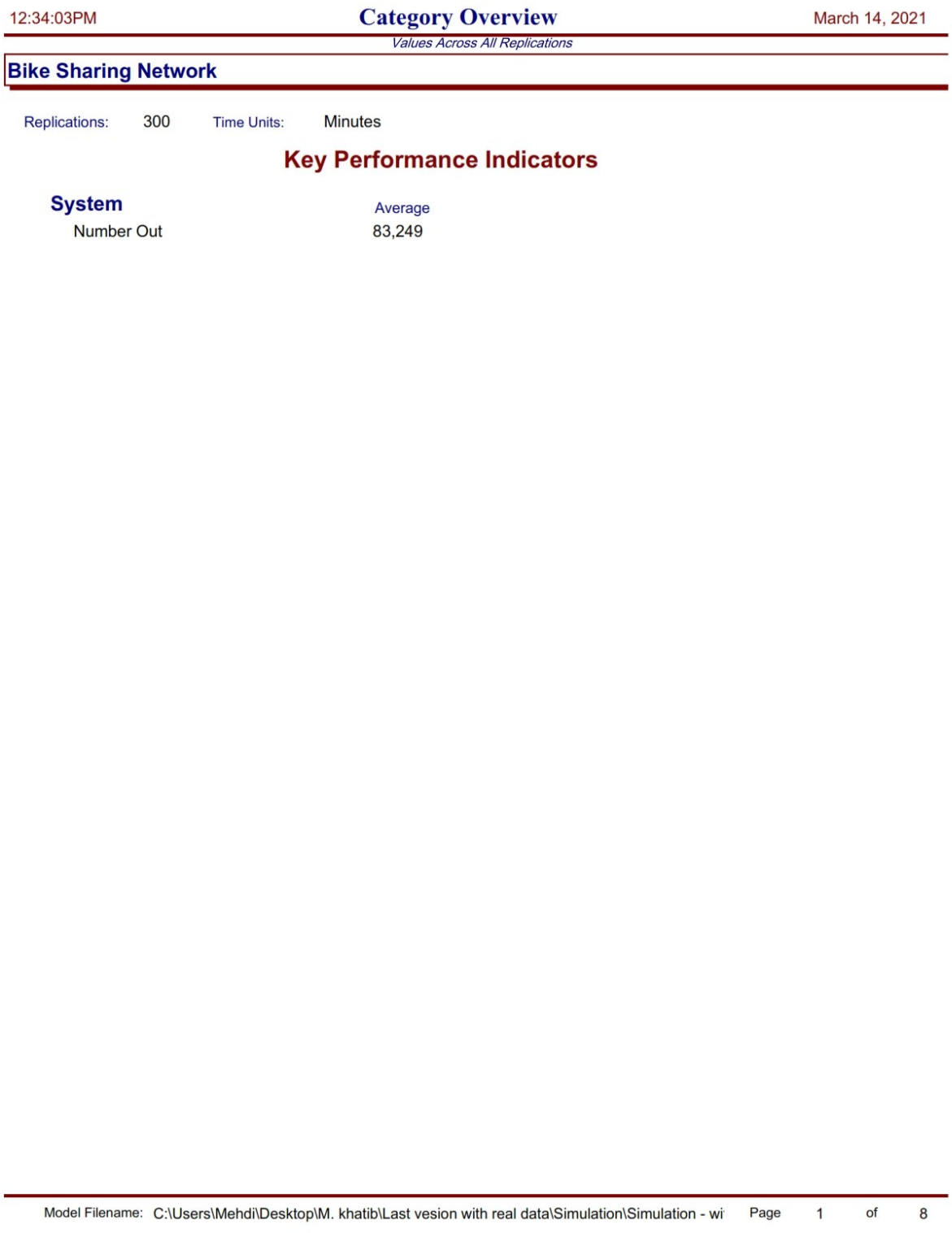
```
            else if (Open_Station[i] && (Lambda[i] - Mu[i]) <= 0){
                Init_Bicycle[i] = (int) ceil(Mu[i] - Lambda[i] + 1);
                Cap[i] = 2 * Init_Bicycle[i] - 1;
                if (Cap[i] < Cap[0]){
                    Cap[i] = Cap[0];
                    Init_Bicycle[i] = (int) ceil((double) (Cap[i] + 1)/2);
                }
            }
            else{
                Cap[i] = 0;
                Init_Bicycle[i] = 0;
            }
        }

        for(i = 1; i <= NO_Zone; i++){
            for(j = 1; j < i; j++)
                FitValue = FitValue + UWC * Demand[i][j] * (ZS_Dis[i][Pickup[i][j]] +
ZS_Dis[j][Drop[i][j]]);
            for(j = i + 1; j <= NO_Zone; j++)
                FitValue = FitValue + UWC * Demand[i][j] * (ZS_Dis[i][Pickup[i][j]] +
ZS_Dis[j][Drop[i][j]]);
        }



        for(i = 1; i <= NO_Station; i++)
            if(Cap[i] <= Max_Cap)
```

```
                FitValue = FitValue + FC[Cap[i]] + FCB * Init_Bicycle[i];
        else
                FitValue = FitValue + (Cap[i] - Max_Cap) * FC[Max_Cap] + FCB *
Init_Bicycle[i];




    for(i = 1; i <= NO_Station; i++){
        if(Open_Station[i]){
            if(Mu[i]/Lambda[i] < Min_p)
                FitValue = FitValue + BIG * (Min_p - Mu[i]/Lambda[i]) * (Min_p
- Mu[i]/Lambda[i]);
            else if(Mu[i]/Lambda[i] > Max_r)
                FitValue = FitValue + BIG * (Max_r - Mu[i]/Lambda[i]) * (Max_r
- Mu[i]/Lambda[i]);
        }
    }


    for(i = 1; i <= NO_Zone; i++){
        for(j = 1; j < i; j++)
            if(Pickup[i][j] == Drop[i][j])
                FitValue = FitValue + BIG;
        for(j = i + 1; j <= NO_Zone; j++)
            if(Pickup[i][j] == Drop[i][j])
                FitValue = FitValue + BIG;
    }
```

```
	return FitValue;
}

void FinalFree(void) {
	register int i;

	for(i = 1; i <= NO_Zone; i++){
		delete Demand[i];
		delete ZS_Dis[i];
		delete Pr_Dis[i];
	}
	delete [] Demand;
	delete [] ZS_Dis;
	delete [] Pr_Dis;

	for(i = 1; i <= NO_Station; i++)
		delete SS_Dis[i];
	delete [] SS_Dis;

	delete [] FC;
	delete [] Init_Bicycle;
	delete [] Cap;
	delete [] Near1;
	delete [] Near2;
```

```
        delete [] Near3;
        delete [] Near4;
        delete [] Near5;
        delete [] Open_Station;
}
```

The result of Arena for a bike sharing network with 10 demand zones and 3 possible stations:



12:34:03PM **Category Overview** March 14, 2021

Values Across All Replications

Bike Sharing Network

Replications: 300 Time Units: Minutes

Key Performance Indicators

System	Average
Number Out	83,249

Model Filename: C:\Users\Mehdi\Desktop\M. khatib\Last vesion with real data\Simulation\Simulation - wi Page 1 of 8

Bike Sharing Network

Replications: 300 Time Units: Minutes

Entity

Time

VA Time	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Arrival passenger to pick up at station 1	0.01463299	< 0.00	0.01446017	0.01485085	0.00	0.02000000
Arrival passenger to pick up at station 2	0.01469814	< 0.00	0.01445810	0.01503794	0.00	0.02000000
Arrival passenger to pick up at station 3	0.01425365	< 0.00	0.01381903	0.01477796	0.00	0.02000000

NVA Time	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Arrival passenger to pick up at station 1	0.00	< 0.00	0.00	0.00	0.00	0.00
Arrival passenger to pick up at station 2	0.00	< 0.00	0.00	0.00	0.00	0.00
Arrival passenger to pick up at station 3	0.00	< 0.00	0.00	0.00	0.00	0.00

Wait Time	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Arrival passenger to pick up at station 1	0.02770623	< 0.00	0.02410865	0.03352045	0.00	11.9410
Arrival passenger to pick up at station 2	0.02950874	< 0.00	0.02370274	0.03501347	0.00	14.1631
Arrival passenger to pick up at station 3	0.05616262	< 0.00	0.04497790	0.06876548	0.00	23.7134

Transfer Time	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Arrival passenger to pick up at station 1	4.8597	< 0.00	4.8013	4.9317	0.00	12.2195
Arrival passenger to pick up at station 2	3.7882	< 0.00	3.7252	3.8859	0.00	6.4203
Arrival passenger to pick up at station 3	3.4582	< 0.00	3.3443	3.5973	0.00	7.1410

Other Time	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Arrival passenger to pick up at station 1	0.00	< 0.00	0.00	0.00	0.00	0.00
Arrival passenger to pick up at station 2	0.00	< 0.00	0.00	0.00	0.00	0.00
Arrival passenger to pick up at station 3	0.00	< 0.00	0.00	0.00	0.00	0.00

Values Across All Replications

Bike Sharing Network

Replications: 300 Time Units: Minutes

Entity

Time

Total Time	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Arrival passenger to pick up at station 1	4.9020	< 0.00	4.8444	4.9739	0.00	19.1020
Arrival passenger to pick up at station 2	3.8325	< 0.00	3.7683	3.9274	0.00	20.6034
Arrival passenger to pick up at station 3	3.5286	< 0.00	3.4188	3.6668	0.00	30.8744

Other

Number In	Average	Half Width	Minimum Average	Maximum Average
Arrival passenger to pick up at station 1	34601.03	21.16	33967.00	35057.00
Arrival passenger to pick up at station 2	31102.03	19.90	30579.00	31663.00
Arrival passenger to pick up at station 3	17561.82	16.51	17139.00	18042.00

Number Out	Average	Half Width	Minimum Average	Maximum Average
Arrival passenger to pick up at station 1	34593.43	21.18	33959.00	35050.00
Arrival passenger to pick up at station 2	31096.51	19.90	30576.00	31657.00
Arrival passenger to pick up at station 3	17558.96	16.52	17137.00	18037.00

Bike Sharing Network

Replications: 300 Time Units: Minutes

Entity

Other

WIP	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
Arrival passenger to pick up at station 1	7.8519	< 0.00	7.7822	7.9013	0.00	21.0000
Arrival passenger to pick up at station 2	5.5180	< 0.00	5.4079	5.5805	0.00	19.0000
Arrival passenger to pick up at station 3	2.8688	< 0.00	2.7978	2.9368	0.00	14.0000

Bike Sharing Network

Replications: 300 Time Units: Minutes

Queue

Time

Waiting Time	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
1.wait for Dock_2 availability.Queue	0.7536	< 0.04	0.00	2.1104	0.00	7.0635
1.wait for Dock_3 availability.Queue	1.3230	< 0.02	0.8672	2.0761	0.00007625	11.9410
2.wait for Dock_3 availability.Queue	1.3126	< 0.03	0.7136	2.1044	0.00012759	10.8459
3.wait for Dock_2 availability.Queue	0.6197	< 0.06	0.00	4.6664	0.00	4.6664
Wait for Bike_1 availibility.Queue	0.8581	< 0.00	0.7804	0.9514	0.00000049	9.6542
Wait for Bike_2 availibility.Queue	0.9499	< 0.00	0.8330	1.1006	0.00000796	14.1631
Wait for Bike_3 availibility.Queue	1.7552	< 0.01	1.4921	2.0195	0.00001421	23.7134

Other

Number Waiting	Average	Half Width	Minimum Average	Maximum Average	Minimum Value	Maximum Value
1.wait for Dock_2 availability.Queue	0.00032429	< 0.00	0.00	0.00146555	0.00	3.0000
1.wait for Dock_3 availability.Queue	0.00315975	< 0.00	0.00150436	0.00670113	0.00	4.0000
2.wait for Dock_1 availability.Queue	0.00	< 0.00	0.00	0.00	0.00	0.00
2.wait for Dock_3 availability.Queue	0.00242657	< 0.00	0.00082588	0.00493959	0.00	3.0000
3.wait for Dock_1 availability.Queue	0.00	< 0.00	0.00	0.00	0.00	0.00
3.wait for Dock_2 availability.Queue	0.00008982	< 0.00	0.00	0.00047754	0.00	2.0000
Wait for Bike_1 availibility.Queue	0.04090655	< 0.00	0.03530502	0.04867326	0.00	6.0000
Wait for Bike_2 availibility.Queue	0.04006652	< 0.00	0.03258889	0.04855910	0.00	6.0000
Wait for Bike_3 availibility.Queue	0.04558248	< 0.00	0.03614256	0.05652340	0.00	7.0000

Bike Sharing Network

Replications: 300 Time Units: Minutes

User Specified

Counter

Bike Sharing Network

Replications: 300 Time Units: Minutes

User Specified

Counter

Count	Average	Half Width	Minimum Average	Maximum Average
1. Dock_2 shortage	36.9633	< 1.02	19.0000	66.0000
1. Dock_3 shortage	204.69	< 2.35	151.00	291.00
1. Drop off Passengers at station 2	17862.30	< 11.01	17529.00	18116.00
1. Drop off Passengers at station 3	7447.41	< 8.77	7139.00	7706.00
1. Successful Drop off Bike_2	18030.00	< 10.21	17737.00	18281.00
1. Successful Drop off Bike_3	7279.68	< 8.32	7044.00	7484.00
2. Dock_1 shortage	0.00	< 0.00	0.00	0.00
2. Dock_3 shortage	157.85	< 2.03	104.00	208.00
2. Drop off Passengers at station 1	17471.46	< 10.30	17095.00	17695.00
2. Drop off Passengers at station 3	5381.06	< 8.26	5206.00	5578.00
2. Successful Drop off Bike_1	17471.46	< 10.30	17095.00	17695.00
2. Successful Drop off Bike_3	5234.32	< 7.72	5057.00	5400.00
3. Dock_1 shortage	0.00	< 0.00	0.00	0.00
3. Dock_2 shortage	11.1100	< 0.48	2.0000	25.0000
3. Drop off Passengers at station 1	7836.00	< 8.45	7612.00	8050.00
3. Drop off Passengers at station 2	4677.32	< 7.22	4481.00	4863.00
3. Successful Drop off Bike_1	7836.00	< 8.45	7612.00	8050.00
3. Successful Drop off Bike_2	4824.04	< 7.68	4587.00	5036.00
Bike_1 shortage	9283.74	< 19.50	8743.00	9687.00
Bike_2 shortage	8244.01	< 18.99	7586.00	8772.00
Bike_3 shortage	5045.65	< 16.79	4492.00	5574.00
Successful Pick up Bike_1	25317.22	< 8.06	25083.00	25496.00
Successful Pick up Bike_2	22857.97	< 10.40	22551.00	23079.00
Successful Pick up Bike_3	12516.14	< 9.97	12240.00	12757.00
totall pick up passengers at station 1	34601.03	< 21.16	33967.00	35057.00
totall pick up passengers at station 2	31102.03	< 19.90	30579.00	31663.00
totall pick up passengers at station 3	17561.82	< 16.51	17139.00	18042.00

Values Across All Replications

Bike Sharing Network

Replications: 300 Time Units: Minutes

User Specified

Counter