2020

Multi-objective Optimization of a Ridesharing System Performance

Mohammad Nasr Azadani
West Virginia University, monasrazadani@mix.wvu.edu

Recommended Citation

Nasr Azadani, Mohammad, "Multi-objective Optimization of a Ridesharing System Performance" (2020). *Graduate Theses, Dissertations, and Problem Reports*. 7898.
https://researchrepository.wvu.edu/etd/7898

Multi-objective Optimization of a Ridesharing System Performance

Mohammad Nasr Azadani

Thesis submitted to the

Benjamin M. Statler College of Engineering and Mineral Resources

at West Virginia University

In partial fulfillment of the requirements for the degree of

Master of Science in

Civil Engineering

Department of Civil and Environmental Engineering

Morgantown, West Virginia

2020

Keywords:

Ridesharing, Multi-objective, Optimization, Travel Time, Trip Time, Vehicle Traveled Miles, Matching

Abstract

Multi-objective Optimization of a Ridesharing System Performance

Mohammad Nasr Azadani

Ridesharing is a shared vehicle service with the potential to meet the growing travel demand due to population increase, economic growth, and shortage in transportation infrastructure capacity. Compared to the current system of predominantly using personal vehicles, ridesharing services reduce the number of vehicles while providing mobility services to the same number of people with no additional investment in the transportation infrastructure. One of the big challenges in implementing ridesharing services is matching drivers and riders. Conflicts between matching-objectives to comply with the interests of diverse stakeholders influence the efficiency of ridesharing in a transportation system. This study investigates the conflicts between two ridesharing matching-objectives minimization of systemwide Trip Time (TT) and minimization of systemwide Vehicle Miles Traveled (VMT) by adopting a multi-objective optimization technique. The optimization results indicate that it is possible to have an acceptable reduction in TT and VMT by optimizing the conflicts between conflicting objectives in a ridesharing system. Tradeoff analysis indicates the benefits of a multi-objective optimization model in a ridesharing system by optimizing ridesharing system performance considering multiple conflicting matching-objectives.

Dedication:

My family members, who always encourage me to pursue higher education,

My parents, who I owe them everything I have in my life.

And, my sisters and brother for their unconditional supports.

ACKNOWLEDGMENTS

I would like to express my deep and sincere gratitude and appreciation for my advisor and committee chair, Dr. David R. Martinelli for all of his guidance, patience, encouragement, and great support, during my Master's studies. I am grateful to my co-advisor, Dr. Kakan Dey, for his patience, motivation, continuous support, and effective suggestions to do this research. I would like to thank my committee member, Dr. Dimitra Pyrialakou, for serving on my committee and guidance.

And finally, I want to thank all my friends, especially Md. Tawhid Rahman for their help and support.

Table of Contents

CHAPTER 1: INTRODUCTION

Roadway users have different travel mode choices, especially in large cities. Each travel mode has its advantages and disadvantages. If travelers use their personal vehicles, they could reach their destinations at the earliest time, but their travel cost will be more, as they need to pay for gas, parking, and vehicle maintenance, etc. If roadway users choose public transportation, their cost will be less, but they would spend more time to complete the trip. Transportation mode choices of roadway users have different effects on the transportation systems, such as level of service, system-wide travel time, and traffic congestion. By reducing the number of vehicles on roads, it is possible to minimize the negative effects on transportation systems. Traffic congestion has negative impacts on the economy, environment, and quality of life (Mallus et al., 2017). Registered vehicles in the United States increased by more than 23 million from 2010 to 2018 (Bureau of Transportation Statistics, 2019). From 2005 to 2014, the annual delay per commuter increased 1 hour (from 41 hours to 42 hours), whereas the total annual delay increased from 6.3 billion hours to 6.9 billion hours (Transportation Statistics, Annual Report, USDOT, 2017). Traffic congestion increases vehicle idle time, thereby increasing fuel consumption and environmental pollution. Total fuel wasted increased from 2.7 billion gallons to 3.1 billion gallons due to the increase in traffic volume from 2005 to 2014 (Transportation Statistics, Annual Report, USDOT, 2017). Traffic congestion problems and concerns about the environment influence people to rethink the use of personal vehicles (Agatz et al., 2012). Many people prefer to use personal vehicles to commute to work. In 2017, 76.4% of workers in the United States used their personal vehicles to commute (Wagner, 2019). In 2017, each driver in the United States traveled 25.9 miles on average per day. Due to spending 97 hours in traffic congestion, each driver lost $1,348 on average in 2018 as the value of their lost hours (Statista Research Department, 2019). The total cost of lost hours due to congestion was $87 billion in 2018 (Inrix, Inc.). Considering these diverse negative aspects of driving personal vehicles, daily driving costs are significant for each driver as well as for the U.S transportation system and the environment.

Developing, implementing, and improving public transportation systems has been one of the reliable solutions to reduce the impacts of traffic congestion. However, public transportation might not be available in all areas because public transportation services often operate on a specific schedule along fixed routes, and are not suitable for all people as their primary transportation mode (Mallus et al., 2017). One of the relatively new option transportation users have is ridesharing. Ridesharing reduces the number of vehicles on roads and could save travel time compared to public transportation. In ridesharing services, travelers who are on the same routes (i.e., the passengers' origin and destination are close to the drivers' travel path) and have similar time schedules, can share their rides. Ridesharing could reduce the number of vehicles on the road to one third (Conner-Simons, 2017). By using fewer numbers of vehicles on the roads, ridesharing helps to reduce congestion, emission, and fuel consumption (Cap et al., 2018). If transportation users share rides five times a week, 68.2 million vehicle-kilometers could be saved per year in Netherland. Besides, 12,674 tons of CO_2 emissions could be prevented per year (Caulfield, 2009).

Travelers who carpool can save money on tolls. For example, carpoolers can save \$6 per vehicle on tolls when they travel through the George Washington Bridge in New York (Bost, 2012). Although ridesharing has several advantages, some service parameters like increased trip time (TT) along with other disadvantages (such as privacy concerns and safety), make ridesharing less preferable to riders. Decision making between ridesharing and other modes of transportation can be complex for travelers due to conflicts between ridesharing matching-objectives. For example, riders may have to incur longer travel time in favor of lower travel costs and drivers may prefer to maximize profit by sharing a ride with more people with less driving distance. These conflicts might affect ridesharing adversely. From 2010 to 2017, the percentage of those who drive alone in the US increased from 76% to 76.4%, while the percentage of sharing rides has decreased from 10.4% to 8.9% at the same time (Freemark, 2019).

1.1 Different types of ridesharing:

To improve the ridesharing service performance and ridership, consideration of real-world consequences of ridesharing policies and strategies is critical. In the real world, the primary ridesharing vehicle's origin and destination, passengers' origin and destination, and total travel time are unknown and could vary significantly depending on many factors such as time of the day, congestion level, ridesharing demand. Two types of ridesharing are discussed below.

1) Static Ridesharing Service: In this type of ridesharing, the drivers and riders have an agreement to share their daily rides. For example, from home to work and work to home. Most of the

ridesharing travel parameters are constant in this type of ridesharing (e.g., vehicle miles traveled, travel time, travel cost, pick up time, and drop off time). Carpooling can be considered in this category.

2) Dynamic Ridesharing Service: In this type of ridesharing, the shortest routes among riders in vehicles are different. In dynamic ridesharing, a request for the service is executed in real-time. All travel parameters such as VMT, TT, and TC vary between trip requests. For this type of service, researchers are working to find the best answers to the following questions; How can we match drivers and riders in this system? How do we reduce the cost of this system? How can we increase the benefits for both drivers and riders? And, how do we optimize different matching-objectives to reach an optimum service? There is a third type of ridesharing system that is a combination of both previous types.

1.2 Multi-objective optimization in a ridesharing system

Many studies in the past few years investigated the optimization of ridesharing systems. These studies examine matching drivers and riders in different ways such as minimization of vehicle miles travel (VMT), minimization of travel time (TT), minimization of travel cost (TC), or maximization of privacy with limited consideration of the conflicts between the matching-objectives. Conflicts between ride-sharing objectives could be one of the reasons for peoples' decreasing interest in ridesharing. By optimizing the conflicts between objectives, the number of people who will be interested in this transportation mode can be increased. There are different objectives in ridesharing that could conflict with each other, such as VMT and TT, TT and TC, or VMT and TC. For example, if a rider is willing to share a ride with another person, the TC will be less for him, but the passenger's TT could increase as the driver needs to pick-up and drop-off other passengers, and pick-up and drop-off locations could deviate from the original route. Thereby, the optimization of matching-objectives is a challenge for a ridesharing system. To reduce the research gap, this research will evaluate the matching-objectives (minimization of VMT and minimization of TT) and understand their effect on each other in a hypothetical ridesharing scenario. This research attempts to implement an algorithm to improve the method of matching drivers and riders to increase the chance of choosing ridesharing as a transportation mode by more transportation users.

1.3 Problem statement and objectives

Ridesharing, as a transportation method, could be beneficial for transportation systems, especially in large cities, if higher market penetration of ridesharing can be reached. It could save riders time and cost and could be profitable for drivers. Transportation system-level advantages include reducing congestion, emissions, and air pollution. If a ridesharing system is not sufficiently attractive to transportation users and drivers, ridesharing may not achieve full potential. One of the problems that can affect a ridesharing system is the conflict between different ridesharing objectives. This conflict can reduce the performance of a ridesharing system. Two conflicting objectives in a ridesharing system are systemwide trip time (includes travel time, waiting time, and detour time) and systemwide vehicle miles traveled (VMT). An increase in a systemwide travel time might occur due to decreasing the systemwide VMT.

The goal of this study is to research if optimizing the conflict between the objectives in a ridesharing system by applying a multi-objective optimization can provide a desirable performance of the ridesharing system. A driver and rider matching model is proposed to minimize the conflicts between two conflicting objectives in a ridesharing system (systemwide trip time and systemwide vehicle miles traveled). The objectives of this research are:

1- Develop a driver-rider matching model considering multiple stakeholders' interests;
2- Evaluate the performance of ridesharing matching model using Pareto-optimal graphs; and
3- Investigate the tradeoffs of Trip Time as a matching objective with respect to vehicle miles traveled as the other objective

Systemwide trip time and systemwide vehicle miles traveled are selected as two conflicting objectives as increasing trip time has a negative impact on choosing ridesharing as a primary transportation option and trip time increment has a negative impact of systemwide VMT reduction.

CHAPTER 2: LITERATURE REVIEW

Although ridesharing has the potential to improve transportation systems performance, reduce traffic congestion, save travelers' time, and reduce emissions, successful deployment depends on the adoption of these services. Past studies on matching drivers and riders in a ridesharing system are reviewed in this chapter to develop a better understanding of the matching problem and justify the research gap.

The literature is categorized into three sections: Matching Drivers and Riders, Matching objective Vehicle Miles Traveled, and Matching objective Trip Time. These are important objectives in a ridesharing system to increase the adaption rate of a ridesharing system. These categories are selected to assess the impact of each objective on ridesharing systems. The impact of single-objective optimization in these studies are reviewed to understand the importance of each objective.

2.1 Matching drivers and riders:

Matching drivers and riders is the basic function of a ridesharing system. By assigning more passengers to each vehicle in a ridesharing system, the system could reduce the number of vehicles to serve more passengers in the entire network. Researchers explored the ways to assign a rider to a driver so that a certain matching objective of ridesharing stakeholders was optimized. Cici et al. (2015) proposed an on-line ridesharing system to match possible riders and drivers. The purpose of their study was to minimize the number of vehicles by providing rides to the maximum number of passengers using one vehicle. The matching ratio (on-line system) of this study was 78% compared to an off-line system whose matching-ratio was 80%. Goel et al. (2016) conducted a study to match drivers and riders based on their trip cost and trip preferences, and named the study "privacy-preserving dynamic ride-sharing system." The results showed that if drivers only had a small detour in their normal trips, ridesharing could save vehicle kilometer 12% on average (from 9% to 21%). For 11.6 million trips/day in Melbourne, Australia, with an average of 10.2 km trip length, their matching method could have a savings of 14.2 million kilometers per weekday. In another work on matching drivers and riders, Kleiner et al. (2011) presented an auction-based algorithm for dynamic ridesharing. In their algorithm, riders can send their request, including their location and time constraints, and then the system finds the drivers who are matched with the rider. Passengers in this system will be visible to drivers based on their bids, and drivers select the riders based on their preferences. Their system allows users more options to choose their ridesharing partners. Kleiner et al. (2011) concluded that if passengers are willing to pay more than the base cost, the chance of matching will be increased. Santos et al. (2013) worked on a framework for ridesharing, named "ridesharing with time window problem." Constraints considered in their framework are time windows to execute each request, vehicle's capacity, and the cost of a ride. Their study showed an 18.58 % (on average) reduction in the passengers' trip cost in the ridesharing system.

2.2 Matching objective- Vehicle Miles Travel (VMT)

In dynamic ridesharing, matching riders and drivers appropriately is important to reduce VMT. Systemwide VMT minimization is an important objective in a ridesharing system because it directly relates to systemwide congestion, fuel consumption, emissions, travel cost, and the maintenance of a transportation system. Many researchers worked on reducing VMT in a ridesharing system. Agatz et al. (2011) used two different algorithms: the GREEDY algorithm and the BIPART (bundle constraints binary integer programming approach) algorithm and compared the results from each algorithm to find the best approaches for ride-matching. They obtained better results in VMT saving (14 to 18%) with BIPART compared to the GREEDY algorithm. Rodier et al. (2016) examined the potential VMT reduction in the Bay Area, California, and found that VMT reduction was very small at a low level of participation in ridesharing. A VMT reduction of 9% at a moderate level and 11% to 19% in high-level participation were reported. The authors applied a VMT fee for the vehicles and concluded that a combination of Dynamic Ride-Sharing Services, Transit-Oriented Development, and VMT fee scenarios could have a better result in VMT reduction than Dynamic Ride-Sharing Service alone. Sun et al. (2018) conducted a study on the effects of applying taxes on reducing VMT in a ridesharing system. They studied a tax-pooling problem to minimize systemwide VMT, where ride requests were grouped by a system dispatcher. If, after grouping the requests, there were several people in the group assigned to a taxi, they would optimize the vehicle route by a sequence of pickups and drop-offs that are closer to the vehicle's location. The findings showed that VMT could be reduced by 18% if the buffer time is on average 25% of travel time.

Some studies focused on the taxi sharing system to find out the strategies to reduce VMT. Qian et al. (2017) did an experiment on TGR (Taxi Group Ride) to combine trips of the passengers with close departure time, origin, and destination. They compared the potential of three different algorithms to reduce VMT. The results showed that TGR could reduce more than 47% of total VMT, and ridesharing with two riders may have the most saving in VMT, and saving could be more on weekdays than weekends. Amey (2011) concluded from his research that VMT reduction of 9% to 27% could be achieved from ridesharing if 50% to 77% of people shared rides. Some studies focused on the taxi trip data in megacities to find the benefits of ridesharing in reducing VMT. Santi et al. (2014) studied share-ability networks to measure ridesharing benefits using New York taxi trips data. Their results showed that travel distance could be reduced by at least 40% in a ridesharing environment. However, their model considered ridesharing for a maximum of two riders only. Lokhandwala et al. (2018) studied dynamic ridesharing using an agent-based simulation model and determined the benefits of the ridesharing system in reducing VMT. Using New York City taxi data, they concluded that total VMT could be decreased by up to 55% by ridesharing. However, the potential benefits have been affected by limitations such as limiting the ridesharing to two riders only, which brought the ridesharing participation level down to 50-75%. In another case study, Cai et al. (2019) analyzed data collected from trajectories of shared taxis in Beijing to evaluate ridesharing benefits. They concluded that VMT could be reduced by 33% in ridesharing. In the study, they assumed that the tolerance level of waiting time for passengers was 10 minutes for this analysis.

2.3 Matching objective- Travel Time

Travel time is another important factor for matching drivers and passengers in a ridesharing system and choosing ridesharing as a preferred transportation option. Reducing travel time in a ridesharing system is important because people do not want to allow excessive travel time in a shared trip compared to driving their own vehicles. Moreover, the amount of emission is not only related to VMT but also related to travel time, and travel time is an important convenience factor for participating in ridesharing (Agatz et al., 2012). Alexander et al. (2015) studied the impacts of real-time ridesharing on congestion and used the data on locations and travel time collected from drivers' mobile phones. The results showed that ridesharing has a considerable impact on travel time if the rate of ridesharing adoption is from a moderate to a high level. The authors concluded that under 50% driver adoption rate, a 17.55% reduction in travel time, and a 37.3% reduction in Congested travel time (this is the time vehicles have to wait for in congestion) could be achieved.

Travel time may not decrease by ridesharing all the time. Sometimes travel time increases due to ridesharing, especially passengers' travel time. The reason for that is waiting time and detour time will be added to the normal travel time for the shortest travel time between each passenger origin and destination. Another reason is that there might be traffic congestion in the shortest route, and vehicles have to spend more time in congestion. Sharing a single ride with multiple passengers might decrease the systemwide VMT, and increase the individual travel times of passengers (Horn, 2002). Lin et al. (2012) investigated the optimization of a ridesharing model applying the genetic algorithm. They introduced a time window to simulate the passenger travel time and location of a real ridesharing system. Lin et al. (2012) measured the travel time before and after ridesharing, and their results showed that on average, the travel time after ridesharing is 59 minutes more than the travel time before ridesharing.

2.4 Ridesharing service quality, cost, and benefits

Service quality and cost are the two factors that can affect the demand for ridesharing services. Passengers look for high-quality service and want to pay the minimum cost for the service. Low quality for an expensive service reduces the likelihood of using the service later. Ridesharing has been modeled as a multi-objective problem by Cap et al. (2018) in their study "Multi-Objective Analysis of Ridesharing in Automated Mobility –on-Demand." They used two criteria to model ridesharing. They wanted to maximize service quality and minimize operating costs. It was concluded that the probability of demands for ridesharing would increase when the cost of operation in the system can be reduced with lower service quality. Lin et al. (2012) proposed a routing optimization model for ridesharing. Their goal was minimizing operating costs and maximizing customer satisfaction. The authors used a Simulated Annealing algorithm to optimize a ride-sharing system considering the benefits of both driver and rider in their model. The study results showed that ridesharing could reduce 19% VMT and 66% taxi demand. Cai et al. (2019) studied the benefits of ridesharing using the data collected from the taxi fleet in Beijing and concluded that if all taxis in Beijing participate in the ridesharing system, the fuel consumption can be reduced by 28.3 million gallons.

2.5 Goal Programming in multi-objective optimization

There are different methods and models to do a multi-objective optimization. Goal programming is one of the methods to solve the multi-objective optimization problem. Charnes et al. (1955) introduced the Goal Programming method to solve multi-objective optimization problems. Charnes and Cooper (1957) proposed an improved version of the goal programming method (Khademi Zareh et al., 2019). Goal programming is a method to convert multi-objective functions to a single objective function. In this method, a goal and a deviation will be assigned to each of the objectives. The deviation is any value higher or lower than the goal. The goal programming method minimizes the deviation of objective functions from their goal (Khademi Zareh et al., 2019). The goal is a target value for each objective function. Goal programming is an extended form of linear programming to solve multi-objective optimization where the objectives are often conflicting (Al Qahtani et al., 2019). As there are two objectives in this research, the goal programming method is used to solve the matching of drivers and riders in a ridesharing environment.

Although past studies considered the optimization of single objectives in a ridesharing system, no study optimized the relative conflicts between different objectives reflecting the interests of different stakeholders. In this study, the multi-objective optimization method is applied to develop a ridesharing system considering various stakeholders' interests. The findings of this study will be useful for researchers and decision-makers in developing a ridesharing service considering the conflicting interests of various stakeholders.

CHAPTER3: RESEARCH METHODS

Minimization of Trip Time (TT) and minimization of Vehicle Miles Travelled (VMT) are the two matching-objectives considered in this research to investigate their relative influence in ridesharing system performance. These two objectives are commonly used as matching-objectives by the ridesharing service platforms in past studies. Due to the conflicting nature of the matching-objectives, optimization of one matching-objective affect the performance of the ridesharing system in terms of other matching-objectives negatively.

3.1 Selection of matching-objectives

Ridesharing services decrease the number of vehicles needed for passenger transportation in a transportation system (Chan et al., 2012). The VMT in the system decreases due to the reduction of the number of vehicles in a transportation system. In addition to VMT reduction, minimizing systemwide VMT (matching-objective 1) as the matching-objective could reduce fuel consumption and associated emissions. However, when the number of vehicles is reduced in the system, the riders have to spend more time on their trip for the following reasons: (i) waiting time at pick up location to receive the ride, and (ii) additional travel time in the vehicle to pick up and drop off other passengers (i.e., detour time).

In ridesharing, trip time usually increases for both drivers and riders. Increased trip time is a drawback, as riders and drivers might have limited flexibility in accommodating additional travel time. High travel time could reduce the number of users of a ridesharing service. Minimizing systemwide trip time (matching-objective 2) could reduce trips with excessive deviation from the desired travel times of users. Optimization of any one of these two matching-objectives separately thereby influences ridesharing system performance in terms of the other matching-objective. Multi-objective optimization considering these two matching-objectives could provide a matching solution by balancing the related benefits and drawbacks.

3.2 Models and parameters

Drivers' origin-destination and passengers' origin-destination are the primary data needed for matching drivers and riders. Besides these data, the model needs transportation network-related data to find the optimum path to match drivers and riders. Table 1 represents the required data for matching drivers and passengers to develop ridesharing matching, and Table 2 summarizes the parameters used in the model formulation.

Table 1: Data required from drivers and passengers

Data from drivers	Data from passengers
Origin (O)	Origin
Destination (D)	Destination
Time of beginning the trip	Time of trip
Maximum acceptable trip distance	Maximum acceptable waiting time
Maximum acceptable trip time	Maximum acceptable detour time
Maximum acceptable number of passengers	Maximum acceptable trip distance

Table 2: Model parameters

Parameters	**Symbol**	**Parameters**	**Symbol**
Passenger request ID	P_i	*Origin of vehicle V_j*	V_j^o
Vehicle ID	V_j	*Destination of vehicle V_j*	V_j^d
Passenger "i" origin	P_i^o	*Waiting time of passenger request, P_i for vehicle V_j at pick-up point*	$W_{P_i^o}^{V_j}$
Passenger "i" destination	P_i^d	*Shortest travel time between the O-D of a passenger request P_i*	$S_{P_i^o, P_i^d}$
Total vehicles number available in the network	T	*Detour time of passenger request P_i in vehicle V_j*	$T_{P_i^o, P_i^d}^{V_j}$
Total number of passengers	N	*Shortest travel time between the O-D of the vehicle V_j*	$S_{V_j^o, V_j^d}$
Congestion factor	$T_{R_n^l}$	*Total travel time of passenger request P_i who did not get the ride*	$T_{P_i^o, P_i^d}$
The number of passengers in the vehicle	$N_{P_{ij}}$	*Detour time of vehicle V_j for serving passenger request*	$D_{V_j^o, V_j^d}$
A portion of Drivers' origin to destination distance when there is no passenger	$\propto$	*Maximum late time allowed for the vehicle to pick up the passenger*	MAW_{P_i, V_j}

A portion of Drivers' origin to destination Travel Time when there is no passenger	β	*Number of passengers allowed to be in the vehicle*	NW_{P_i,V_j}
Detour distance of passenger request P_i in vehicle V_j	$D^{V_j}_{P_i^o,P_i^d}$	*Shortest travel distance between origin and destination of vehicle, V_j*	$SD_{V_j^o,V_j^d}$
Detour distance of the vehicle, V_j for serving passenger requests	$DD_{V_j^o,V_j^d}$	*Travel distance of passenger request, P_i who did not get the ride*	$TD_{P_i^o,P_i^d}$
Drivers' original travel time	T_{t,v_j}	*Vehicle's original path distance*	MDW_{R_n,R_n^l}
Shortest travel distance between origin and destination of a passenger request P_i	$SD_{P_i^o,P_i^d}$	*A portion of Passengers' original travel distance*	μ

Decision variable:

$y^{V_j}_{P_i}$ = Represents if there is any passenger request, P_i is served by a vehicle V_j.

If $y^{V_j}_{P_i}$ = 0, passenger request, P_i is served by a vehicle V_j

If $y^{V_j}_{P_i}$ = 1, passenger request, P_i is not served by a vehicle V_j

3.3 Multi-objective optimization model formulation

The assumptions adopted in the model formulation are listed below.

i. There is no congestion in the network.
ii. The passengers for each vehicle (if there are any) are assigned to the vehicle before the vehicle begins the travel from its origin.
iii. The passengers who could not share their ride will use their own vehicle to complete the trip.
iv. Each passenger can only be assigned to one vehicle, if a passenger is assigned to a vehicle, the passenger will be removed from the list of the passengers for the other vehicles.

3.3.1 Objective 1: Minimization of the systemwide Trip Time:

Objective function:

$$Min\sum_{P_i \in N}[\sum_{y_{Pi}^{Vj}=1}(W_{P_i^o}^{V_j} + S_{P_i^o, P_i^d} + \mathrm{T}_{P_i^o, P_i^d}^{V_j}) + \sum_{y_{Pi}^{Vj}=0} T_{P_i^o, P_i^d}] + \sum_{V_j \in T_1}(S_{V_j^o, V_j^d} + D_{V_j^o, V_j^d}) +$$

$$\sum_{V_j \in T_2} S_{V_j^o, V_j^d} + \sum_{V_j^o}^{V_j^d} T_{R_n^l} * (N_{P_i j} + 1)$$

The first objective function includes five components. Component 1: $\sum_{y_{Pi}^{Vj}=1}(W_{P_i^o}^{V_j} + S_{P_i^o, P_i^d} + T_{P_i^o, P_i^d}^{V_j})$ indicates the total trip time for passengers, including waiting time, shortest travel time, and the detour time. Component 2: $\sum_{y_{Pi}^{Vj}=0} T_{P_i^o, P_i^d}$ shows the total travel time for the passengers who could not share their ride with any driver. Component 3: $\sum_{V_j \in T_1}(S_{V_j^o, V_j^d} + D_{V_j^o, V_j^d})$ represents the total drivers' trip time, including the shortest travel time and detour time. Component 4: $\sum_{V_j \in T_2} S_{V_j^o, V_j^d}$ states the total travel time for the vehicles who could not find a passenger to share their ride. Component 5: $\sum_{V_j^o}^{V_j^d} T_{R_n^l} * (N_{P_i j} + 1)$ presents the travel time due to congestion for both drivers and passengers, which is assumed to be zero in this research.

The objective function is subjected to the following constraints.

$$D_{V_j^o, V_j^d} \leq \beta\, T_{t, v_j} \quad \forall P_i \in N,\ V_j \in T \qquad (1)$$

$$T_{P_i^o, P_i^d}^{V_j} \leq S_{P_i^o, P_i^d} \qquad (2)$$

$$W_{P_i^o}^{V_j} \leq MAW_{P_i, V_j} \qquad (3)$$

$$N_{P_i, V_j} \leq NW_{P_i, V_j} \qquad (4)$$

The first constraint ensures that the driver's detour time to pick up and drop off passengers will not be more than a portion of the driver's original travel time (without passenger, β). Due to this constraint, the detour time of drivers will be limited to a portion of their original travel time to make sure that the assigned passengers are within an acceptable time range from the vehicle's original path. The second constraint assures that passenger's detour time will not be more than a portion of the passenger's original travel time. The third constraint ensures that passengers' waiting time will not be more than the maximum allowed waiting time for each passenger. Passengers in ridesharing can get their ride within an acceptable waiting time. The fourth constraint confirms that the maximum number of passengers in a vehicle can not be more than the seating capacity of a vehicle.

3.3.2 Objective 2: Minimization of the systemwide Vehicle Miles Travelled

Objective function:

$$Min \sum_{V_j \in T_1} (SD_{V_j^o, V_j^d} + DD_{V_j^o, V_j^d}) + \sum_{V_j \in T_2} SD_{V_j^o, V_j^d} + \sum_{y_{Pi}^{Vj}=0} TD_{P_i^o, P_i^d}$$

The second objective function has three components. The first component: $\sum_{V_j \in T_1} (SD_{V_j^o, V_j^d} + DD_{V_j^o, V_j^d})$ represents the total drivers' travel distance, including the shortest path from the drivers' origin to the drivers' destination and the detour distance required to pick up and drop off the passengers. The second component: $\sum_{V_j \in T_2} SD_{V_j^o, V_j^d}$ indicates the shortest travel distance of the vehicles who could not find a passenger to share their ride. The third component: $\sum_{y_{Pi}^{Vj}=0} TD_{P_i^o, P_i^d}$ shows the total travel distance of the passengers who could not able to manage the shared ride.

The objective function is subjected to the following constraints.

$$D_{V_j^o, V_j^d} \leq \propto S_{V_j^o, V_j^d} \quad \forall \ i \in N, V_j \in T \qquad (1)$$

$$D_{V_j^o, V_j^d} \leq \delta \ MDW_{R_n, R_n^l} \qquad (2)$$

$$D^{V_j}_{P_i^o, P_i^d} \leq \mu SD_{P_i^o, P_i^d} \qquad (3)$$

The first constraint ensures that detour distance for pickup and drop-off passengers will not be more than a certain percentage of the driver's original travel path (without passenger). This ensures drivers to find the passengers closest to their travel path. The second constraint guarantees that the driver's detour distance will not be more than a certain percentage of the vehicle's original path's distance (δ). This constraint makes sure that the drivers will not travel more than an acceptable distance to pick up and drop off passengers from their shortest original path/route. The third constraint confirms that passengers' detour distance cannot be more than a percentage of the passengers' original travel distance. This constraint gives an option to passengers to not travel more than a percentage of their original travel distance and should be served by their closest available drivers and share their ride with the closest passengers.

3.4 Application of Goal Programming

Goal programming is used to solve multi-objective optimization problems. Each objective function has a goal (i.e., target value) to be achieved, where deviations can be measured above and below the goal. When optimizing two objective functions, an optimal solution can be achieved by minimizing the deviations from the target value for each optimization objectives. The Weighted Goal Programming (WGP) was adopted to solve the bi-objective optimization problem developed in this research. In WGP, the weights of the undesirable deviations will be assigned based on their degree of importance to decision-makers and will be minimized in an Archimedean sum (Tamiz et al., 1997). The algebraic formulation for the WGP is:

$$f_1(x_i) + n_1 - p_1 < b_1 \quad \ldots\ldots\ldots\ldots\ldots\ldots.. First\ Objective$$

$$f_2(x_i) + n_2 - p_2 < b_2 \quad \ldots\ldots\ldots\ldots\ldots\ldots.. Second\ Objective$$

$$Z = u_1 n_1 + u_2 n_2 + v_1 p_1 + v_2 p_2 \ \ldots\ Objective\ function\ for\ the\ Goal\ Programming\ Model$$

$$Min\ Z = \sum_{i=1}^{k} u_1 n_1 + u_2 n_2 + v_1 p_1 + v_2 p_2$$

$$x \in H_c$$

In this formulation, $f_1\ (x)\ and\ f_2\ (x)$ are objective functions of variable X = $\{x_1, x_2, \ldots\ldots x_n\}$, $b_1\ and\ b_2\ are$ the goals of the objectives (target values). The positive deviation from the goals are displayed as $p_1\ and\ p_2$ and the negative deviations from the goal are denoted as $n_1\ and\ n_2$. The weights of the deviations in the objective function Z are represented as u_i and v_i. Finally, H_c represents a set of constraints for each objective.

3.6 Solution Algorithm

Efficiently matching drivers and riders is a major challenge for the operators of a ridesharing service. In this research, Breadth-First Search (BFS) algorithm is applied to find the shortest path (based on either distance or time) from origin to destination. Selected routes of drivers and riders will be based on the multi-objective optimization of two matching objectives. Multi-objective optimization optimized the conflicts between the two objectives to find the optimum paths. Depending on the importance of each matching-objective, there are different methods and models to do a multi-objective optimization in a ridesharing system. Charnes et al. (1955) suggested the Goal Programming method, which converts the two matching-objectives to one objective function. Goal programming attempts to minimize the deviations of objective functions to reach the optimum solution (Khademi Zareh et. al, 2019).

CHAPTER 4: DATA ANALYSIS AND SIMULATION RESULTS

4.1 Data analysis method

To assess the performance of multi-objective optimization in a ridesharing system, a ridesharing system is simulated on a hypothetical grid network. For this simulation, an undirected graph was created with 100 nodes (see Fig. 1). The graph has two different values for distance and time between every two nodes. The value of time depends on distance and speed. The speed of each edge is randomly selected between the minimum speed limit (30 mph) and the maximum speed limit (60 mph). The edges' weights are not based on the length of the edges but assigned randomly.

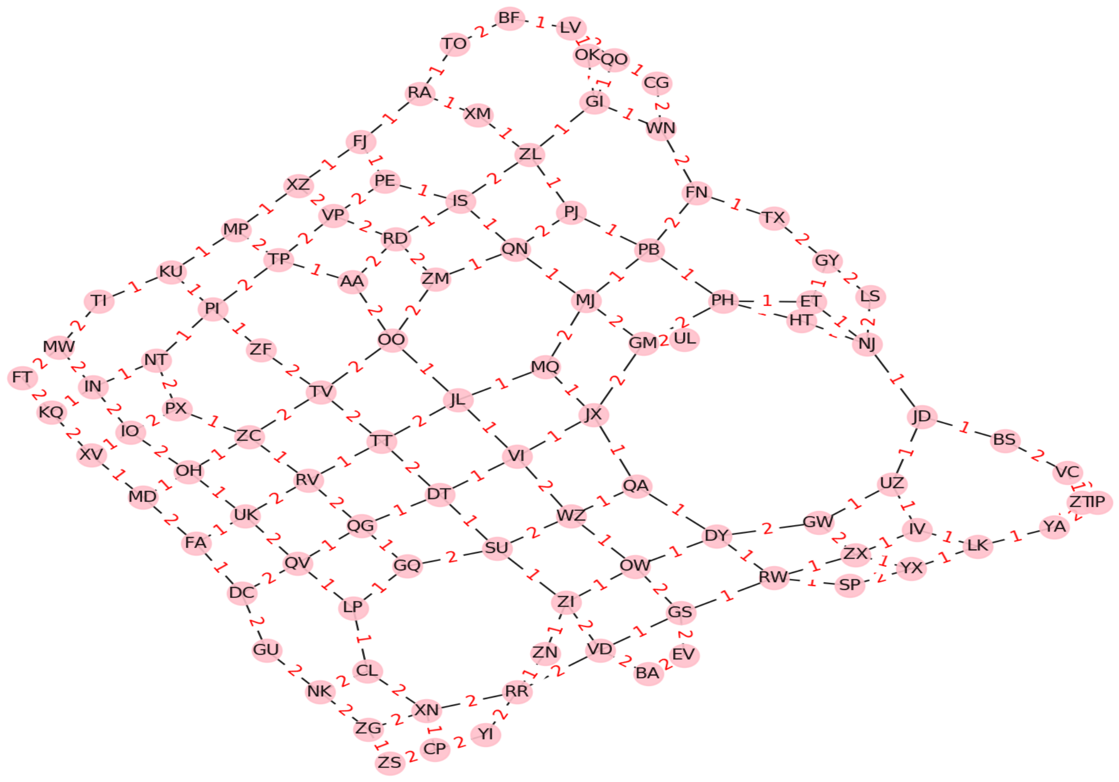


Fig. 1: Hypothetical random roadway network

The nodes represent intersections with the cross street that can be used as the pick-up or drop off locations for passengers. The number on each link indicates the distance (in miles) between the corresponding nodes.

Origin–Destination (OD) points for each driver and passenger were different for different combinations of drivers and passengers and are shown in Appendix A. The maximum capacity for each vehicle was three passengers (excluding the driver). If any of the passengers did not get a ride (i.e., no match with a driver), it was assumed that they would use their personal vehicles. The drivers and passengers OD data used in this research were stochastic data and generated in python. Factors in a ridesharing system (i.e., number of available vehicles, number of passengers, ODs of drivers and passengers) vary in the real world. A stochastic simulation scenario reflects the real-world scenario. Data were collected before and after ridesharing, where the data collected before ridesharing were considered as the base condition, and the data collected after ridesharing were compared with the base condition.

After creating the graph, we randomly select two of the nodes for each vehicle $G_j = (O_j, D_j)$, in which "O" represents the origin, "D" represents the destination, and "j" represents the vehicle's ID. Two random nodes also will be selected for each passenger $G_i = (O_i, D_i)$, in which "O" represents the origin, "D" represents the destination, and "i" represents the passenger's ID. In the first step, the system identified the shortest path for the OD of each vehicle and each passenger. In this step, we had two different parameters- distance and time- for each vehicle and each passenger.

In the next step, passengers were assigned to each vehicle based on the minimum distance path or minimum time path between the vehicles' origin to passengers' origin and the vehicles' destination to passengers' destination. For this, the system checked all the passengers existing in the network for each vehicle and found the closest passengers for each vehicle. Three different strategies were used to assign passengers to available vehicles. In the first strategy, passengers were assigned to the vehicles based on the minimum distance path. In this scenario, the system did not consider the minimum time path. The total trip distances in the system were minimum irrespective of the trip time. The second strategy was to assign passengers based on the minimum time path, where passengers were assigned to the vehicles such that the system experienced the minimum possible trip time. The total trip time following this strategy was less than the passenger allocation using the first strategy, but the total trip distance might be higher. The third strategy was based on optimizing the conflicts between the first and the second strategies. Here "Goal Programming" is used to optimize the conflicts and identify the optimum possible travel paths for the drivers and passengers. BFS algorithm was used to find the shortest paths between the origins and destinations of both drivers and passengers. The systemwide network was created and programmed in Python. From the simulation, travel time, travel distance, detour distance, and detour time data were collected for the vehicles, and travel time, waiting time, detour time, travel distance, and detour distance data were collected for the passengers.

4.2 The parameters used in the simulation scenario

Tables 4-1 and 4-2 summarized different ridesharing demand (i.e., number of riders) and supply (i.e., number of drivers) and network characteristics used to solve the bi-objective optimization problem developed in this research.

Table 4-1: The parameters have been used in this research

Parameters in the system	**Scenario #1**	**Scenario # 2**	**Scenario # 3**	**Scenario # 4**	**Scenario # 5**	**Scenario # 6**
Network size	100 nodes	100 nodes	100 nodes	100 nodes	100 nodes	100 nodes
Length of the link	Varies for successive nodes	Varies for successive nodes	Varies for successive nodes	Varies for successive nodes	Varies for successive nodes	Varies for successive nodes
Number of drivers (vehicles)	15	15	15	15	15	50
Number of passengers	20	30	40	50	60	40
Percentage of the maximum allowed extra ride distance for drivers	100 % increase in the vehicle's original travel distance	100 % increase in the vehicle's original travel distance	100 % increase in the vehicle's original travel distance	100 % increase in the vehicle's original travel distance	100 % increase in the vehicle's original travel distance	100 % increase in the vehicle's original travel distance
Percentage of the maximum allowed extra ride time for drivers	100 % increase in the vehicle's original travel time	100 % increase in the vehicle's original travel time	100 % increase in the vehicle's original travel time	100 % increase in the vehicle's original travel time	100 % increase in the vehicle's original travel time	100 % increase in the vehicle's original travel time
Percentage of the maximum allowed extra ride distance for passengers	50 % increase in the passenger's original	50 % increase in the passenger's original	50 % increase in the passenger's original	50 % increase in the passenger's original	50 % increase in the passenger's original	50 % increase in the passenger's original

	travel distance	travel distance	travel distance	travel distance	travel distance	travel distance
Percentage of the maximum allowed extra ride time for passengers	50 % increase in the passenger's original travel time	50 % increase in the passenger's original travel time	50 % increase in the passenger's original travel time	50 % increase in the passenger's original travel time	50 % increase in the passenger's original travel time	50 % increase in the passenger's original travel time
Percentage of the maximum allowed waiting time for passengers	20 % increase in the passenger's original travel time	20 % increase in the passenger's original travel time	20 % increase in the passenger's original travel time	20 % increase in the passenger's original travel time	20 % increase in the passenger's original travel time	20 % increase in the passenger's original travel time
Passengers capacity of a vehicle	3	3	3	3	3	3
Minimum speed limit (mph)	30	30	30	30	30	30
Maximum speed limit (mph)	60	60	60	60	60	60
Congestion delay	0	0	0	0	0	0

Table 4-2: The parameters have been used in this research

Parameters in the system	**Scenario#7**	**Scenario # 8**	**Scenario # 9**	**Scenario# 10**	**Scenario# 11**	**Scenario# 12**
Network size	100 nodes	100 nodes	100 nodes	100 nodes	100 nodes	100 nodes
Length of the link	Varies for successive nodes	Varies for successive nodes	Varies for successive nodes	Varies for successive nodes	Varies for successive nodes	Varies for successive nodes
Number of drivers (vehicles)	60	70	60	60	60	60
Number of passengers	40	40	60	80	100	120

Percentage of the maximum allowed extra ride distance for drivers	100 % increase in the vehicle's original travel distance	100 % increase in the vehicle's original travel distance	100 % increase in the vehicle's original travel distance	100 % increase in the vehicle's original travel distance	100 % increase in the vehicle's original travel distance	100 % increase in the vehicle's original travel distance
Percentage of the maximum allowed extra ride time for drivers	100 % increase in the vehicle's original travel time	100 % increase in the vehicle's original travel time	100 % increase in the vehicle's original travel time	100 % increase in the vehicle's original travel time	100 % increase in the vehicle's original travel time	100 % increase in the vehicle's original travel time
Percentage of the maximum allowed extra ride distance for passengers	50 % increase in the passenger's original travel distance	50 % increase in the passenger's original travel distance	50 % increase in the passenger's original travel distance	50 % increase in the passenger's original travel distance	50 % increase in the passenger's original travel distance	50 % increase in the passenger's original travel distance
Percentage of the maximum allowed extra ride time for passengers	50 % increase in the passenger's original travel time	50 % increase in the passenger's original travel time	50 % increase in the passenger's original travel time	50 % increase in the passenger's original travel time	50 % increase in the passenger's original travel time	50 % increase in the passenger's original travel time
Percentage of the maximum allowed waiting time for passengers	20 % increase in the passenger's original travel time	20 % increase in the passenger's original travel time	20 % increase in the passenger's original travel time	20 % increase in the passenger's original travel time	20 % increase in the passenger's original travel time	20 % increase in the passenger's original travel time
Passengers capacity of a vehicle	3	3	3	3	3	3
Minimum speed limit (mph)	30	30	30	30	30	30
Maximum speed limit (mph)	60	60	60	60	60	60

Congestion delay	0	0	0	0	0	0

4.3 Bi-objective optimization results

The parameters of the ridesharing system are recorded in 11 different weight combination levels. In the first level, the VMT objective function weight is 0, and the TT objective function weight is 10. This weight combination indicates that the importance of extra distance is considered 0 (0%), and the weight for minimizing TT is considered 10 (100%). In the first scenario, the system attempted to determine the best paths to minimize the amount of "TT multiplied by the weight." As the TT weight is 0, increasing the VMT does not add any weight to the system. In the result of the first weight combination, the system will be optimized for the lowest possible TT time without considering VMT consequence. In the next levels, the weights for time and distance will change (i.e., time weight factor from 10 to 0 and distance weight factor from 0 to 10 presented below).

Weight combination #	Weights assigned to objective functions (i.e., Minimization of VMT and Minimization of TT)
Weight combination 1:	VMT=10 (100%), TT=0 (0%)
Weight combination 2:	VMT=9 (90%), TT=1 (10%)
Weight combination 3:	VMT=8 (80%), TT=2 (20%)
Weight combination 4:	VMT=7 (70%), TT=3 (30%)
Weight combination 5:	VMT=6 (60%), TT=4 (40%)
Weight combination 6:	VMT=5 (50%), TT=5 (50%)
Weight combination 7:	VMT=4 (40%), TT=6 (60%)
Weight combination 8:	VMT=3 (30%), TT=7 (70%)
Weight combination 9:	VMT=2 (20%), TT=8 (80%)
Weight combination 10:	VMT=1 (10%), TT=9 (90%)
Weight combination 11:	VMT=0 (0%), TT=10 (100%)

Variation in assigned weights changes the routes for ridesharing drivers. By changing the paths, the number of vehicles and passengers who can share their ride will also change. As a result, the amount of VMT and TT in the entire system will change. To perform a better assessment of multi-objective optimization in a ridesharing system, 12 different demand and supply scenarios are performed and listed below:

Scenario # 1:	Number of vehicles = 15, Number of passengers = 20
Scenario # 2:	Number of vehicles = 15, Number of passengers = 30
Scenario # 3:	Number of vehicles = 15, Number of passengers = 40
Scenario # 4:	Number of vehicles = 15, Number of passengers = 50
Scenario # 5:	Number of vehicles = 15, Number of passengers = 60
Scenario # 6:	Number of vehicles = 50, Number of passengers = 40
Scenario # 7:	Number of vehicles = 60, Number of passengers = 40
Scenario # 8:	Number of vehicles = 70, Number of passengers = 40
Scenario # 9:	Number of vehicles = 60, Number of passengers = 60
Scenario # 10:	Number of vehicles = 60, Number of passengers = 80
Scenario # 11:	Number of vehicles = 60, Number of passengers = 100
Scenario # 12:	Number of vehicles = 60, Number of passengers = 120

To assess the impact of multi-objective optimization on a ridesharing system, different scenarios have been developed by changing the number of vehicles and riders in each scenario. The scenarios have been categorized into three different groups. Group 1 includes scenario #1 to scenario #5 where the number of vehicles is 15, and the number of passengers is varied between 20 to 60. Group 2 has three scenarios from scenario #6 to scenario #8 where the number of passengers is 40, and the number of vehicles is varied between 50 to 70, and the scenarios from #9 to #12 are categorized in group 3 where the number of vehicles is 60 and the number of passengers varied between 60 to 120.

The reason for considering the different numbers for passengers in group 1 is to measure the effectiveness of the ridesharing matching model with a low number of vehicles and a varying number of passengers. After analyzing the results of group 1, the difference between the results of the model in group 2, when the number of vehicles is more than the number of passengers in the network, is investigated. The third group is created to evaluate the results of the model when the number of drivers and passengers in the ridesharing system is more than group 1.

The first group has five sets of scenarios (scenario #1 to #5) in which the number of vehicles was 15, and the number of passengers was 20, 30, 40, 50, and 60, respectively. The second group has

3 scenarios (scenarios #6 to #8) in which the number of passengers was 40, and the number of vehicles was 50, 60, and 70. And the third group includes 4 scenarios (scenario #9 to #12) in which the number of vehicles was 60, and the number of passengers was 60, 80, 100, and 120. For each of the scenarios, the bi-objective problem was solved eleven weight combinations between objective function 1 and objective function 2. For example, in the first combination, the weight of objective 1 is 10, and the weight for objective 2 was 0, wherein in the second combination, the weight of objective 1 is 9, and the weight for objective 2 was 1. Similarly, in the eleventh combination, the weight of objective 1 is 0, and the weight for objective 2 was 10. These different weight combinations were used to estimate the sacrifice the systems need to accept when a certain weight is assigned to both objectives (e.g., combination 2 provided weights of 9 and 1 to objective function 1 and 2, respectively). The corresponding bi-objective optimization solutions are presented in Fig. 1 to Fig. 5.

Eleven solutions corresponding to 11 weight combinations discussed before are presented in Figure 1 for the demand and supply scenario #1. In the second weight combination (corresponding to VMT weight=9 and TT weight =1 compared to VMT weight=10 and TT weight =0), the system decreased 40 minutes of the TT, but the VMT increased by five miles. In the third weight combination (corresponding to VMT weight=8 and TT weight =2), the system saved 31 minutes while VMT grew by five miles. The system compromised increasing eight miles of VMT for saving 25 minutes of TT in the fourth weight combination compared to the third weight combination. In the fourth solution, the system experienced three miles increment in VMT to decrease six minutes of TT. In the next two solutions, VMT grew by one and four miles to have three and two minutes saving in TT. The seventh solution did not show TT saving or VMT increment. The eighth solution decreased TT by two minutes and increased VMT by six miles. The ninth solution experienced a one-mile increment to have 7 minutes saving in TT, and no change was observed in the last solution

In the second scenario, the first solution showed two miles increment in VMT in return for saving 74 minutes of TT and the second solution saved 10 minutes of TT when the VMT was raised by two miles. The third solution showed five miles increments in VMT to save 15 minutes of TT. There was no change in the fourth solution, but the fifth solution increased five miles to the VMT to decrease six minutes of TT in the system. The sixth scenario showed a two minutes TT reduction and five miles increment, and the seventh scenario and after did not have a major change in the records.

The first solution in the third scenario had 63 minutes saving on TT when it showed 22 miles VMT increment. The second solution had only 7 miles raising in VMT to save 42 minutes TT, and the third solution saved 10 minutes of TT while increasing three miles of VMT. The fourth solution raised six miles of VMT to decrease five minutes of the TT, while the fifth solution did not show any changes in the two objectives. The sixth solution saved four minutes TT over four miles increasing in VMT, and there was no change in the system's records for the other solutions.

In the fourth scenario, the first solution showed a one-mile raising in VMT to save 23 minutes of TT, and the second solution had six miles increment in VMT, and the TT saving was 37 minutes in this solution. The third solution saved 36 minutes of TT and increased eight miles, and the fourth solution saved 13 minutes with no change in the VMT. The fifth solution raised one-mile VMT to save six minutes TT, and the sixth solution saved three minutes of TT when 34 miles increased in VMT. There was no change in the solutions after that.

The fifth scenario was the last scenario in this group, and the first solution had a saving of TT by decreasing 12 minutes of TT while only raising VMT by one mile. The second solution also showed a saving by decreasing 11 minutes of TT and one-mile increment in VMT. In the third solution, we still see a reasonable saving of 41 minutes of TT when the VMT increased by 14 miles. The fourth solution decreased one minute from the TT but added five miles to VMT. The amount of TT saving and VMT increment were the same in the fifth solution, which was three minutes and three miles. In the sixth scenario, the amount of VMT increment was 10 miles to save 6 minutes of TT. The seventh solution showed a slight change in TT and VMT (two minutes saving, and four miles increment). There was no change in the eighth solution, and there was an increase of four miles in VMT to save one minute in TT in the ninth solution. There was no change in the 10th solution.

Appendix B includes the solutions for other scenarios corresponding to eleven weight combinations.

In the next group of scenarios, three different scenarios (scenario #6 to #8) were developed where the number of passengers was 40 passengers, and the number of vehicles was 50, 60, and 70 vehicles in scenarios 6, 7, and 8, respectively.

In the last group of scenarios (scenario #9 to #12), the number of passengers changes, while the number of vehicles remains fixed (i.e., 60 vehicles)

The results show that if multi-objective optimization is implemented considering conflicting objectives, the ridesharing system could lead to a reduction in systemwide trip time, and the ridesharing system can serve more travelers using fewer vehicles considering the conflicts with systemwide VMT concerns.

Also, it is proved that the method we used in this research was able to optimize the conflict between the two objectives. The system shows the method successfully optimized the conflict between the two objectives and was able to reduce the TT increment in the entire network while the system has saved in VMT (as it is expected from a ridesharing system).

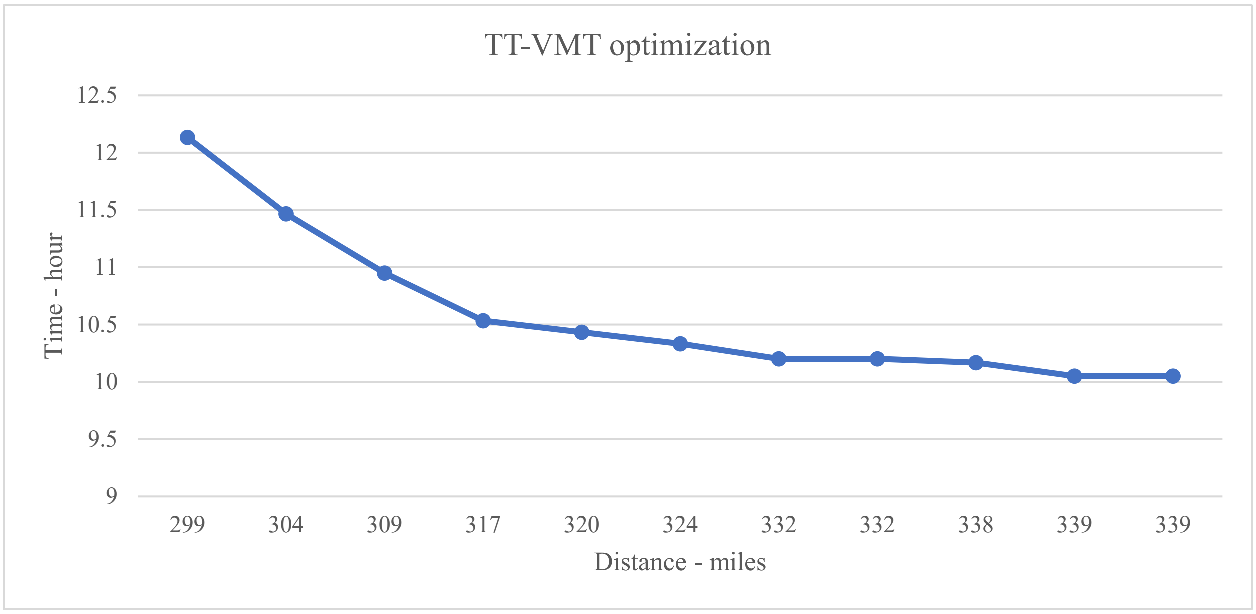


Figure 1: The changes in systemwide distance and trip time of the entire network after multi-objective optimization, Vehicles = 15 Passengers = 20

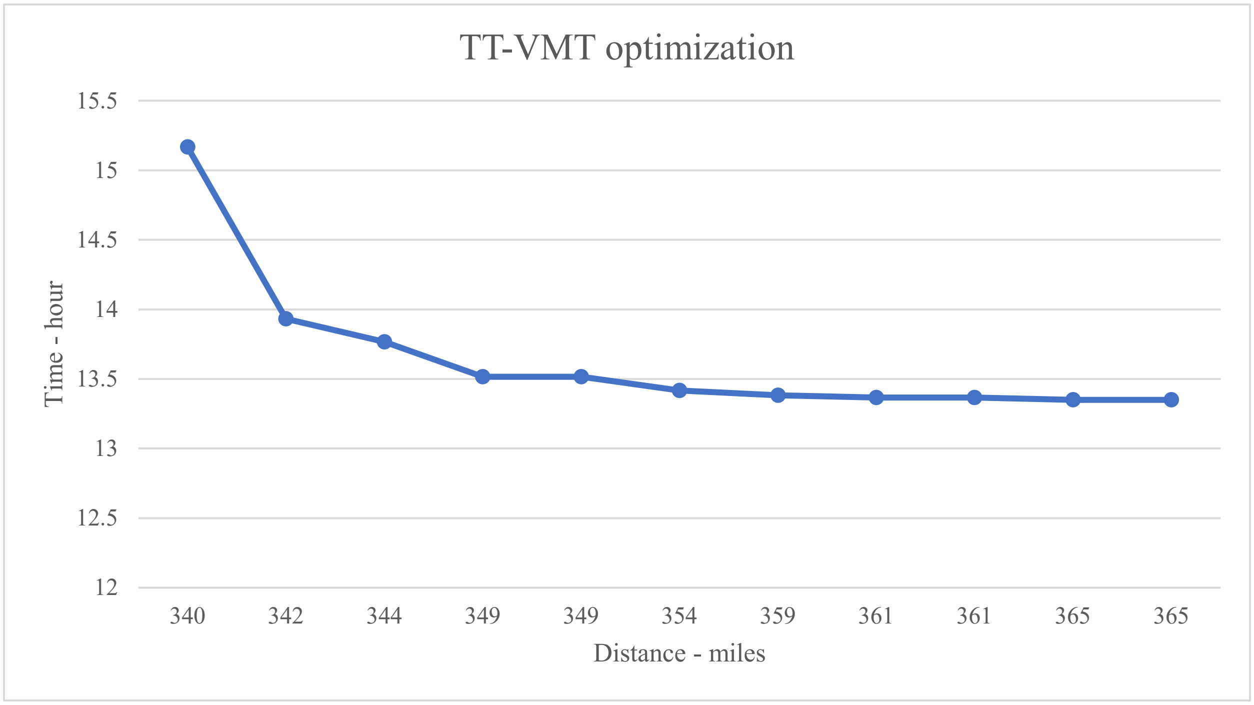


Figure 2: The changes in distance and time of the entire network after multi-objective optimization, Vehicles = 15 Passengers = 30

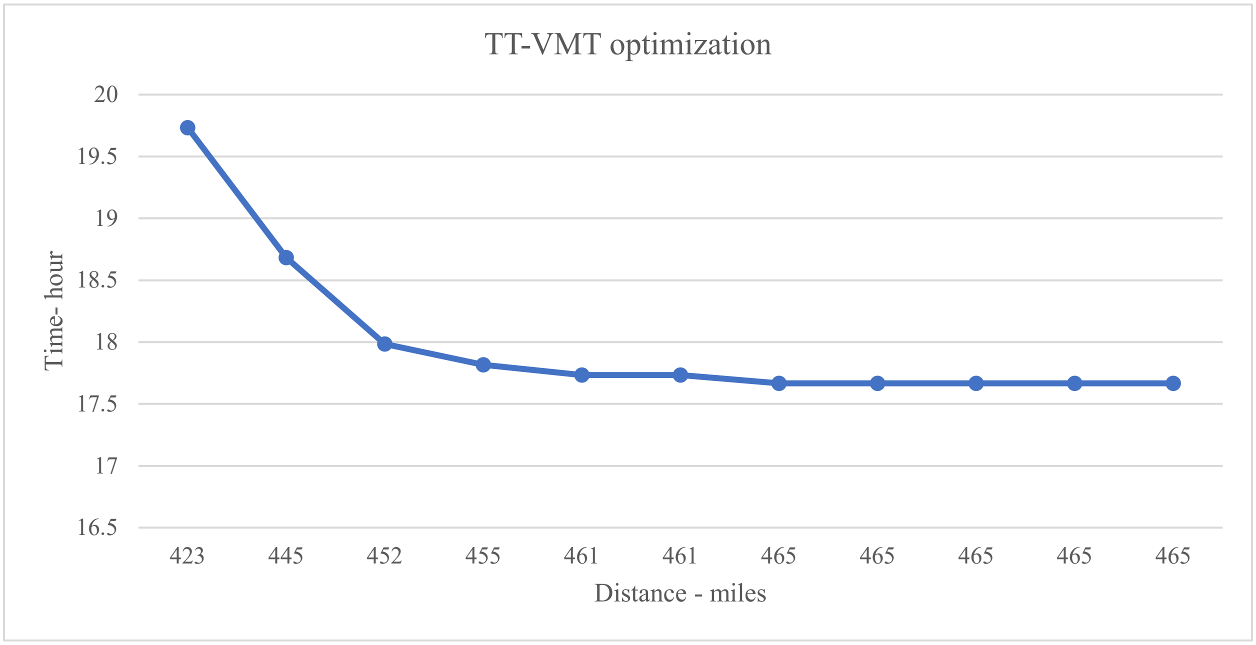


Figure 3: The changes in distance and time of the entire network after multi-objective optimization, Vehicles = 15 Passengers = 40

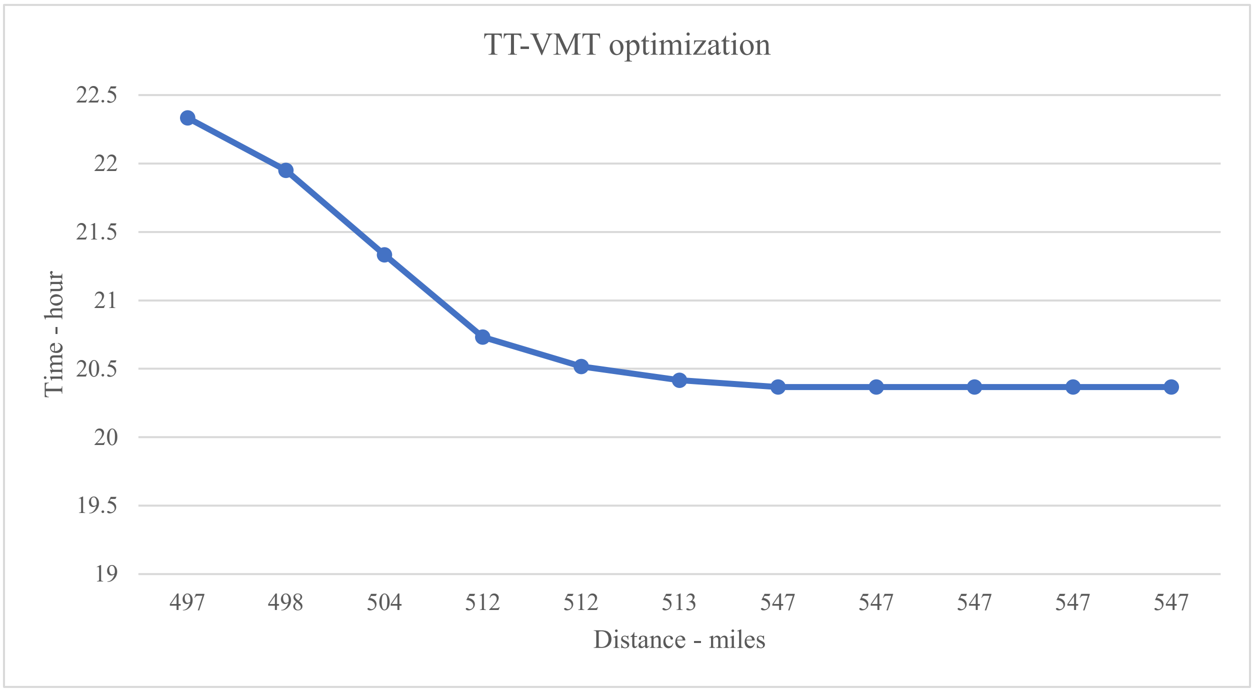


Figure 4: The changes in distance and time of the entire network after multi-objective optimization, Vehicles = 15 Passengers = 50

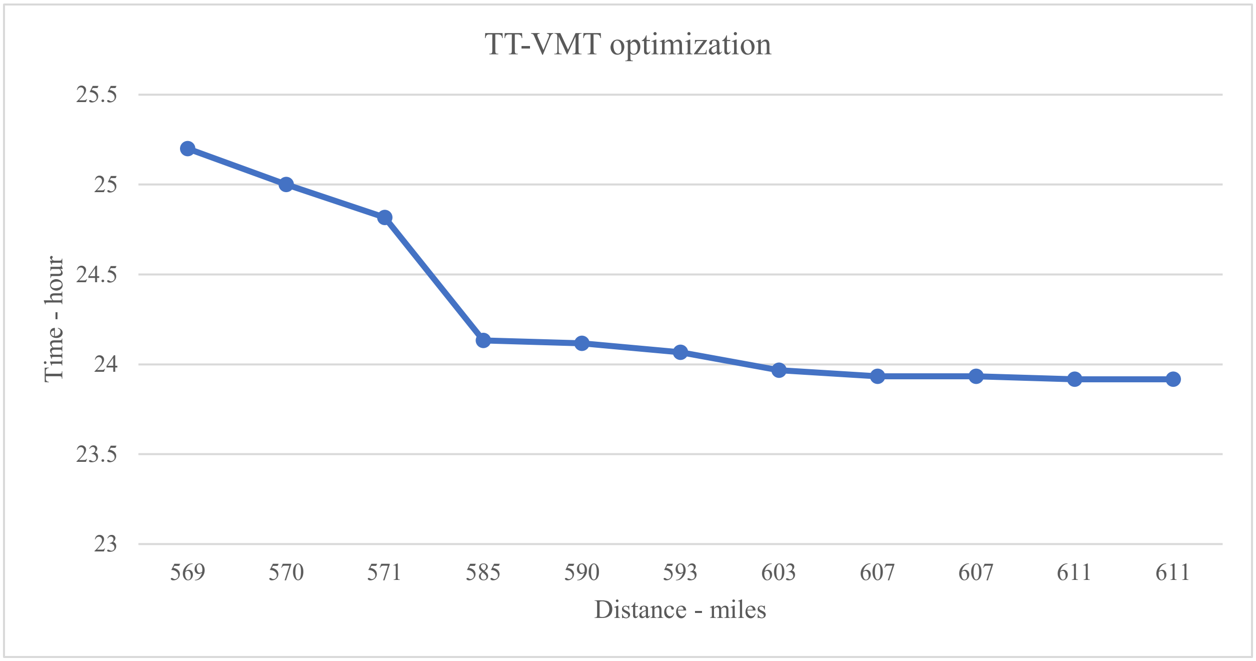


Figure 5: The changes in distance and time of the entire network after multi-objective optimization, Vehicles = 15 Passengers = 60

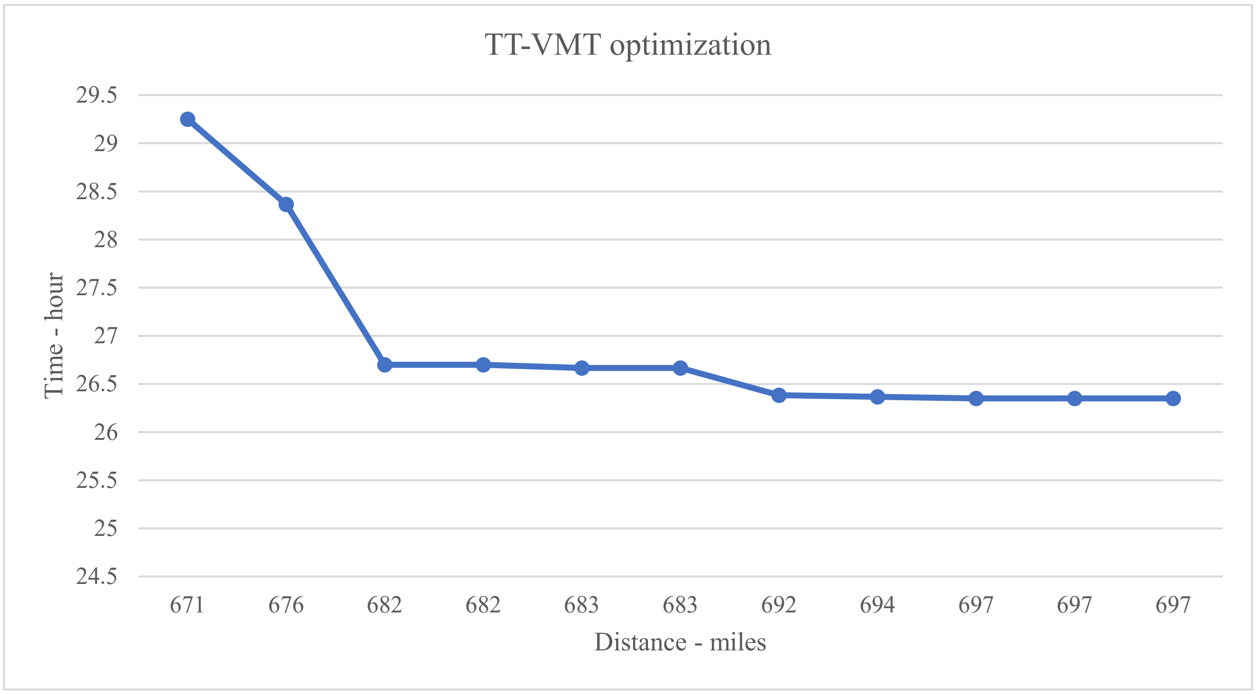


Figure 6: The changes in distance and time of the entire network after multi-objective optimization, Vehicles = 50 Passengers = 40

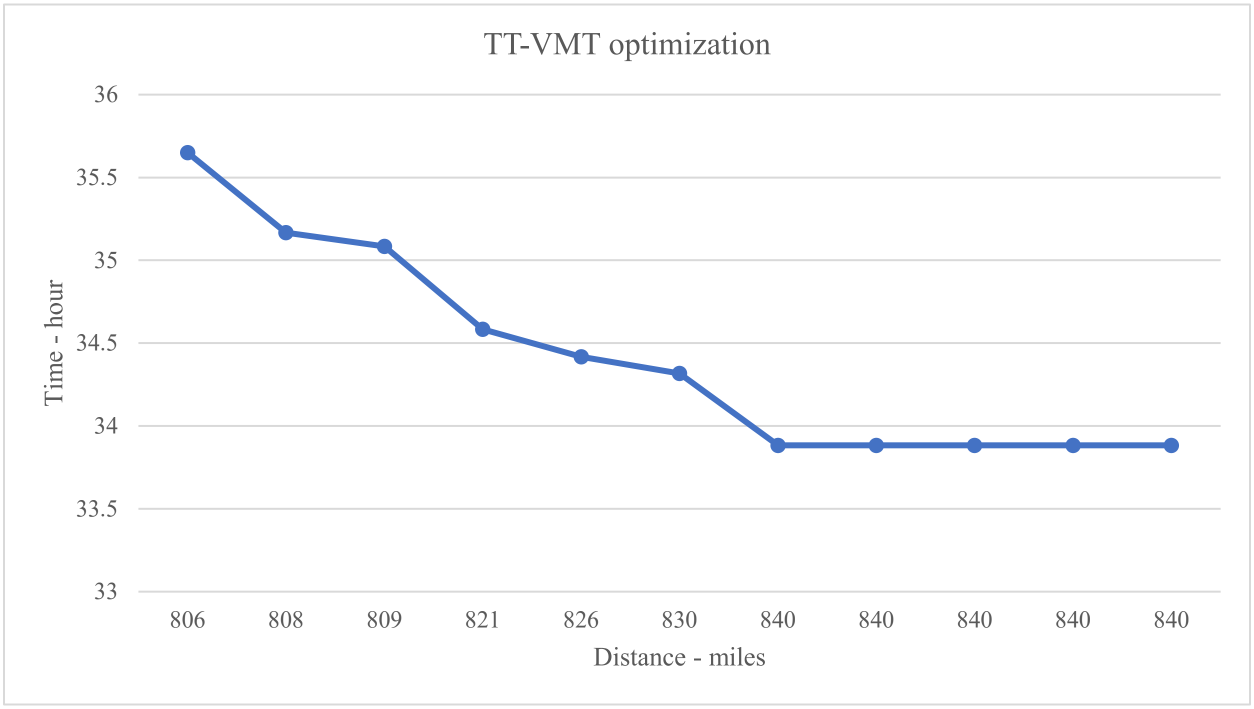


Figure 7: The changes in distance and time of the entire network after multi-objective optimization, Vehicles = 60 Passengers = 40

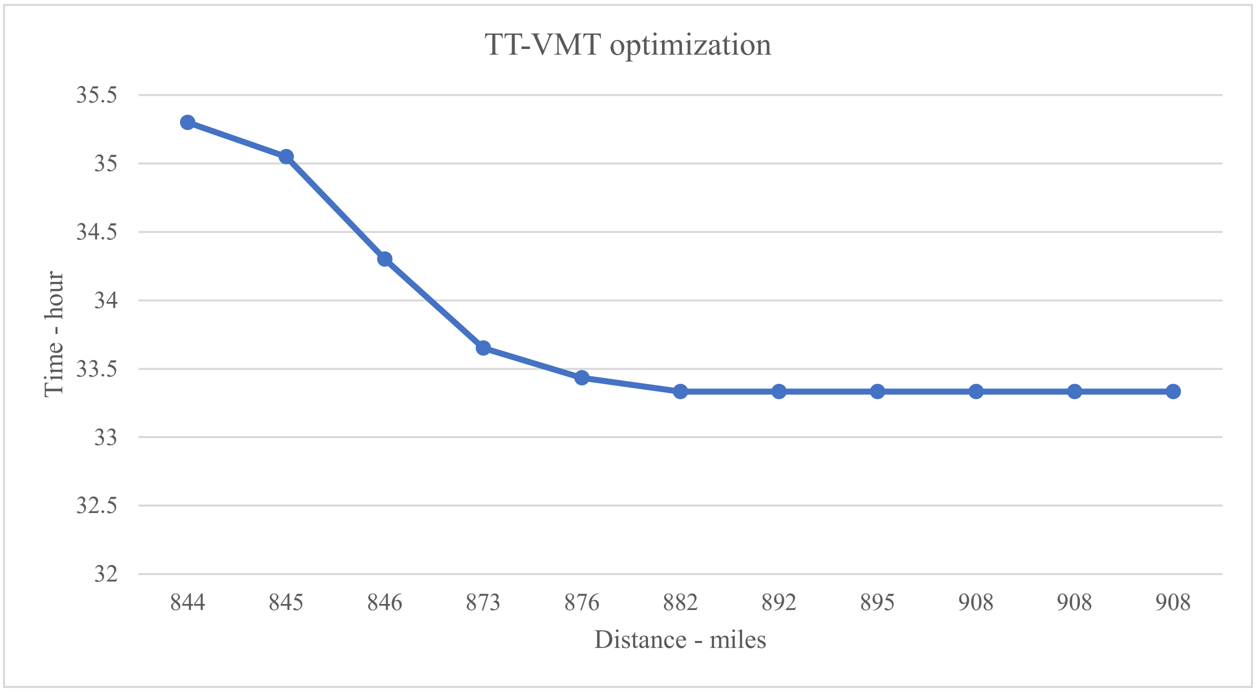


Figure 8: The changes in distance and time of the entire network after multi-objective optimization, Vehicles = 70 Passengers = 40

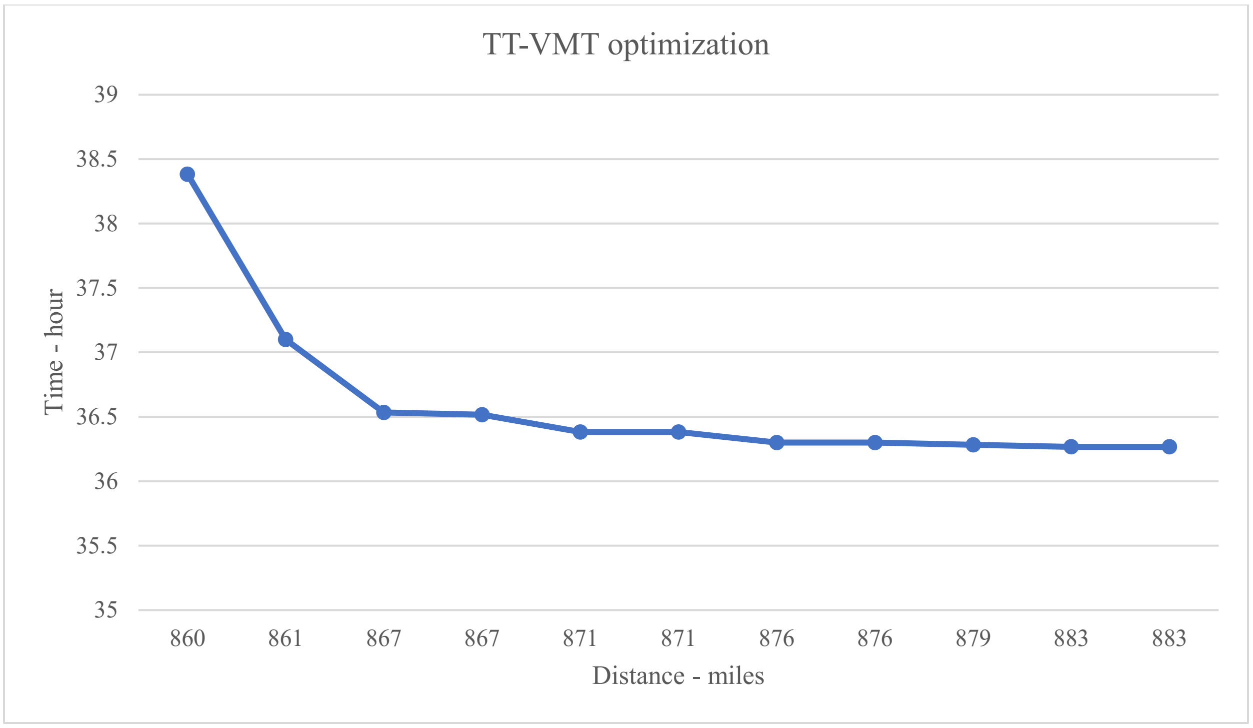


Figure 9: The changes in distance and time of the entire network after multi-objective optimization, Vehicles = 60 Passengers = 60

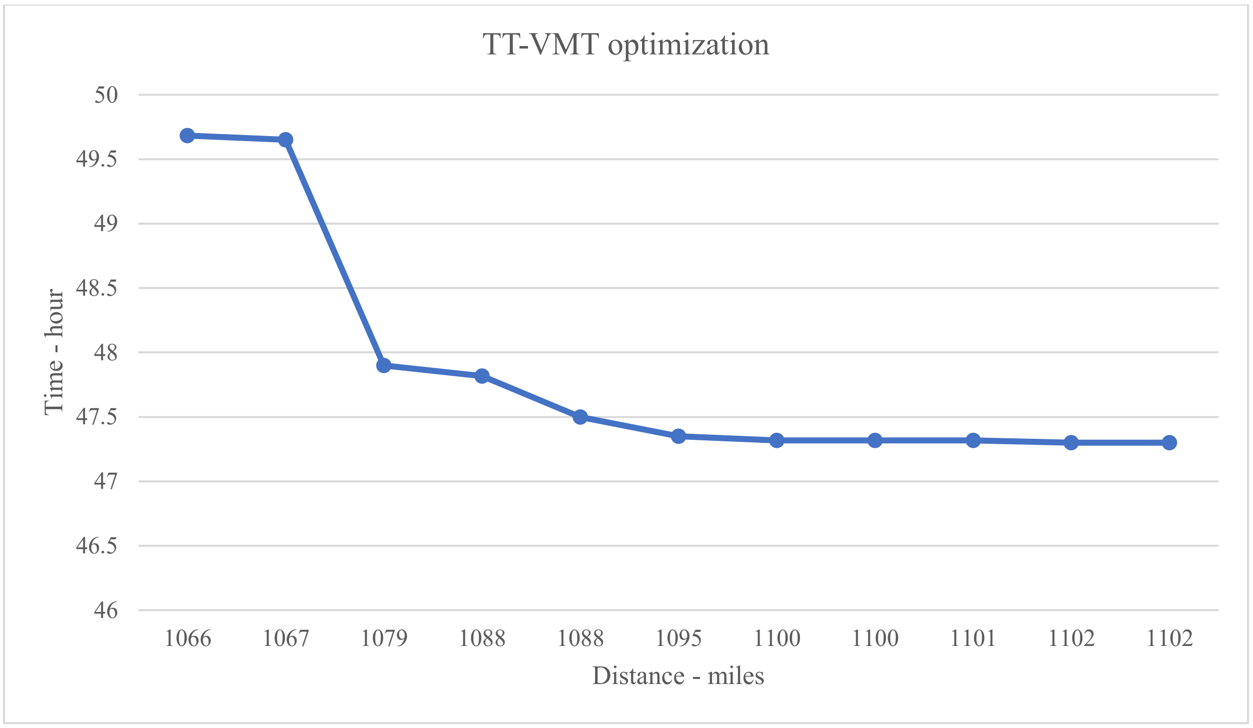


Figure 10: The changes in distance and time of the entire network after multi-objective optimization, Vehicles = 60 Passengers = 80

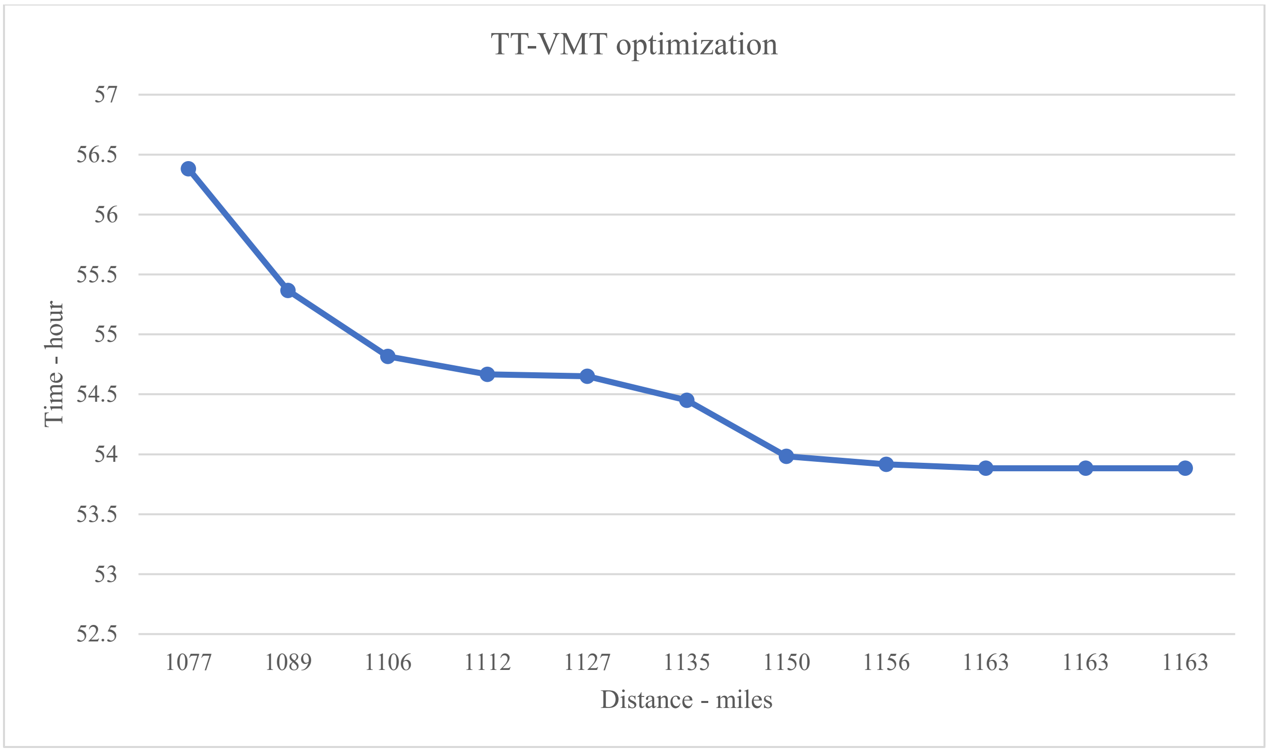


Figure 11: The changes in distance and time of the entire network after multi-objective optimization, Vehicles = 60 Passengers = 100

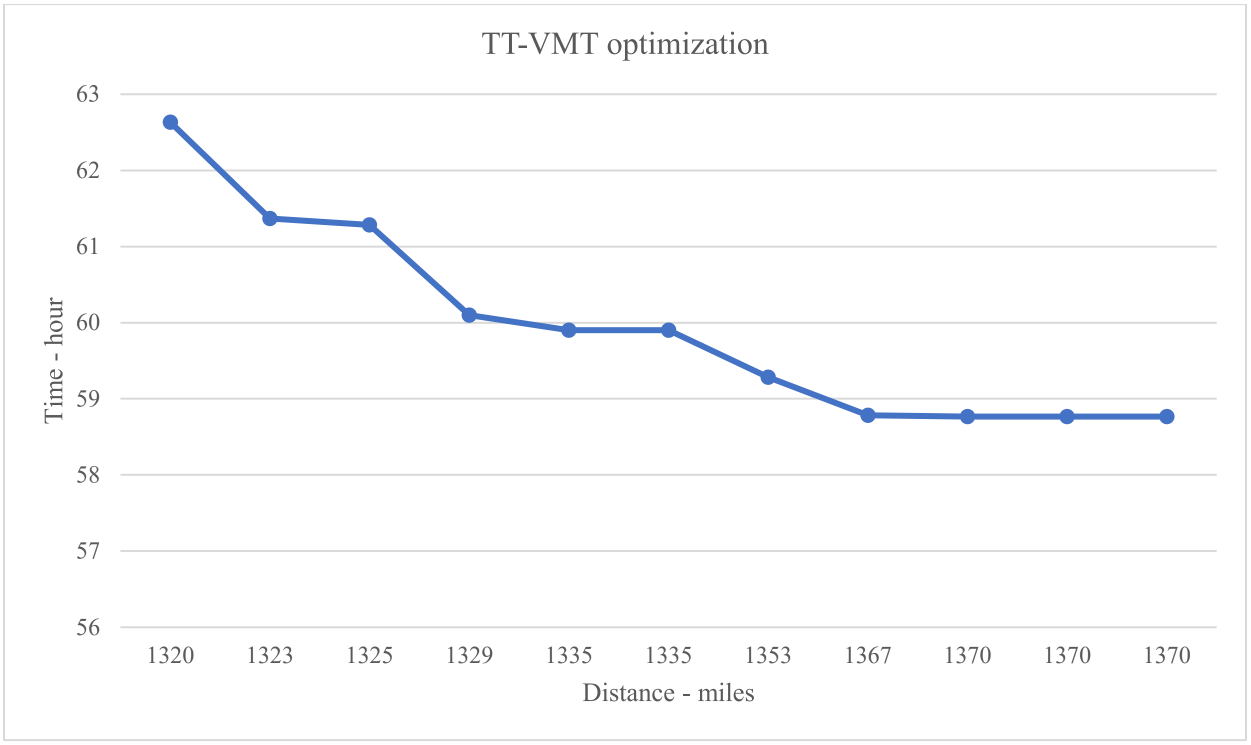


Figure 12: The changes in distance and time of the entire network after multi-objective optimization, Vehicles = 60 Passengers = 120

4.4 Trade-off analysis

The tradeoff analysis provides an increase or decrease in one objective function outcome due to a unit increase or decrease in another objective function. The tradeoff means if one of the objectives loses one unit amount in terms of its outcome, then it will impact how much incremental gain/loss, the other objective will experience. The formula to calculate the tradeoff rate used in this research is: $T_t(\text{x}) = \frac{\partial f_t}{\partial f_d}$ Where T_t represents tradeoff value for the objective t (trip time objective) at the solution x, f_d is VMT value and d= 1,..,11. f_t is TT value and t= 1,..,11. Tradeoff value is a specific value in multi-objective optimization, and it is also a good value for decision-makers. As we plotted the Pareto optimal graphs for different scenarios as presented in section 4.3, we can calculate the tradeoff between the two objectives for the Pareto optimal graph of each scenario.

The tradeoffs of the two objective functions for all demand and supply scenarios presented in Section 4.3 are shown in Fig. 13 to Fig. 24. As the objectives have different units, the tradeoff unit is (hour/ miles) and the tradeoff values show the reduction of a matching-objective value when the value of another matching-objective was increased. These numbers show the amount of the TT decrement and vehicle miles traveled increment, which is the result of optimizing the conflict between the two objectives. To quantify these values, I calculated the tradeoff values, which are more tangible to realize the effects of the method to optimize the conflicts between the two objectives. The tradeoff values show how much travel distance could be reduced if we have a unit increase in trip time. For example, in the 2nd level (weights for VMT=1, TT=9) of the first scenario (15 vehicles and 20 passengers in the network), the systemwide travel distance is 304 miles, and the systemwide trip time is 11 hours and 28 minutes. The tradeoff value is 0.13 hour (8 minutes), which means 0.13 hour (8 minutes) of TT can be recovered if the system allows one-mile increment in terms of VMT objective compared to the previous state (travel distance was 299 miles and trip time was 12 hours and 8 minutes). Tradeoff values for different demand and supply scenarios indicate that the multi-objective optimization method has the potential to reveal tradeoff values between the two conflicting objectives. Tradeoff values also indicate that the method used in this study worked well in terms of optimizing the conflicts between the two objectives, TT and VMT.

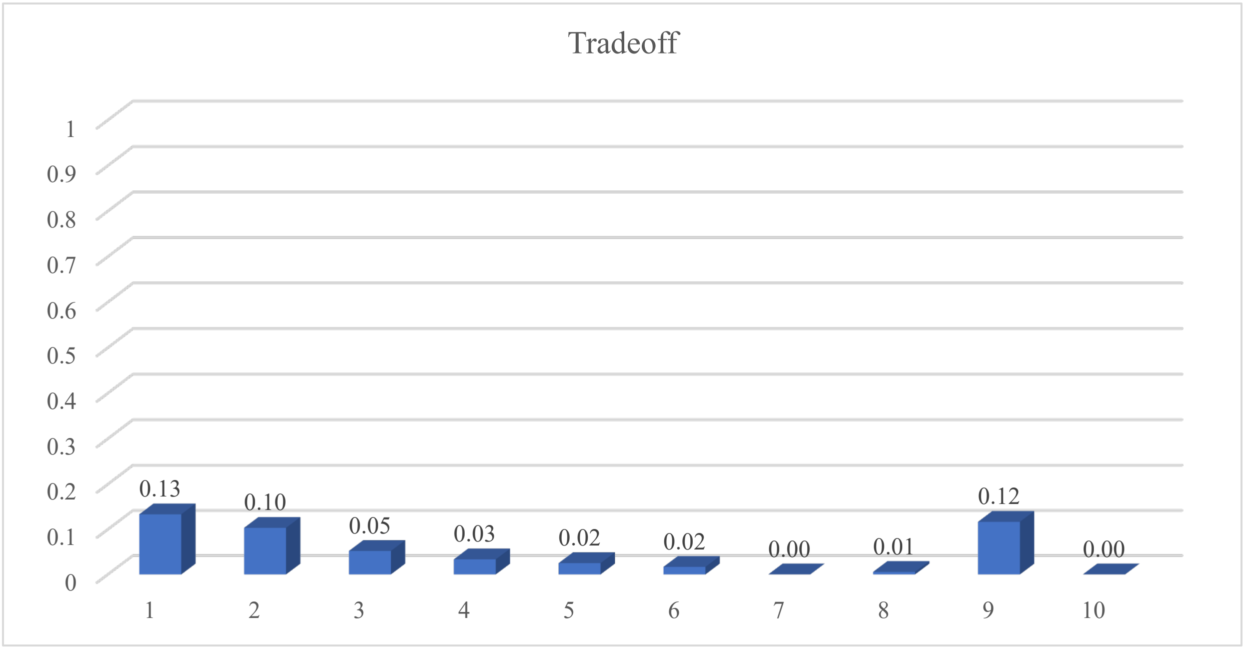


Figure 13: Tradeoff between system wide TT and VMT for Scenario #1 (Number of Vehicle=15, Number of Passengers:20).

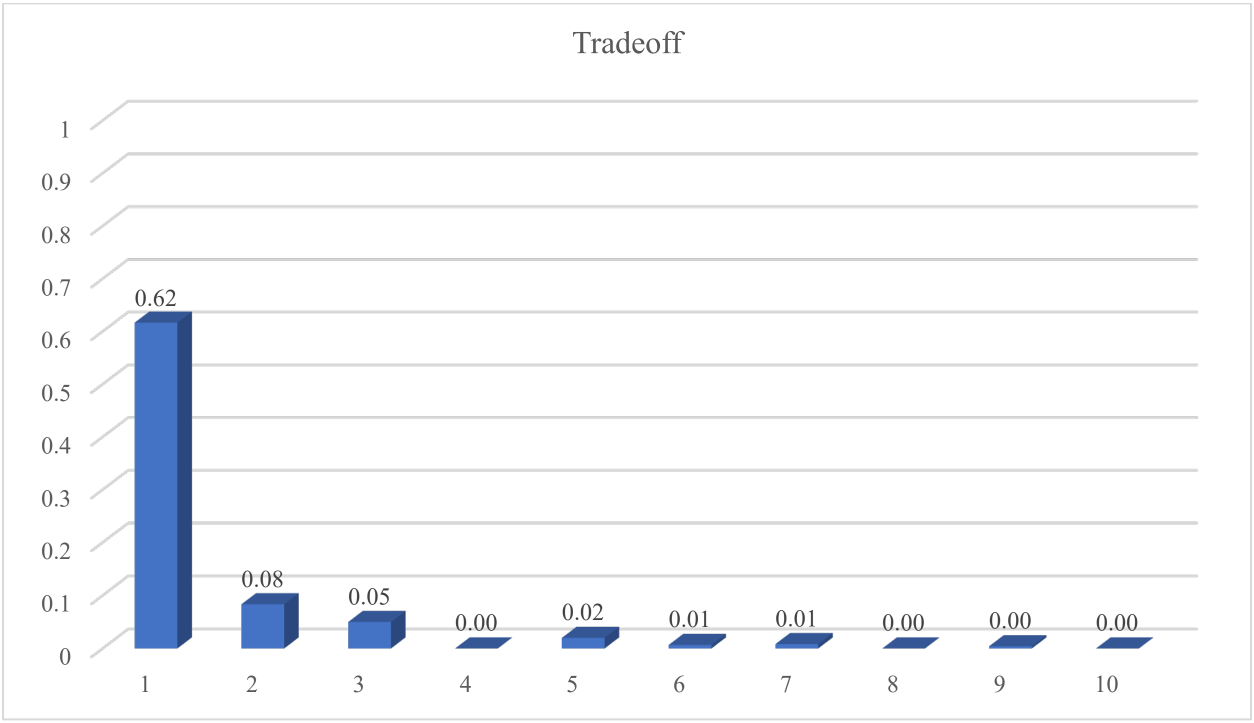


Figure 24: Tradeoff between system wide TT and VMT for Scenario #2 (Number of Vehicle=15, Number of Passengers:30)

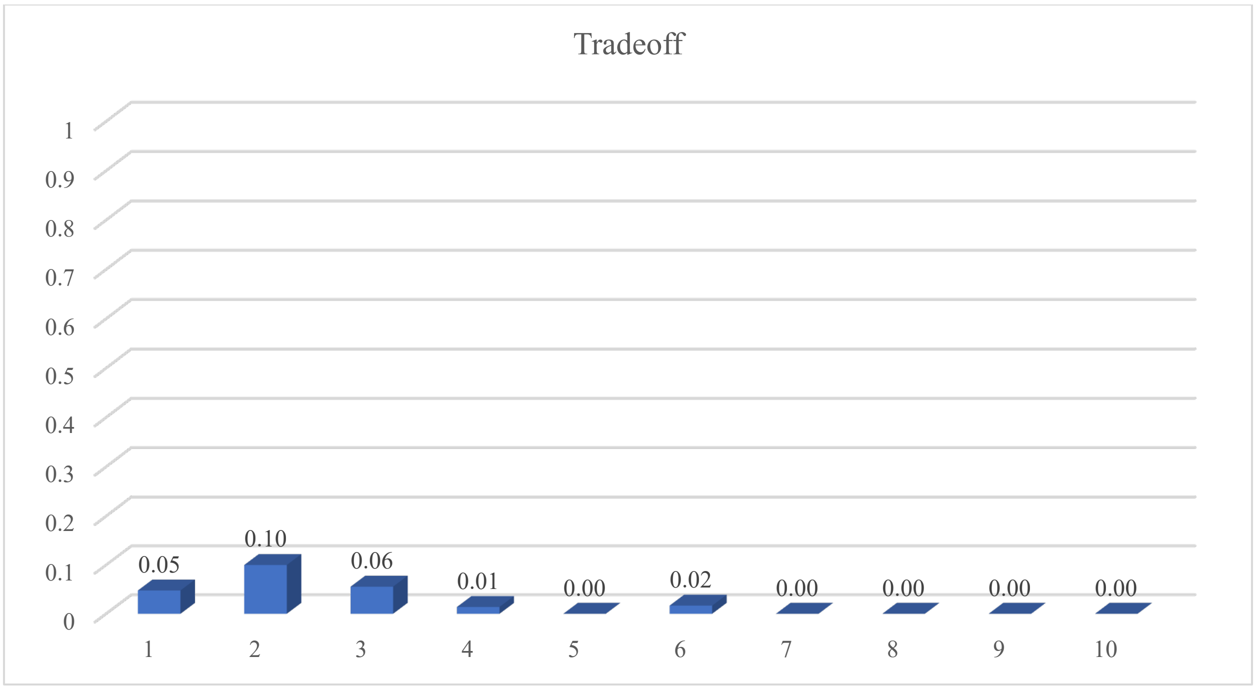


Figure 35: Tradeoff between system wide TT and VMT for Scenario #3 (Number of Vehicle=15, Number of Passengers:40)

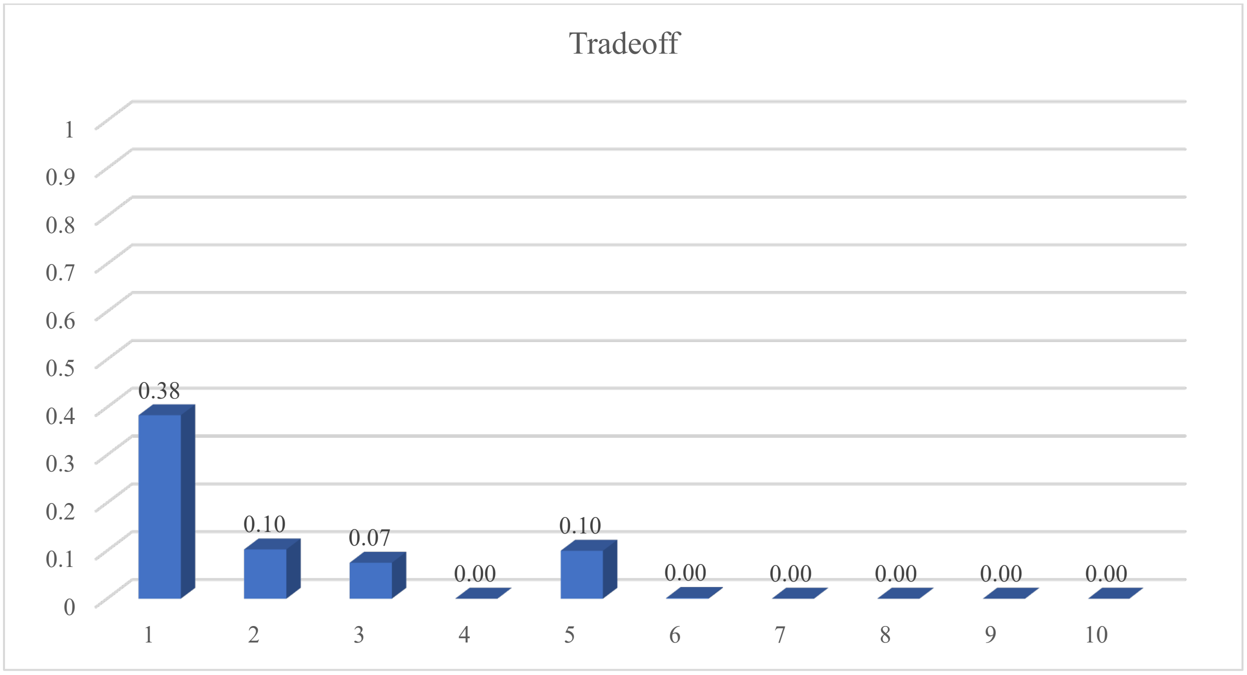


Figure 46: Tradeoff between system wide TT and VMT for Scenario #4 (Number of Vehicle=15, Number of Passengers:50)

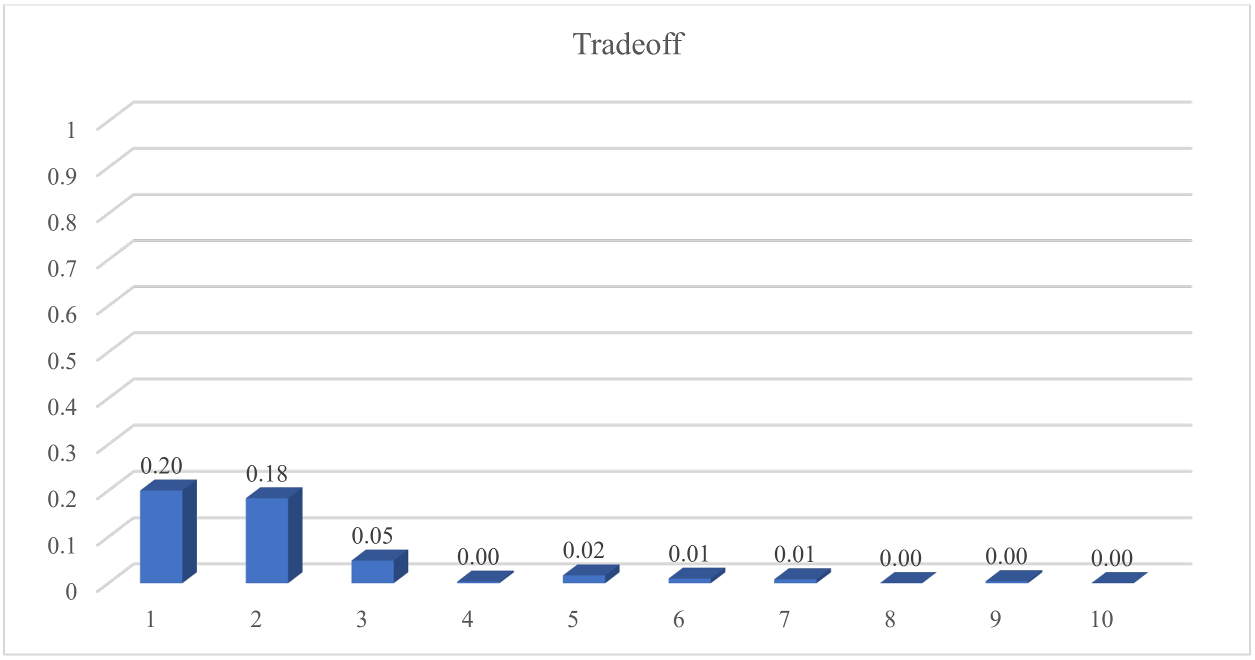


Figure 57: Tradeoff between system wide TT and VMT for Scenario #5 (Number of Vehicle=15, Number of Passengers:60)

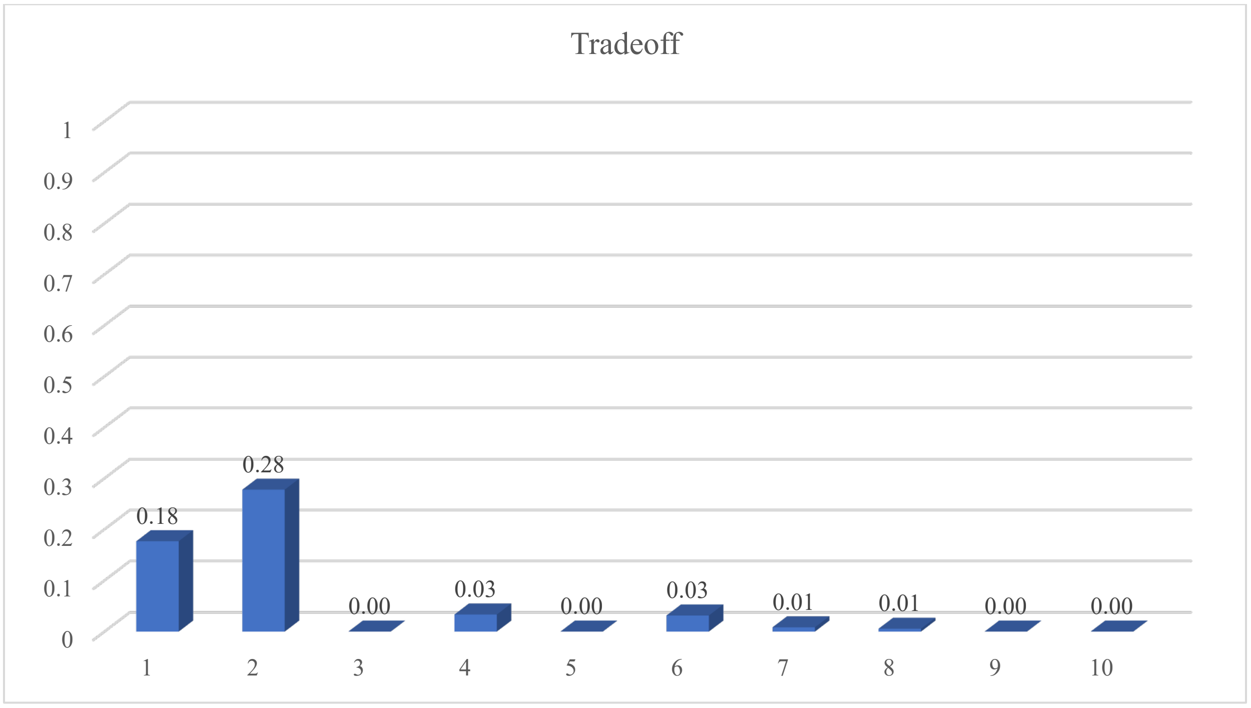


Figure 68: Tradeoff between system wide TT and VMT for Scenario #6 (Number of Vehicle=50, Number of Passengers:40)

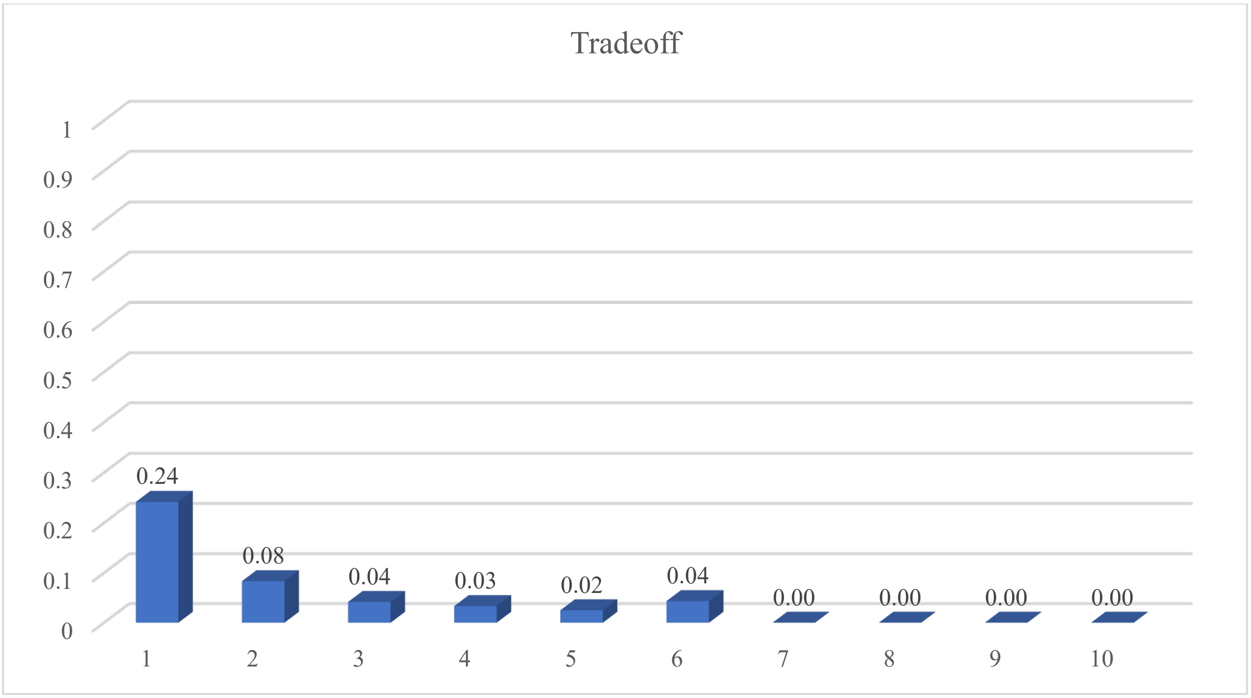


Figure 79: Tradeoff between system wide TT and VMT for Scenario #7 (Number of Vehicle=60, Number of Passengers:40)

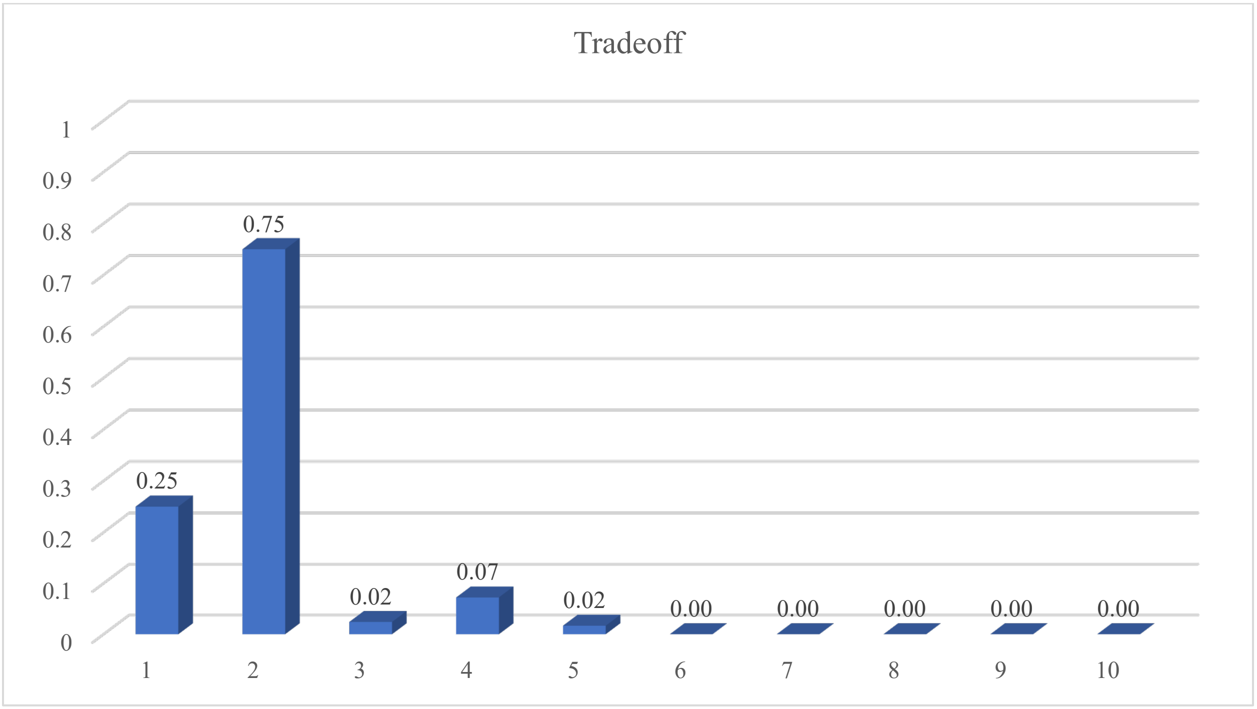


Figure 20: Tradeoff between system wide TT and VMT for Scenario #8 (Number of Vehicle=70, Number of Passengers:40)

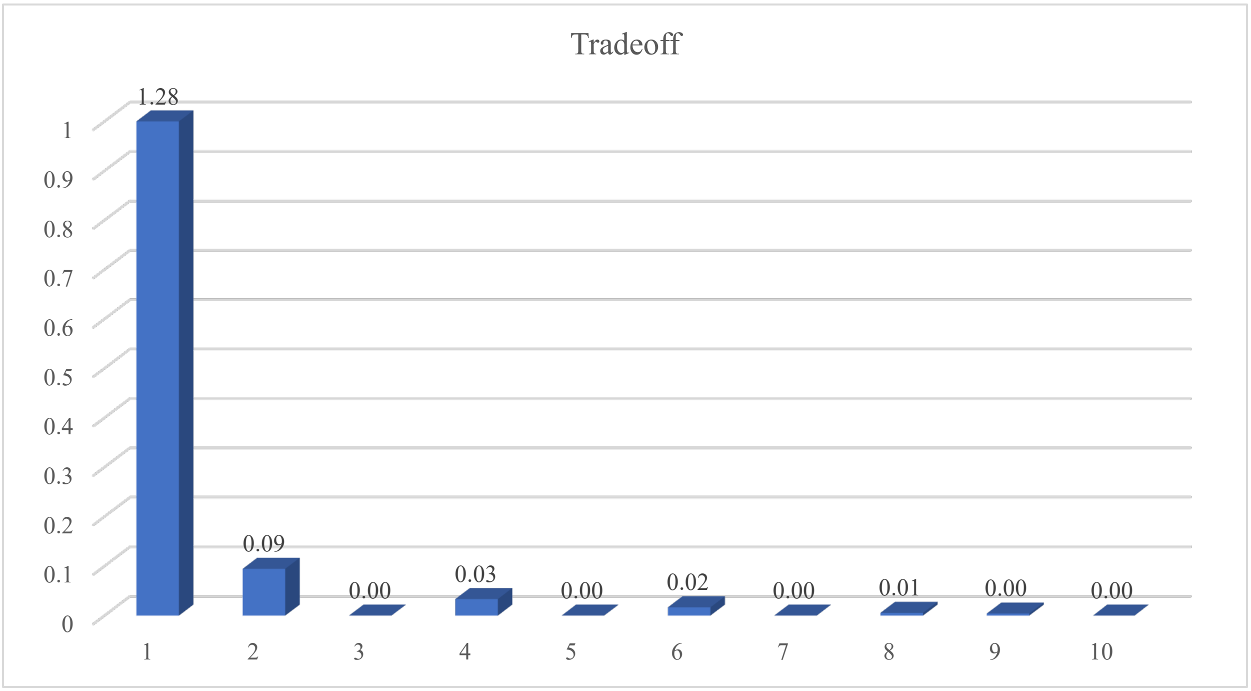


Figure 21: Tradeoff between system wide TT and VMT for Scenario #9 (Number of Vehicle=60, Number of Passengers:60)

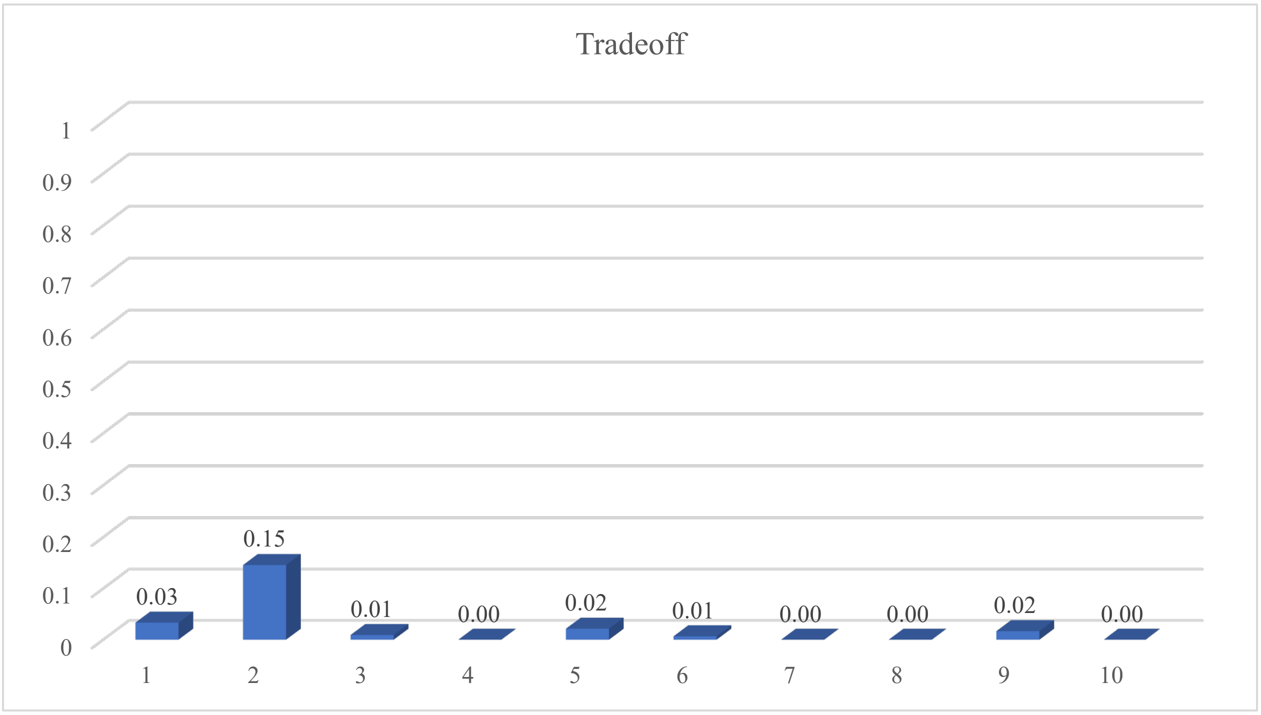


Figure 22: Tradeoff between system wide TT and VMT for Scenario #10 (Number of Vehicle=60, Number of Passengers:80)

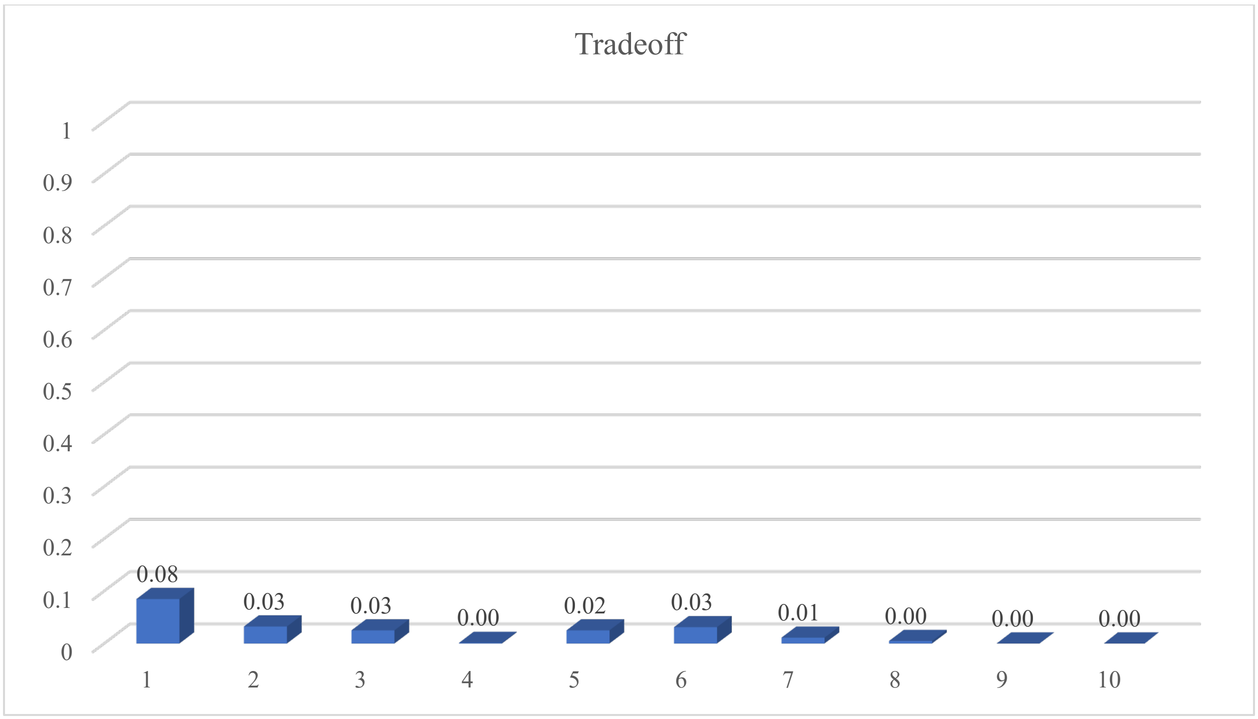


Figure 23: Tradeoff between system wide TT and VMT for Scenario #11 (Number of Vehicle=60, Number of Passengers:100)

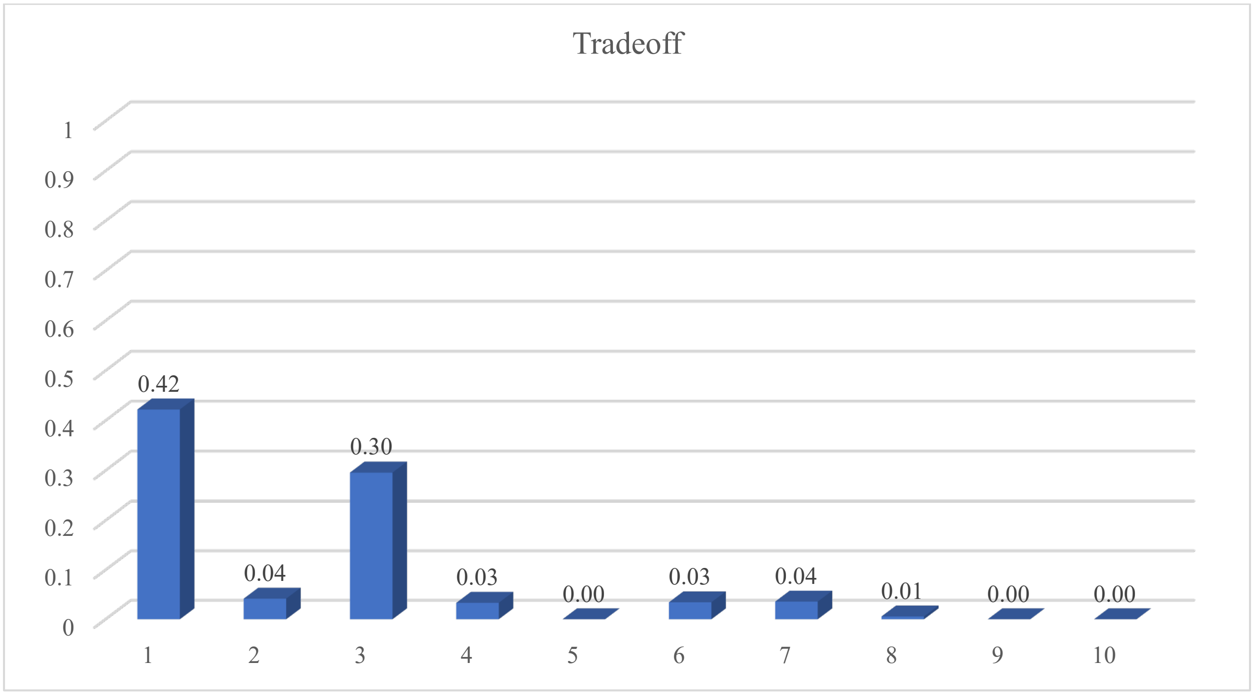


Figure 24: Tradeoff between system wide TT and VMT for Scenario #12 (Number of Vehicle=60, Number of Passengers:120)

CHAPTER 5: CONCLUSIONS AND FUTURE RESEARCH DIRECTION

One of the issues in a ridesharing system is the conflict between the different matching-objectives. These conflicts have a detrimental impact on the system and could decrease the benefits for some stakeholders. The previous studies on optimization in a ridesharing system have been mostly for single objectives. In this research, however, the conflict between two objectives (i.e., Trip Time/TT and Vehicle Miles Traveled/VMT) was studied applying a multi-objective optimization technique. Based on the results obtained from this research, it is concluded that in a multi-objective optimization state, the model optimized both objectives in all scenarios with a tradeoff in both objectives. Moreover, in all demand and supply scenarios considered in this study, a reduction in TT was observed with increasing VMT in the entire ridesharing network. Multi-objective optimization to optimize the conflicts between the objectives could also improve the ridesharing performance and, by optimizing the conflicts between two objectives, the system can reach an optimum point of the objectives in the whole system. All in all, multi-objective optimization is effective in the improvement of the performance of a ridesharing system.

Therefore, the contributions of this research can be highlighted as considering TT and VMT effects; decision-makers can better decide when they choose to invest in or operate a ridesharing system. A matching method in a ridesharing system can be developed based on the outcomes of this study and, transportation decision-makers can improve ridesharing systems with regard to the various perspectives of stakeholders.

This study was built upon some limitations, which could be addressed in future studies. Thus, for future studies, it is recommended that the optimization of more than two objectives in a ridesharing system could be the subject of a future study. A low number of riders and passengers have been used in this research, which can be enhanced in future studies to assess the effect of a high number of riders and passengers in a ridesharing network. The maximum allowed extra trip time and travel distance to 50% and 100% is another limitation of this study. Hence, future works can be performed based on different limitations in order to assess different levels of these parameters. Twelve scenarios with a different number of drivers and passengers in the network have been performed to assess the results for different ratios of riders and drivers in a network. Future studies can be done using different ratios to assess the impact of multi-objective optimization in a higher or lower ratio of drivers and passengers and to identify the impact of different ratios on a multi-objective optimization model. Travel cost minimization is an important objective that has not been considered in this study. Multi-objective optimization using travel cost and other matching-objectives can provide a more comprehensive perspective of the impacts of the matching-objectives on a ridesharing system and the other matching objectives. Traffic congestion has not been considered in optimizing the matching-objectives. As congestion is a recurring issue of an urban transportation system, the inclusion of congestion parameters could reveal critical insights. In this study, vehicle occupancy was limited to three passengers. Thus, future research could deploy ridesharing vehicles with different seating capacities (i.e., the van has higher seating capacity) to understand its effect on system performance.

REFERENCES

Agatz, Niels, et al. "Optimization for dynamic ride-sharing: A review." European Journal of Operational Research 223.2 (2012): 295-303.

Mallus, Matteo, et al. "Dynamic carpooling in urban areas: design and experimentation with a multi-objective route matching algorith." *Sustainability* 9.2 (2017): 254.

Cáp, Michal, and Javier Alonso Mora. "Multi-objective analysis of ridesharing in automated mobility-on-demand." Proceedings of RSS 2018: Robotics-Science and Systems XIV (2018).

Statista Research Department, US Department of Transportation; Federal Highway Administration. "Number of Vehicles in U.S." Statista, Statista Research Department, 22 July 2019, www.statista.com/statistics/183505/number-of-vehicles-in-the-united-states-since-1990/.

Freemark, Yonah. "Travel Mode Shares in the U.S." The Transport Politic, The Transport Politic, 23 Aug. 2019, www.thetransportpolitic.com/databook/travel-mode-shares-in-the-u-s/.

Lee, Alan, and Martin Savelsbergh. "Dynamic ridesharing: Is there a role for dedicated drivers?." Transportation Research Part B: Methodological 81 (2015): 483-497.

Hosni, Hadi, Joe Naoum-Sawaya, and Hassan Artail. "The shared-taxi problem: Formulation and solution methods." Transportation Research Part B: Methodological 70 (2014): 303-318.

Li, Baoxiang, et al. "The share-a-ride problem with stochastic travel times and stochastic delivery locations." Transportation Research Part C: Emerging Technologies 67 (2016): 95-108.

Inrix. "Congestion Costs Each American 97 Hours, $1,348 A Year." Inrix, 21 Sept. 2020, inrix.com/press-releases/scorevehicled-2018-us/.

US Census Bureau. "Transportation Modes of U.S. Workers Commuting to Work in 2017." Statista, Statista Inc., 23 Oct 2018, https://www.statista.com/statistics/183907/transportation-modes-used-to-commute-to-work-in-the-us/

A.T. Kearney. "Breakdown of U.S. Transportation Costs in 2018, by Mode of Transportation (in Billion U.S. Dollars)." Statista, Statista Inc., 18 Jun 2019, https://www.statista.com/statistics/645322/us-logistics-market-transportation-costs-breakdown/

A.T. Kearney. "Breakdown of U.S. Transportation Costs in 2018, by Mode of Transportation (in Billion U.S. Dollars)." Statista, Statista Inc., 18 Jun 2019, https://www.statista.com/statistics/645322/us-logistics-market-transportation-costs-breakdown/.

“Number of U.S. Aircraft, Vehicles, Vessels, and Other Conveyances.” Number of U.S. Aircraft, Vehicles, Vessels, and Other Conveyances | Bureau of Transportation Statistics, 2020 www.bts.gov/content/number-us-aircraft-vehicles-vessels-and-other-conveyances.

Lin, Yeqian, et al. "Research on optimization of vehicle routing problem for ride-sharing taxi." Procedia-Social and Behavioral Sciences 43 (2012): 494-502.

Di Febbraro, A., E. Gattorna, and N. Sacco. "Optimization of dynamic ridesharing systems." Transportation research record 2359.1 (2013): 44-50.

Chen, Rui, and Christos G. Cassandras. "Optimization of ride sharing systems using event-driven receding horizon control." arXiv preprint arXiv:1901.01919 (2019).

Cici, Blerim, Athina Markopoulou, and Nikolaos Laoutaris. "Designing an on-line ride-sharing system." Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems. 2015.

Goel, Preeti, Lars Kulik, and Kotagiri Ramamohanarao. "Privacy-aware dynamic ride sharing." ACM Transactions on Spatial Algorithms and Systems (TSAS) 2.1 (2016): 1-41.

Kleiner, Alexander, Bernhard Nebel, and V. Ziparo. "A mechanism for dynamic ride sharing based on parallel auctions." (2011): 266-272.

Bei, Xiaohui, and Shengyu Zhang. "Algorithms for Trip-Vehicle Assignment in Ride-Sharing." AAAI. Vol. 18. 2018.

Santos, Douglas Oliveira, and Eduardo Candido Xavier. "Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem." Twenty-Third International Joint Conference on Artificial Intelligence. 2013.

Conner-Simons, Adam. "How ride-sharing can improve traffic, save money, and help the environment." (2017).

Bost, Callie. “Drivers Are Now Picking Up Hitchhikers To Avoid High Tolls At The George Washington Bridge.” Business Insider, Business Insider, 13 June 2012, https://www.businessinsider.com/drivers-use-vehiclepooling-to-avoid-high-toll-fees-on-the-george-washington-bridge-2012-6.

Caulfield, Brian. "Estimating the environmental benefits of ride-sharing: A case study of Dublin." Transportation Research Part D: Transport and Environment 14.7 (2009): 527-531.

Agatz, Niels, et al. "Dynamic ride-sharing: A simulation study in metro Atlanta." Procedia-Social and Behavioral Sciences 17 (2011): 532-550.

Qian, Xinwu, et al. "Optimal assignment and incentive design in the taxi group ride problem." Transportation Research Part B: Methodological 103 (2017): 208-226.

Santi, Paolo, et al. "Quantifying the benefits of vehicle pooling with shareability networks." Proceedings of the National Academy of Sciences 111.37 (2014): 13290-13294.

Lokhandwala, Mustafa, and Hua Cai. "Dynamic ride sharing using traditional taxis and shared autonomous taxis: A case study of NYC." Transportation Research Part C: Emerging Technologies 97 (2018): 45-60.

Cai, Hua, et al. "Environmental benefits of taxi ride sharing in Beijing." Energy 174 (2019): 503-508.

Amey, Andrew. "A proposed methodology for estimating rideshare viability within an organization, applied to the mit community." TRB Annual Meeting Procediings. 2011.

Sun, Yanshuo, and Lei Zhang. "Potential of Taxi-Pooling to Reduce Vehicle Miles Traveled in Washington, DC." Transportation Research Record 2672.8 (2018): 775-784.

Rodier, Caroline, Farzad Alemi, and Dylan Smith. "Dynamic ridesharing: Exploration of potential for reduction in vehicle miles traveled." Transportation Research Record 2542.1 (2016): 120-126.

Alexander, Lauren P., and Marta C. González. "Assessing the impact of real-time ridesharing on urban traffic using mobile phone data." Proc. UrbComp (2015): 1-9.

Horn, Mark ET. "Fleet scheduling and dispatching for demand-responsive passenger services." Transportation Research Part C: Emerging Technologies 10.1 (2002): 35-63.

Khademi Zareh, Hassan, et al. "Designing a ride-sharing transportation system for assignment and transfer of passengers to a common destination." Journal of Industrial and Systems Engineering 12.3 (2019): 141-153.

Al Qahtani, Hadeel, et al. "A Goal Programming Approach to Multichoice Multiobjective Stochastic Transportation Problems with Extreme Value Distribution." Advances in Operations Research 2019 (2019).

Chan, Nelson D., and Susan A. Shaheen. "Ridesharing in North America: Past, present, and future." Transport reviews 32.1 (2012): 93-112.

Tamiz, Mehrdad, Dylan Jones, and Carlos Romero. "Goal programming for decision making: An overview of the current state-of-the-art." European Journal of operational research 111.3 (1998): 569-581.

Appendix A: Vehicles' and passengers' paths

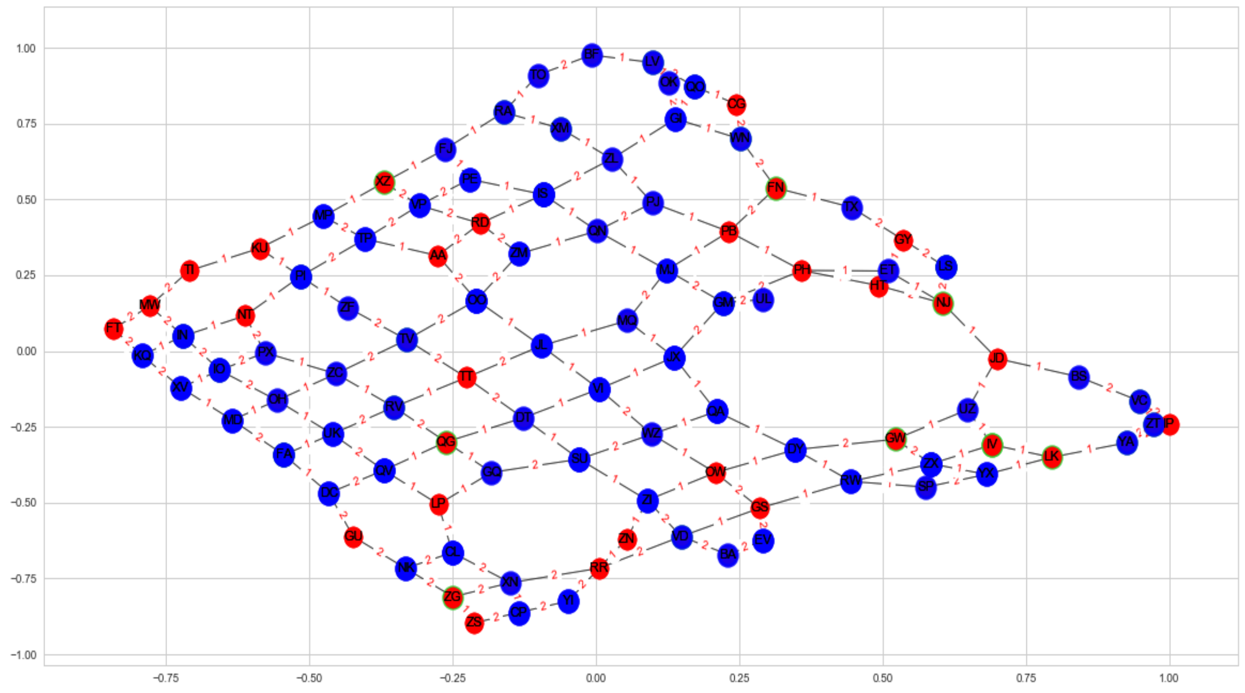
1)

Number of nodes = 10*10

Number of vehicles = 15

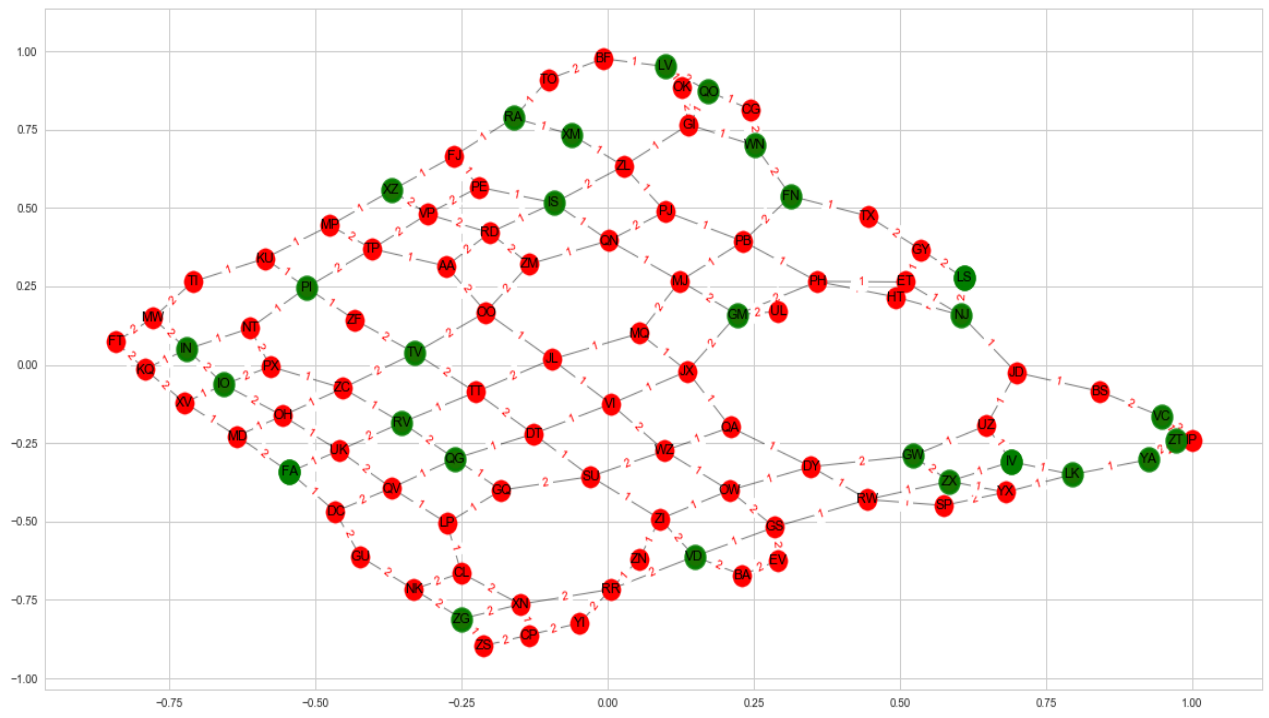
Number of passengers = 20

The blue color is for passengers' paths, green color is for vehicles' paths



*Graph1: Distance graph- 10*10 nodes - 15 vehicles and 20 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly



*Graph3: Distance graph- 10*10 nodes - 15 vehicles and 20 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly

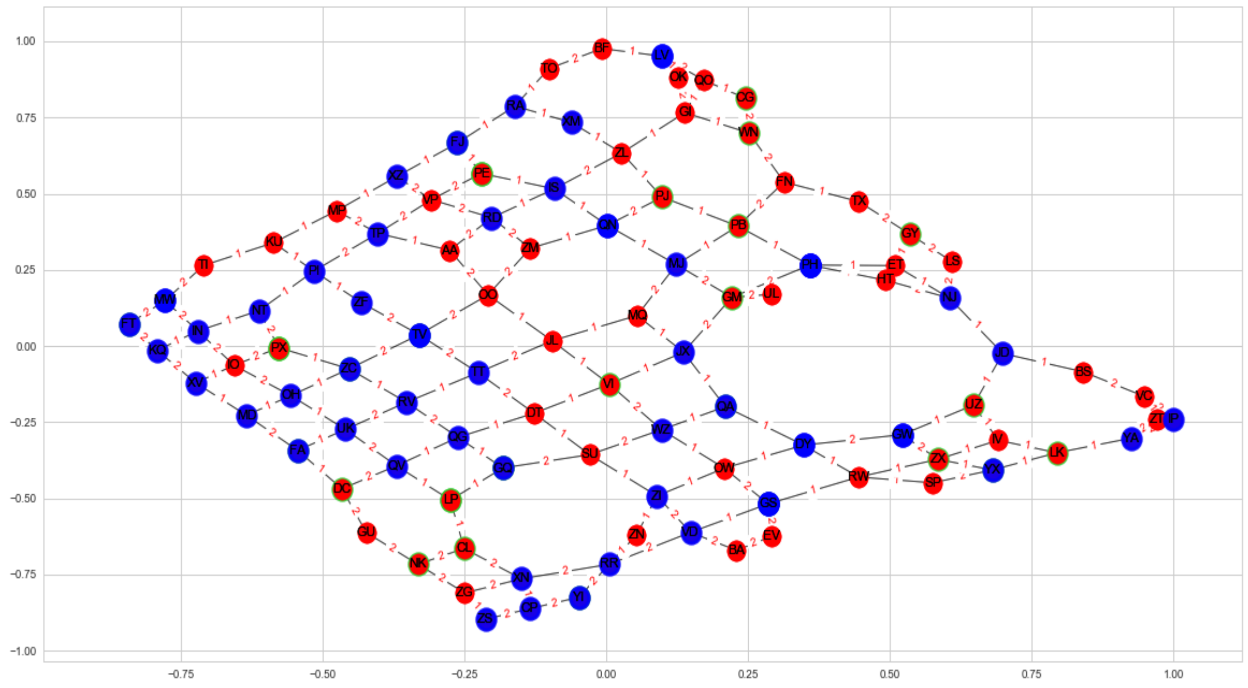
2)

Number of nodes = 10*10

Number of vehicles = 15

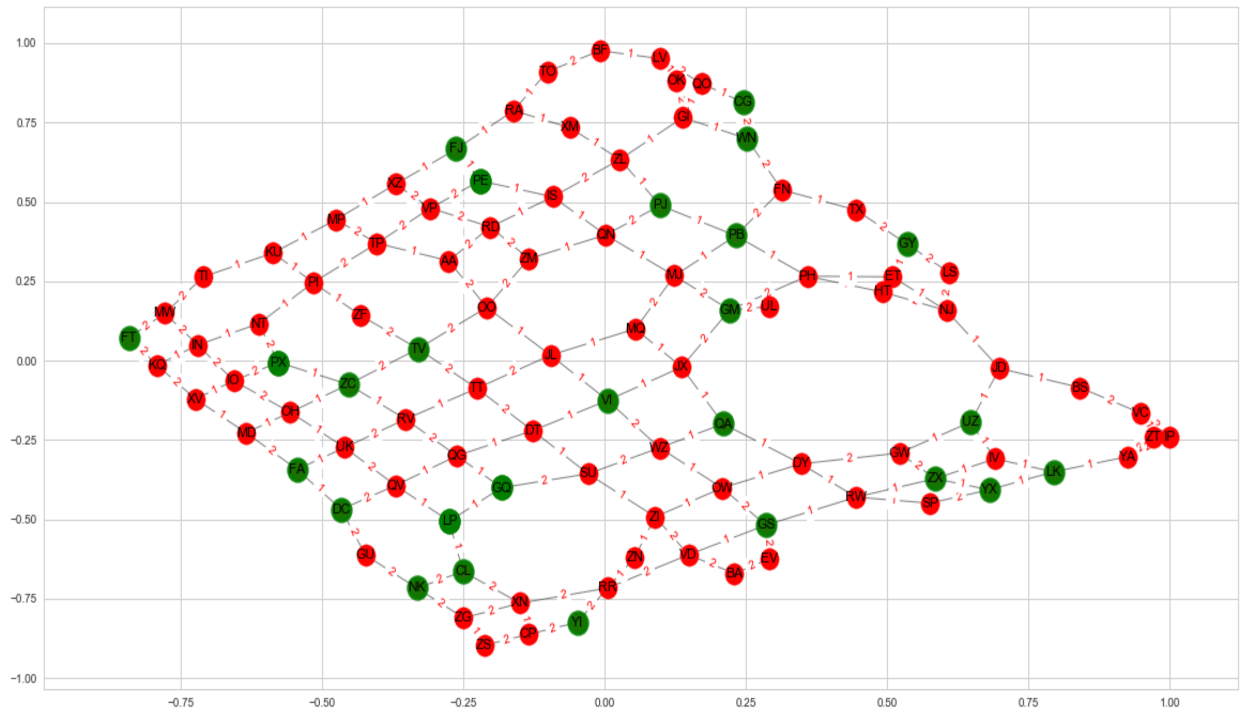
Number of passengers = 30

The blue color is for passengers' paths, green color is for vehicles' paths



*Graph4: Distance graph- 10*10 nodes - 15 vehicles and 30 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly



*Graph 5: Distance graph- 10*10 nodes - 15 vehicles and 30 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly

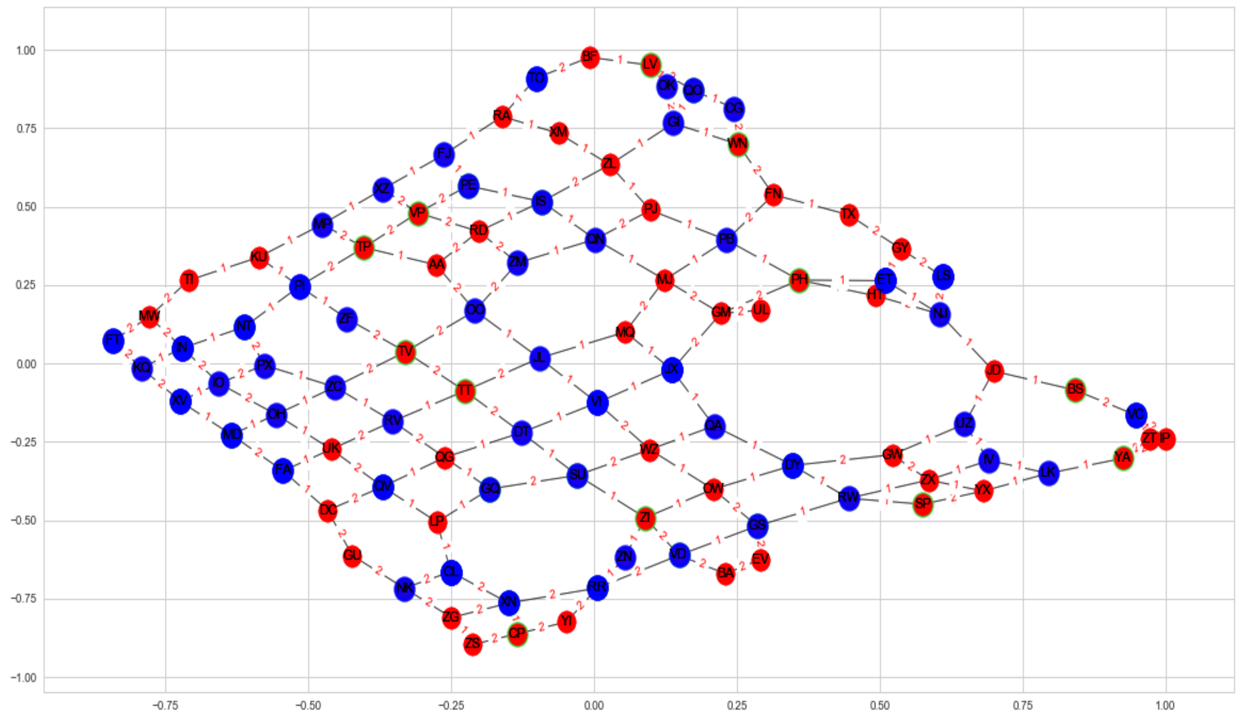
3)

Number of nodes = 10*10

Number of vehicles = 15

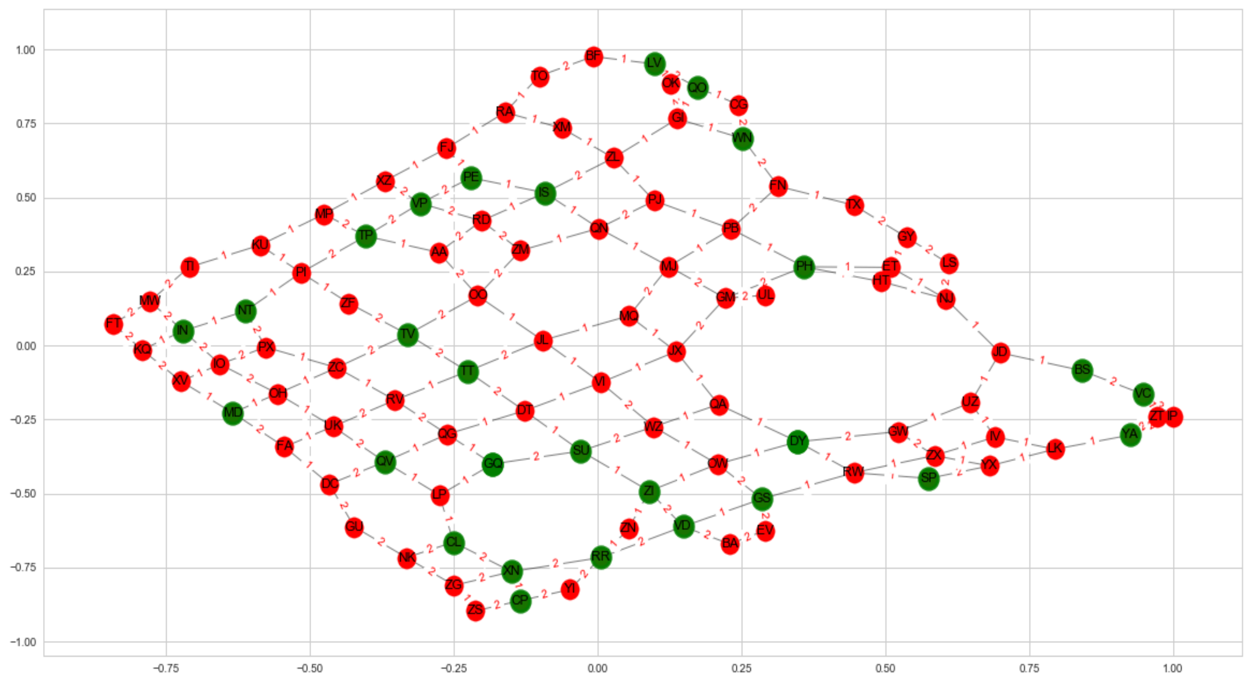
Number of passengers = 40

The blue color is for passengers' paths, green color is for vehicles' paths



*Graph 6: Distance graph- 10*10 nodes - 15 vehicles and 40 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly



*Graph 7: Distance graph- 10*10 nodes - 15 vehicles and 40 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly

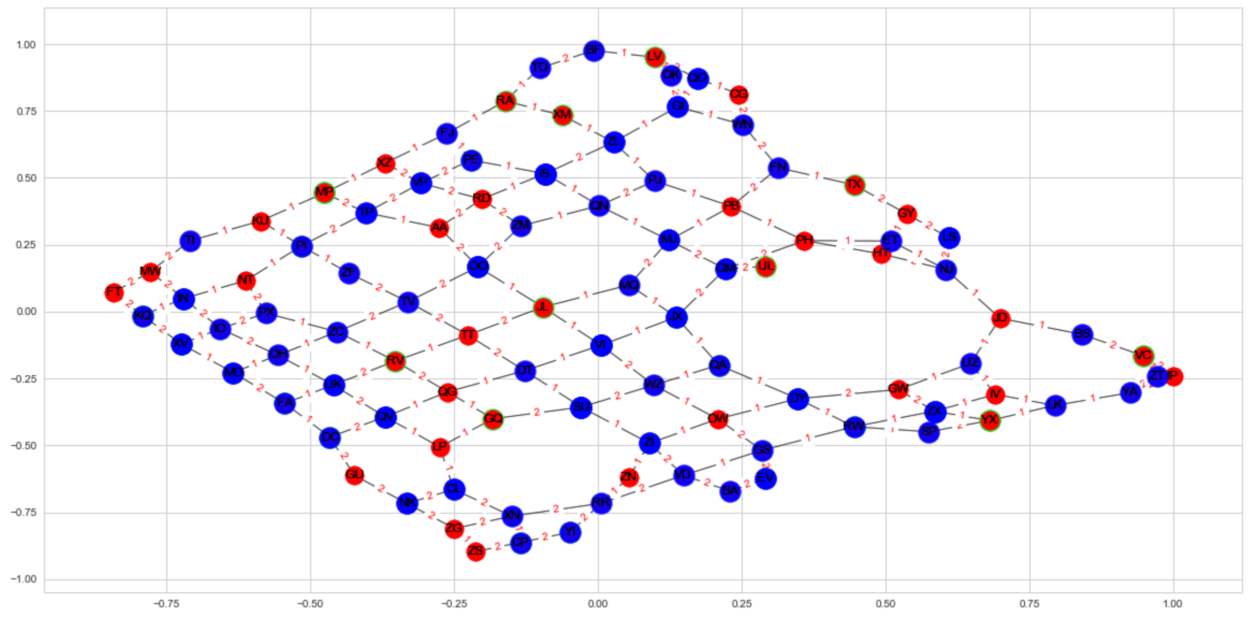
4)

Number of nodes = 10*10

Number of vehicles = 15

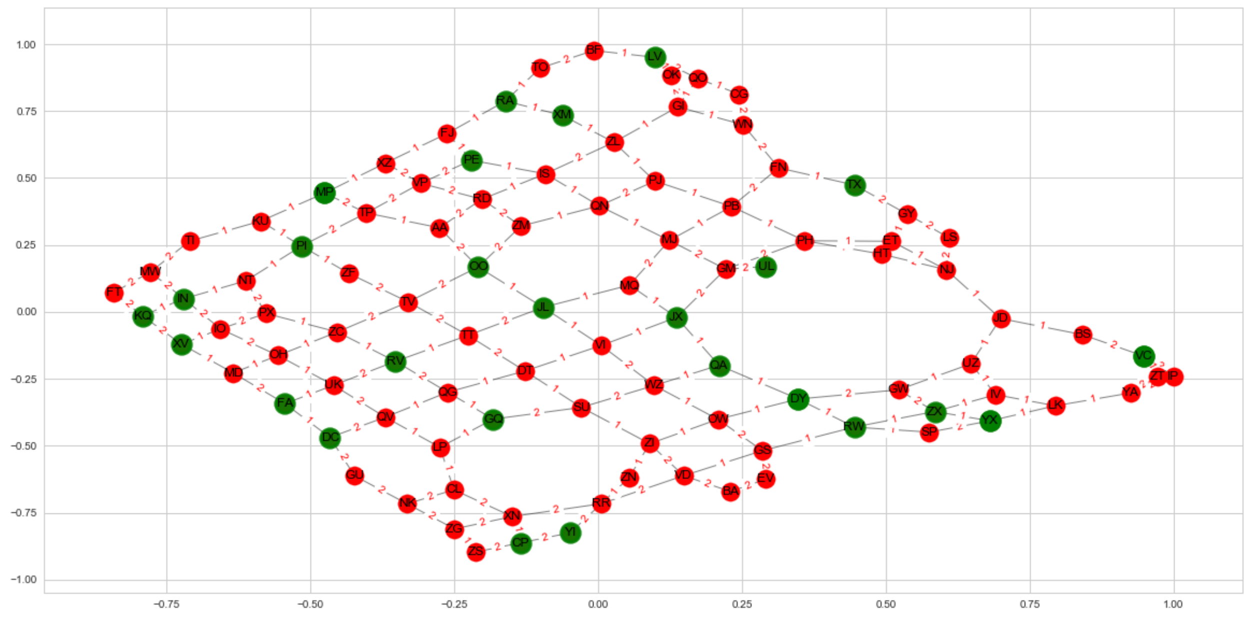
Number of passengers = 50

The blue color is for passengers' paths, green color is for vehicles' paths



*Graph 8: Distance graph- 10*10 nodes - 15 vehicles and 50 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly



*Graph 9: Distance graph- 10*10 nodes - 15 vehicles and 50 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly

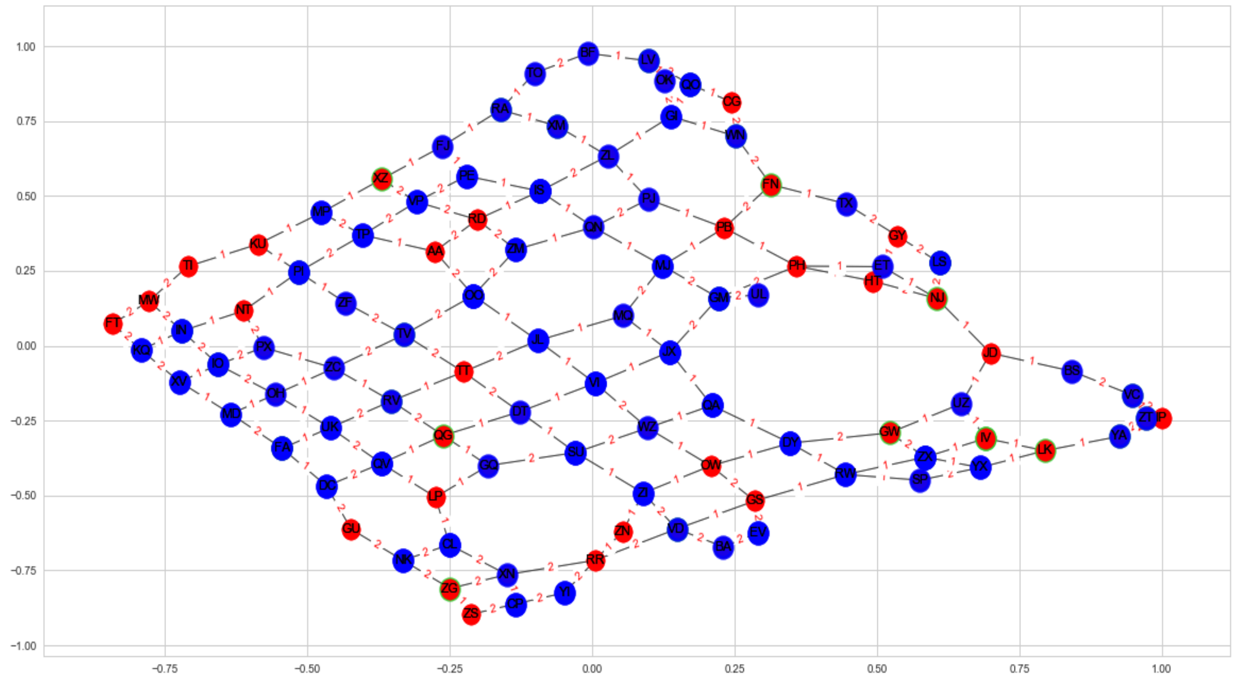
5)

Number of nodes = 10*10

Number of vehicles = 15

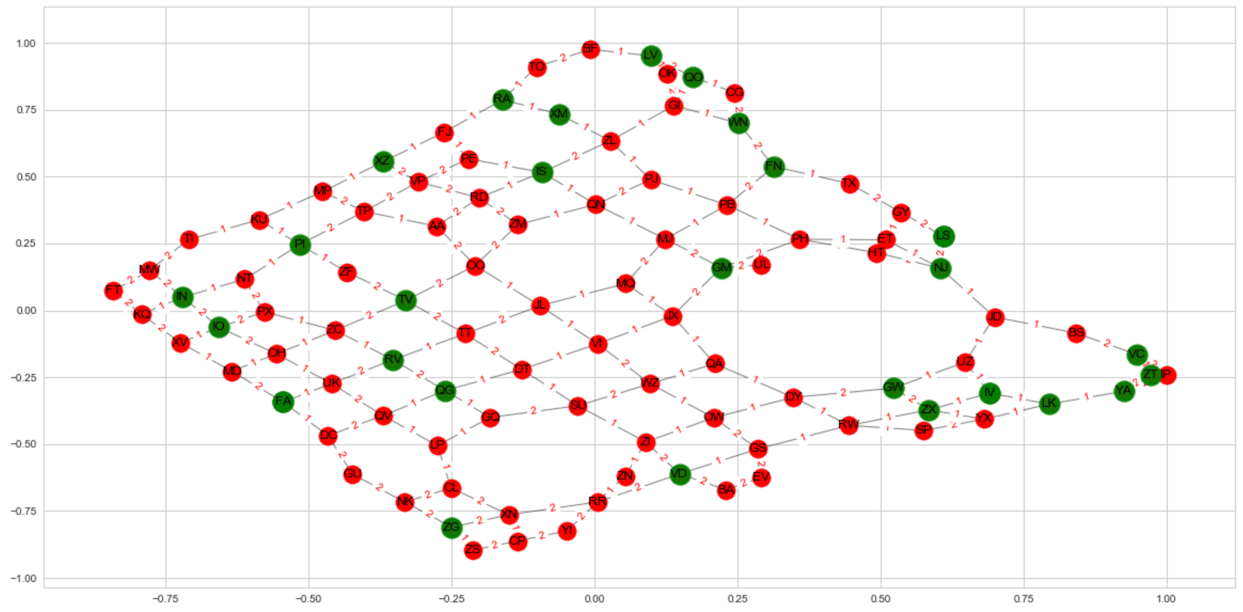
Number of passengers = 60

The blue color is for passengers' paths, green color is for vehicles' paths



*Graph 10: Distance graph- 10*10 nodes - 15 vehicles and 60 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly



*Graph 11: Distance graph- 10*10 nodes - 15 vehicles and 60 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly

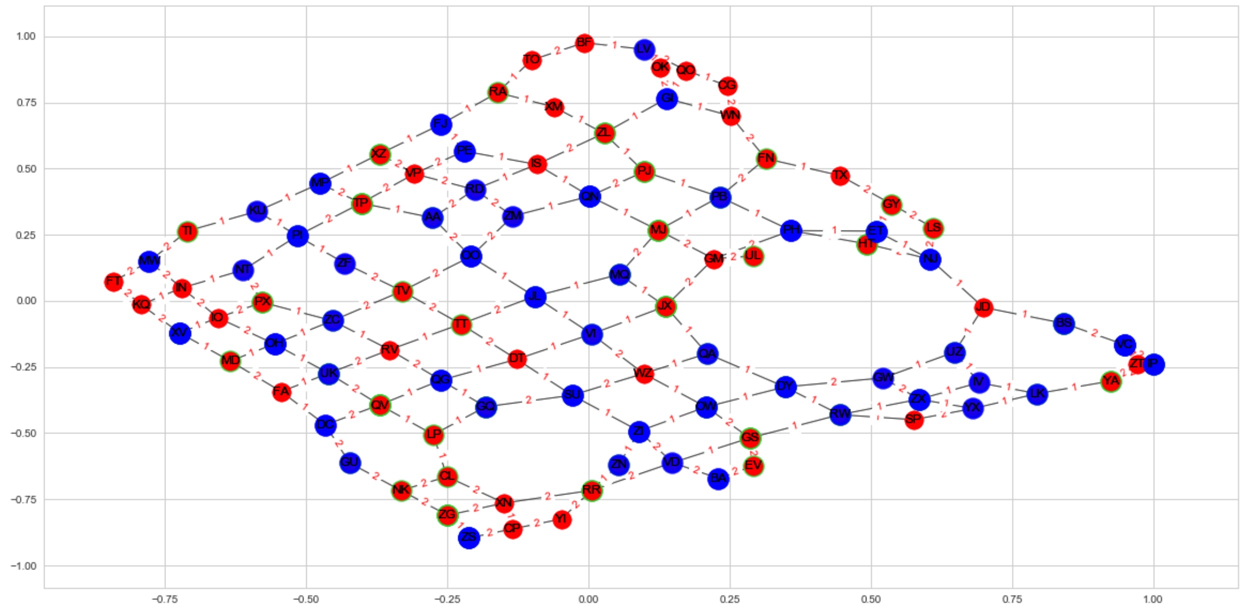
6)

Number of nodes = 10*10

Number of vehicles = 50

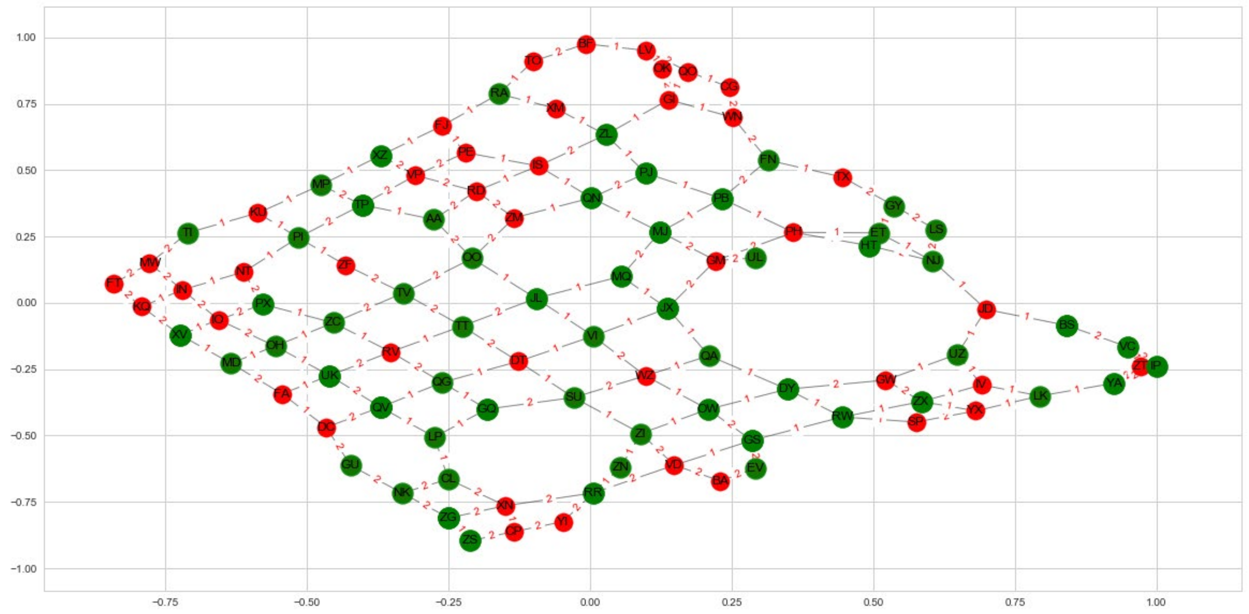
Number of passengers = 40

The blue color is for passengers’ paths, green color is for vehicles’ paths



*Graph 12: Distance graph- 10*10 nodes – 50 vehicles and 40 passengers*

The edges’ weights are not based on the length of the edges; the weights are assigned to the edges randomly


Graph 13: Distance graph- 10*10 nodes - 50 vehicles and 40 passengers

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly

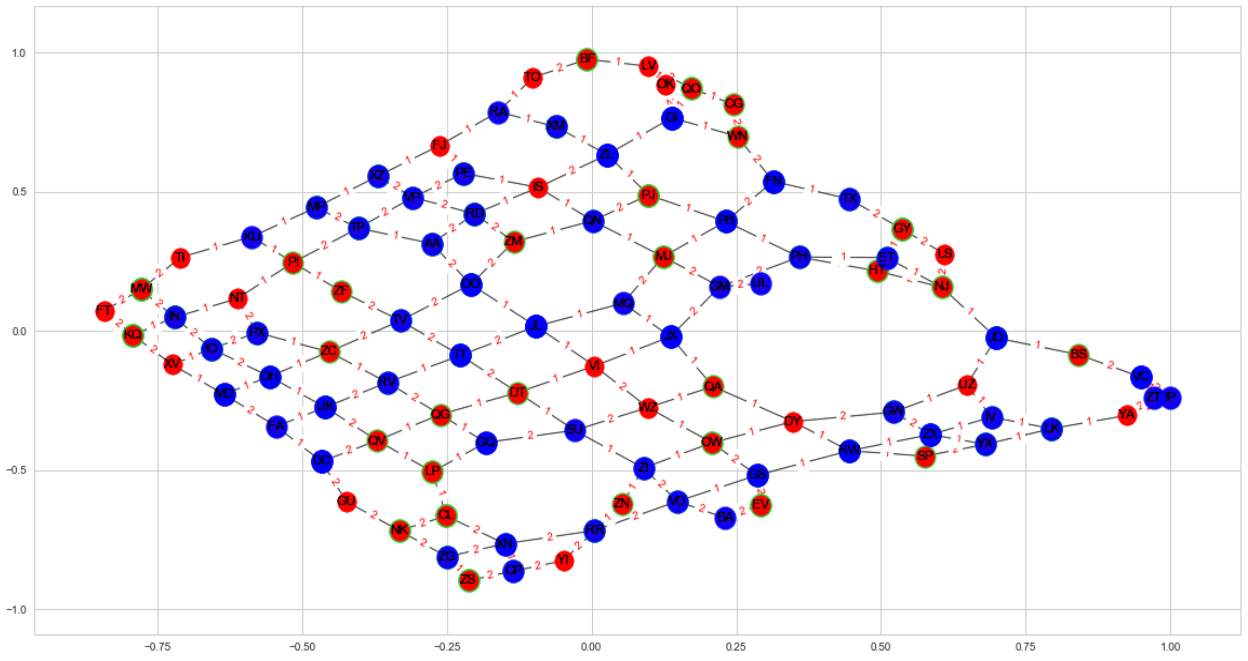
7)

Number of nodes = 10*10

Number of vehicles = 60

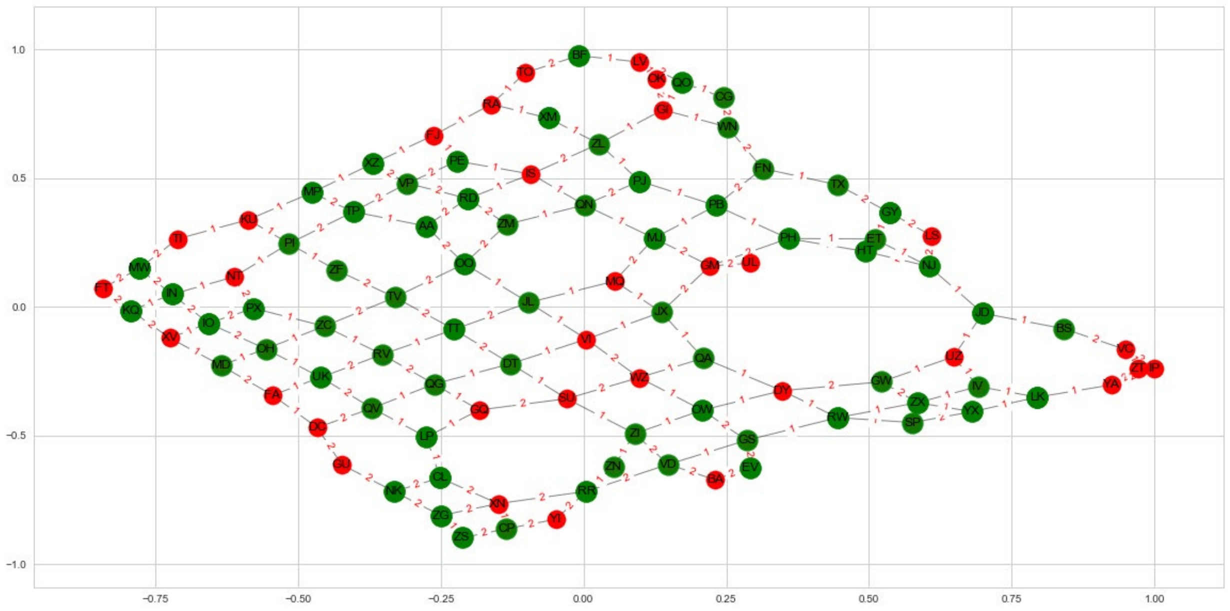
Number of passengers = 40

The blue color is for passengers' paths, green color is for vehicles' paths



*Graph 14: Distance graph- 10*10 nodes - 60 vehicles and 40 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly



*Graph 15: Distance graph- 10*10 nodes - 60 vehicles and 40 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly

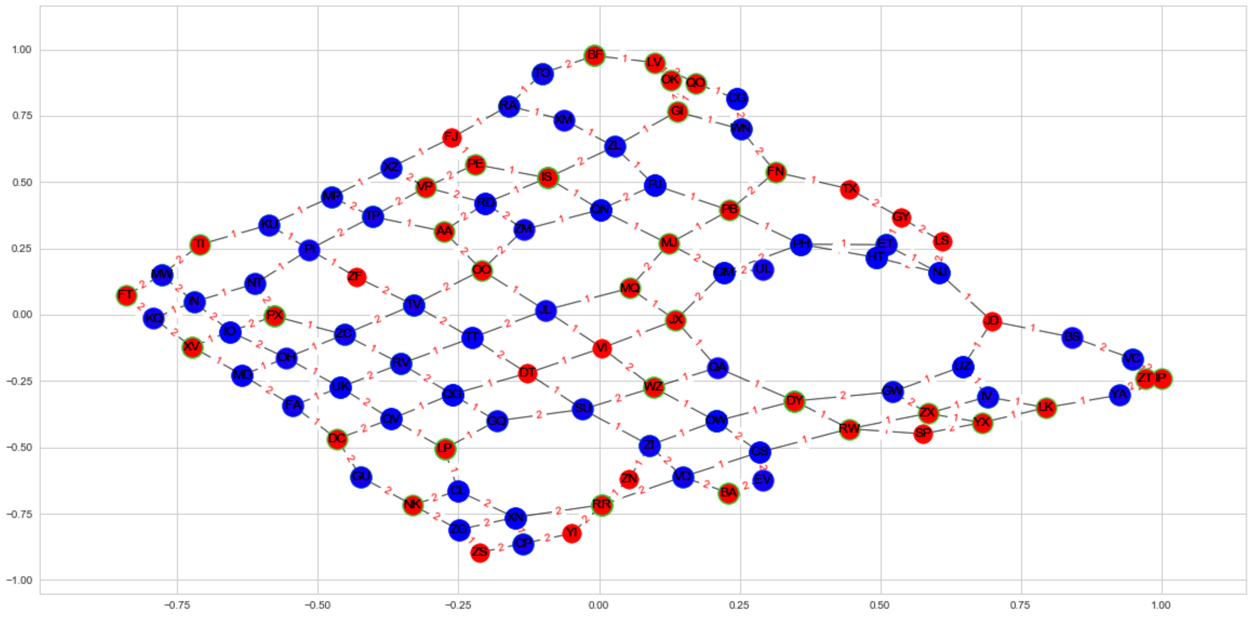
8)

Number of nodes = 10*10

Number of vehicles = 70

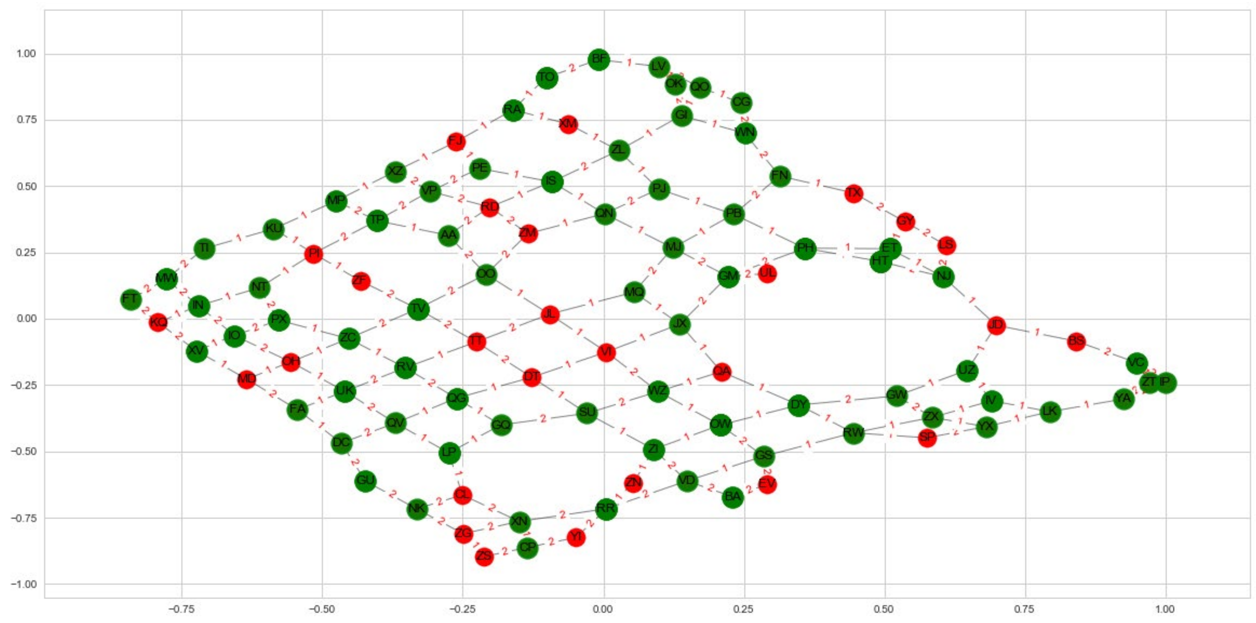
Number of passengers = 40

The blue color is for passengers' paths, green color is for vehicles' paths



*Graph 16: Distance graph- 10*10 nodes - 70 vehicles and 40 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly



*Graph 17: Distance graph- 10*10 nodes - 70 vehicles and 40 passengers*

The edges’ weights are not based on the length of the edges; the weights are assigned to the edges randomly

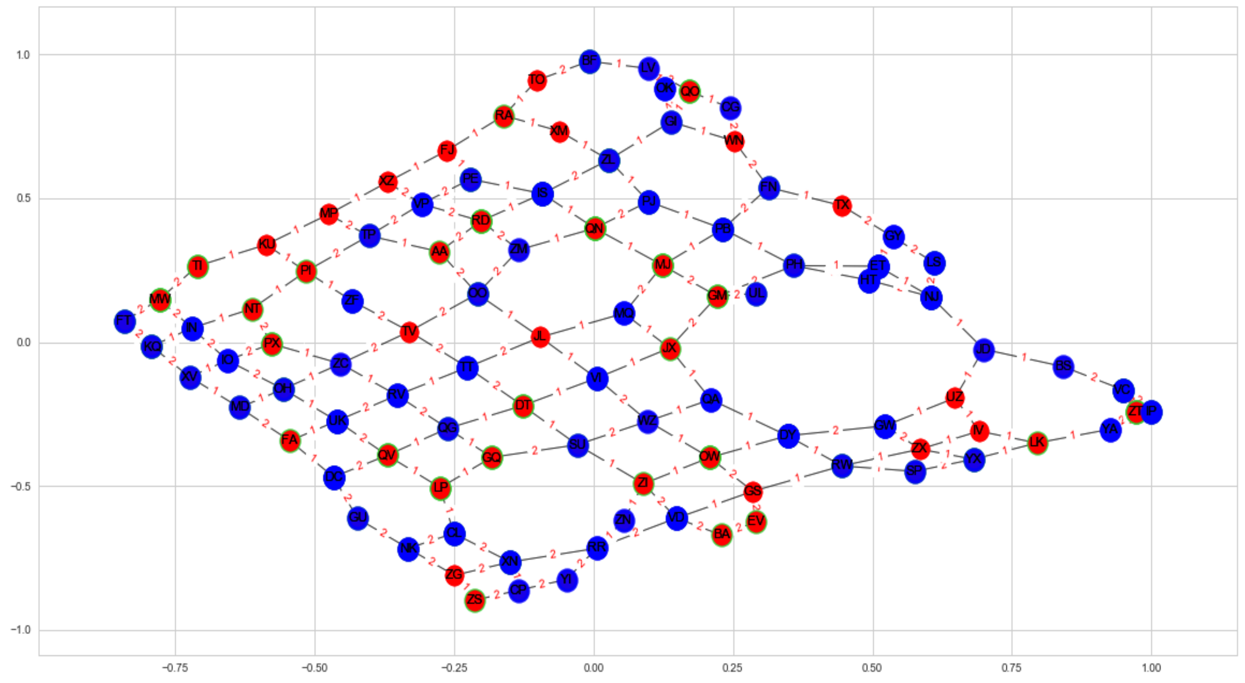
9)

Number of nodes = 10*10

Number of vehicles = 60

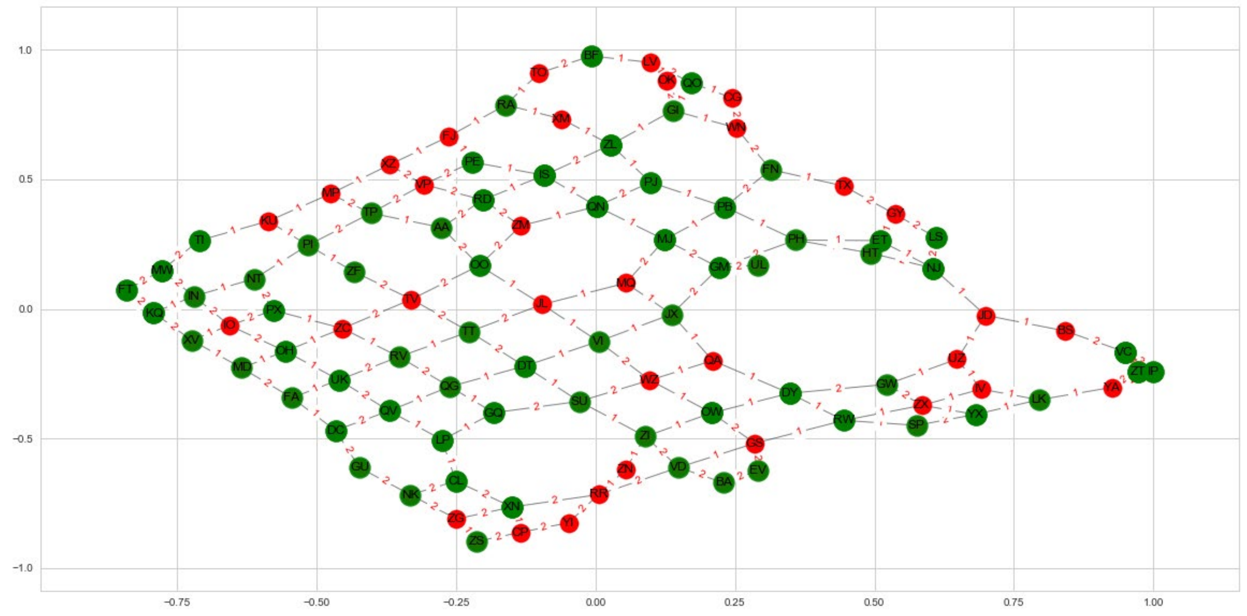
Number of passengers = 60

The blue color is for passengers' paths, green color is for vehicles' paths



*Graph 18: Distance graph- 10*10 nodes - 60 vehicles and 60 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly


*Graph 19: Distance graph- 10*10 nodes - 60 vehicles and 60 passengers*

The edges’ weights are not based on the length of the edges; the weights are assigned to the edges randomly

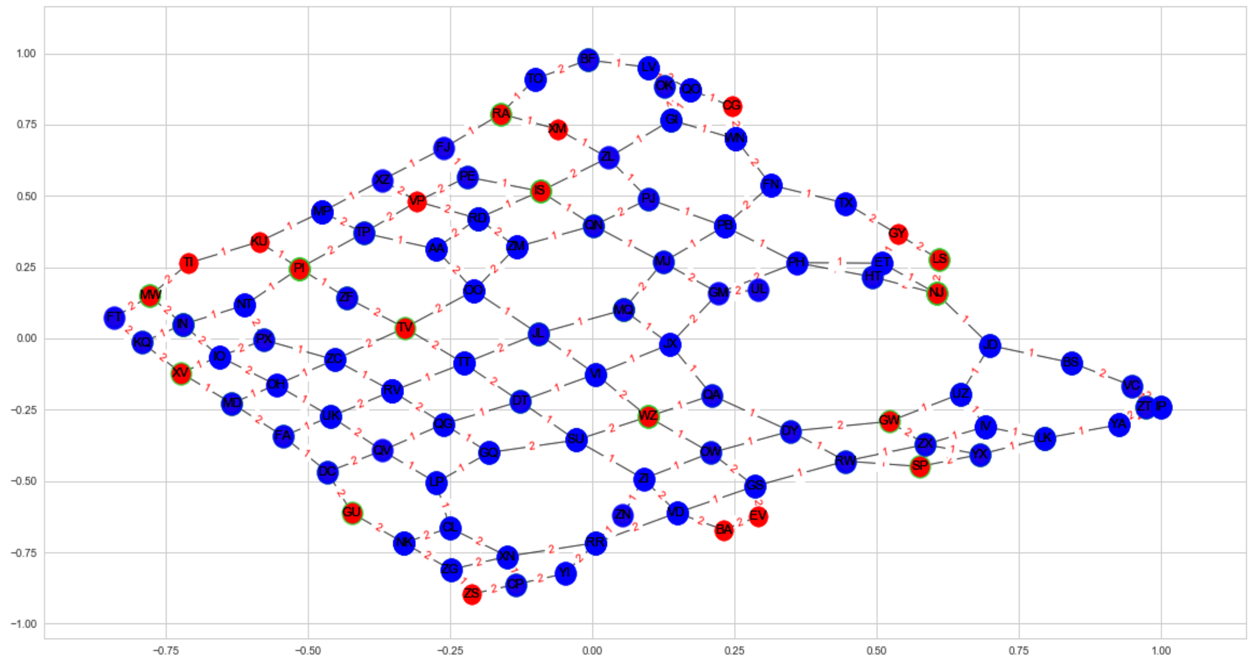
10)

Number of nodes = 10*10

Number of vehicles = 60

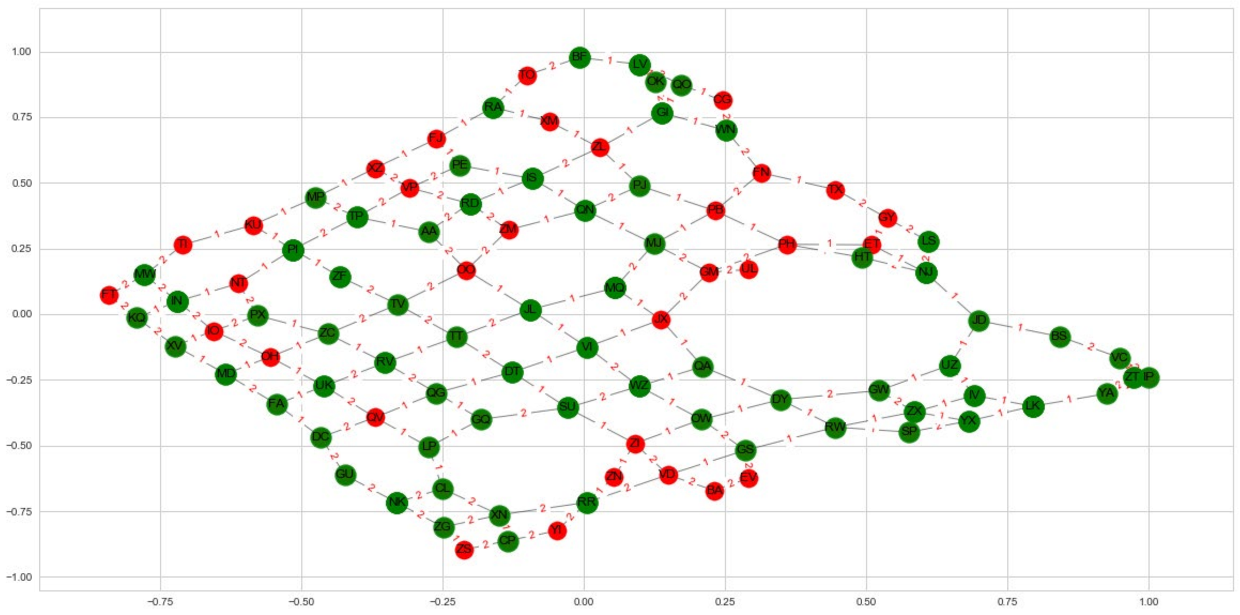
Number of passengers = 80

The blue color is for passengers' paths, green color is for vehicles' paths



*Graph 20: Distance graph- 10*10 nodes - 60 vehicles and 80 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly



*Graph 21: Distance graph- 10*10 nodes - 60 vehicles and 80 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly

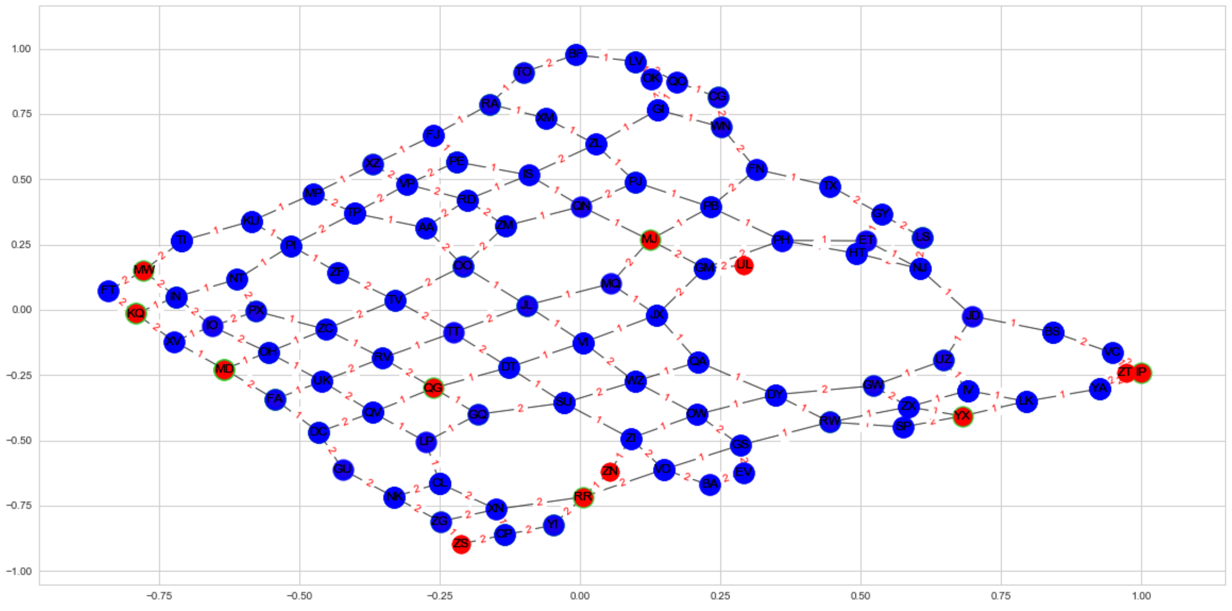
11)

Number of nodes = 10*10

Number of vehicles = 60

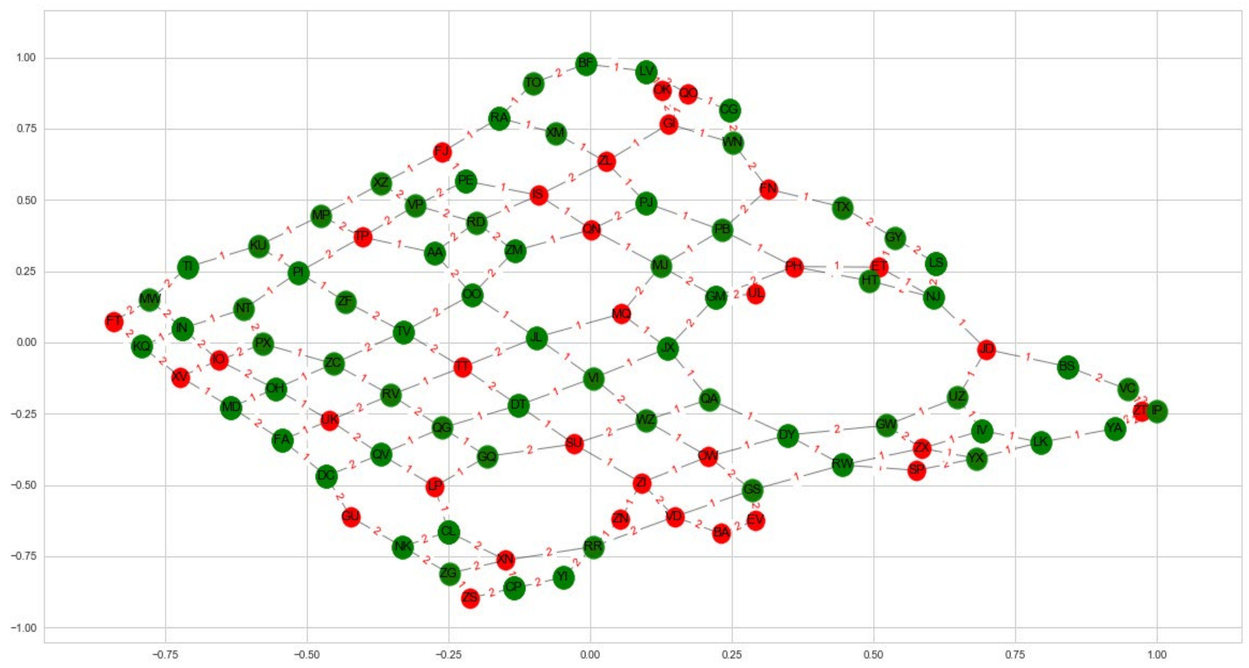
Number of passengers = 100

The blue color is for passengers' paths, green color is for vehicles' paths



Graph 22: Distance graph- 10*10 nodes - 60 vehicles and 100 passengers

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly



*Graph 23: Distance graph- 10*10 nodes - 60 vehicles and 100 passengers*

The edges’ weights are not based on the length of the edges; the weights are assigned to the edges randomly

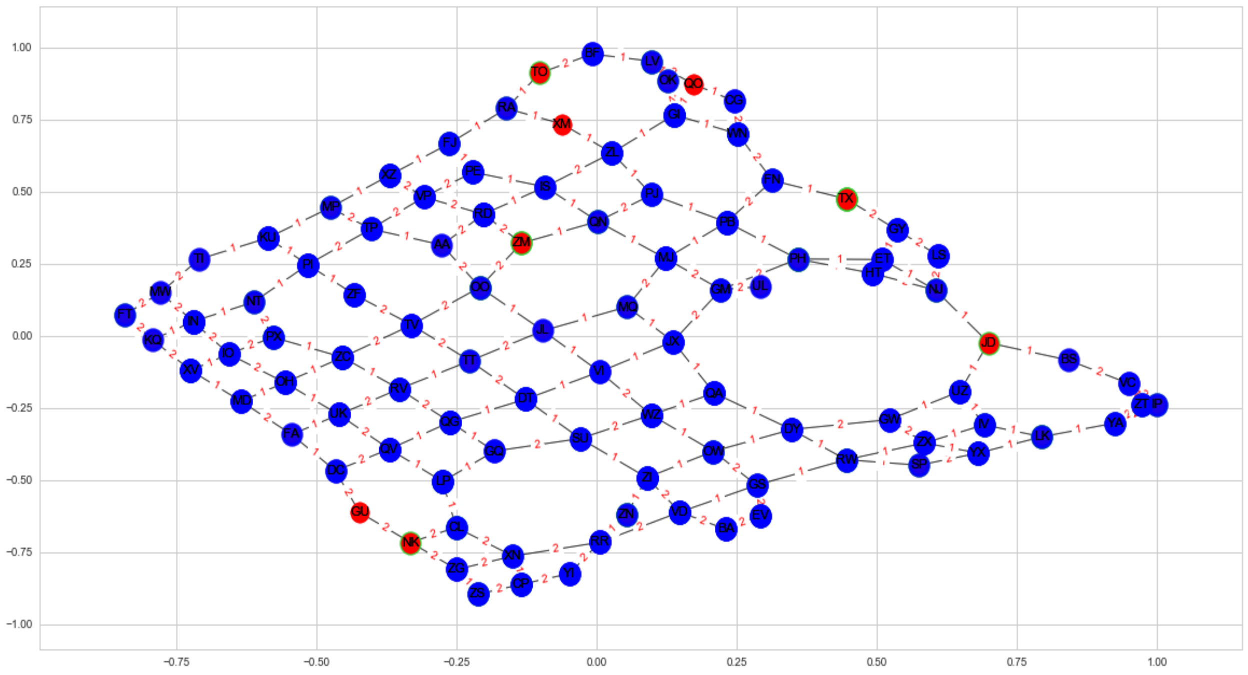
12)

Number of nodes = 10*10

Number of vehicles = 60

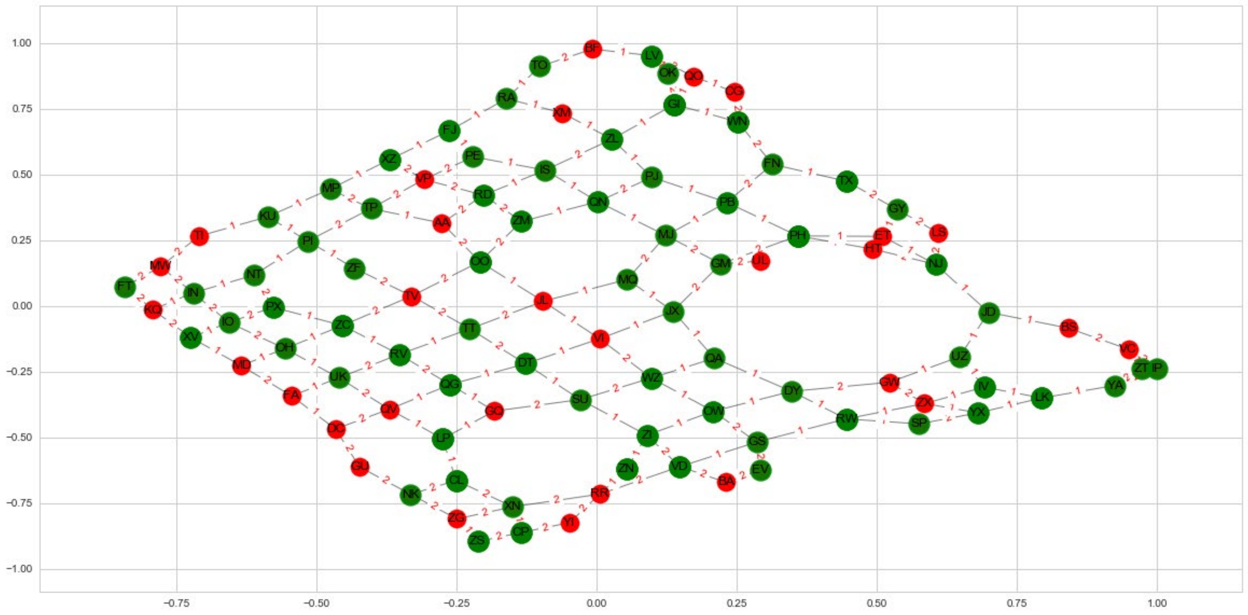
Number of passengers = 120

The blue color is for passengers' paths, green color is for vehicles' paths



Graph 24: Distance graph- 10*10 nodes - 60 vehicles and 120 passengers

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly



*Graph 25: Distance graph- 10*10 nodes - 60 vehicles and 120 passengers*

The edges' weights are not based on the length of the edges; the weights are assigned to the edges randomly

Appendix B: Tables

1)

	System pe	BR-D	BR-T	AR- Distance	AR-Time
1	D=10,T=0	302	572	299	728
2	D=9,T= 1	302	572	304	688
3	D=8,T=2	302	572	309	657
4	D=7,T=3	302	572	317	632
5	D=6,T=4	302	572	320	626
6	D=5,T=5	302	572	324	620
7	D=4,T=6	302	572	332	612
8	D=3,T=7	302	572	332	612
9	D=2,T=8	302	572	338	610
10	D=1,T=9	302	572	339	603
11	D=0,T=10	302	572	339	603

Table 1: The results of multi-objective optimization for 15 vehicles and 20 passengers

2)

	System pe	BR-D	BR-T	AR- Distance	AR-Time
1	D=10,T=0	398	714	340	910
2	D=9,T= 1	398	714	342	836
3	D=8,T=2	398	714	344	826
4	D=7,T=3	398	714	349	811
5	D=6,T=4	398	714	349	811
6	D=5,T=5	398	714	354	805
7	D=4,T=6	398	714	359	803
8	D=3,T=7	398	714	361	802
9	D=2,T=8	398	714	361	802
10	D=1,T=9	398	714	365	801
11	D=0,T=10	398	714	365	801

Table 2: The results of multi-objective optimization for 15 vehicles and 30 passengers

3)

	System pe	BR-D	BR-T	AR- Distar	AR-Time
1	D=10,T=0	496	931	423	1184
2	D=9,T= 1	496	931	445	1121
3	D=8,T=2	496	931	452	1079
4	D=7,T=3	496	931	455	1069
5	D=6,T=4	496	931	461	1064
6	D=5,T=5	496	931	461	1064
7	D=4,T=6	496	931	465	1060
8	D=3,T=7	496	931	465	1060
9	D=2,T=8	496	931	465	1060
10	D=1,T=9	496	931	465	1060
11	D=0,T=10	496	931	465	1060

Table 3: The results of multi-objective optimization for 15 vehicles and 40 passengers

4)

	System pe	BR-D	BR-T	AR- Distar	AR-Time
1	D=10,T=0	604	1104	497	1340
2	D=9,T= 1	604	1104	498	1317
3	D=8,T=2	604	1104	504	1280
4	D=7,T=3	604	1104	512	1244
5	D=6,T=4	604	1104	512	1231
6	D=5,T=5	604	1104	513	1225
7	D=4,T=6	604	1104	547	1222
8	D=3,T=7	604	1104	547	1222
9	D=2,T=8	604	1104	547	1222
10	D=1,T=9	604	1104	547	1222
11	D=0,T=10	604	1104	547	1222

Table 4: The results of multi-objective optimization for 15 vehicles and 50 passengers

5)

	System pe	BR-D	BR-T	AR- Distar	AR-Time
1	D=10,T=0	634	1216	569	1512
2	D=9,T= 1	634	1216	570	1500
3	D=8,T=2	634	1216	571	1489
4	D=7,T=3	634	1216	585	1448
5	D=6,T=4	634	1216	590	1447
6	D=5,T=5	634	1216	593	1444
7	D=4,T=6	634	1216	603	1438
8	D=3,T=7	634	1216	607	1436
9	D=2,T=8	634	1216	607	1436
10	D=1,T=9	634	1216	611	1435
11	D=0,T=10	634	1216	611	1435

Table 5: The results of multi-objective optimization for 15 vehicles and 60 passengers

6)

	System pe	BR-D	BR-T	AR- Distar	AR-Time
1	D=10,T=0	772	1393	671	1755
2	D=9,T= 1	772	1393	676	1702
3	D=8,T=2	772	1393	682	1602
4	D=7,T=3	772	1393	682	1602
5	D=6,T=4	772	1393	683	1600
6	D=5,T=5	772	1393	683	1600
7	D=4,T=6	772	1393	692	1583
8	D=3,T=7	772	1393	694	1582
9	D=2,T=8	772	1393	697	1581
10	D=1,T=9	772	1393	697	1581
11	D=0,T=10	772	1393	697	1581

Table 6: The results of multi-objective optimization for 50 vehicles and 40 passengers

7)

	System pe	BR-D	BR-T	AR- Distar	AR-Time
1	D=10,T=0	945	1710	806	2139
2	D=9,T= 1	945	1710	808	2110
3	D=8,T=2	945	1710	809	2105
4	D=7,T=3	945	1710	821	2075
5	D=6,T=4	945	1710	826	2065
6	D=5,T=5	945	1710	830	2059
7	D=4,T=6	945	1710	840	2033
8	D=3,T=7	945	1710	840	2033
9	D=2,T=8	945	1710	840	2033
10	D=1,T=9	945	1710	840	2033
11	D=0,T=10	945	1710	840	2033

Table 7: The results of multi-objective optimization for 60 vehicles and 40 passengers

8)

	System pe	BR-D	BR-T	AR- Distar	AR-Time
1	D=10,T=0	935	1751	844	2118
2	D=9,T= 1	935	1751	845	2103
3	D=8,T=2	935	1751	846	2058
4	D=7,T=3	935	1751	873	2019
5	D=6,T=4	935	1751	876	2006
6	D=5,T=5	935	1751	882	2000
7	D=4,T=6	935	1751	892	2000
8	D=3,T=7	935	1751	895	2000
9	D=2,T=8	935	1751	908	2000
10	D=1,T=9	935	1751	908	2000
11	D=0,T=10	935	1751	908	2000

Table 8: The results of multi-objective optimization for 70 vehicles and 40 passengers

9)

	System pe	BR-D	BR-T	AR- Distar	AR-Time
1	D=10,T=0	1003	1846	860	2303
2	D=9,T= 1	1003	1846	861	2226
3	D=8,T=2	1003	1846	867	2192
4	D=7,T=3	1003	1846	867	2191
5	D=6,T=4	1003	1846	871	2183
6	D=5,T=5	1003	1846	871	2183
7	D=4,T=6	1003	1846	876	2178
8	D=3,T=7	1003	1846	876	2178
9	D=2,T=8	1003	1846	879	2177
10	D=1,T=9	1003	1846	883	2176
11	D=0,T=10	1003	1846	883	2176

Table 9: The results of multi-objective optimization for 60 vehicles and 60 passengers

10)

	System pe	BR-D	BR-T	AR- Distar	AR-Time
1	D=10,T=0	1249	2326	1066	2981
2	D=9,T= 1	1249	2326	1067	2979
3	D=8,T=2	1249	2326	1079	2874
4	D=7,T=3	1249	2326	1088	2869
5	D=6,T=4	1249	2326	1088	2850
6	D=5,T=5	1249	2326	1095	2841
7	D=4,T=6	1249	2326	1100	2839
8	D=3,T=7	1249	2326	1100	2839
9	D=2,T=8	1249	2326	1101	2839
10	D=1,T=9	1249	2326	1102	2838
11	D=0,T=10	1249	2326	1102	2838

Table 10: The results of multi-objective optimization for 60 vehicles and 80 passengers

11)

	System pe	BR-D	BR-T	AR- Distar	AR-Time
1	D=10,T=0	1425	2658	1077	3383
2	D=9,T= 1	1425	2658	1089	3322
3	D=8,T=2	1425	2658	1106	3289
4	D=7,T=3	1425	2658	1112	3280
5	D=6,T=4	1425	2658	1127	3279
6	D=5,T=5	1425	2658	1135	3267
7	D=4,T=6	1425	2658	1150	3239
8	D=3,T=7	1425	2658	1156	3235
9	D=2,T=8	1425	2658	1163	3233
10	D=1,T=9	1425	2658	1163	3233
11	D=0,T=10	1425	2658	1163	3233

Table 11: The results of multi-objective optimization for 60 vehicles and 100 passengers

12)

	System pe	BR-D	BR-T	AR- Distar	AR-Time
1	D=10,T=0	1599	2978	1320	3758
2	D=9,T= 1	1599	2978	1323	3682
3	D=8,T=2	1599	2978	1325	3677
4	D=7,T=3	1599	2978	1329	3606
5	D=6,T=4	1599	2978	1335	3594
6	D=5,T=5	1599	2978	1335	3594
7	D=4,T=6	1599	2978	1353	3557
8	D=3,T=7	1599	2978	1367	3527
9	D=2,T=8	1599	2978	1370	3526
10	D=1,T=9	1599	2978	1370	3526
11	D=0,T=10	1599	2978	1370	3526

Table 12: The results of multi-objective optimization for 60 vehicles and 120 passengers

Appendix C: Coding

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 23 16:01:06 2019

@author: monasrazadani
"""
import string
import random
from collections import defaultdict
import pandas as pd
import seaborn as sns
import networkx as nx
import matplotlib.pyplot as plt
import math
import numpy as np



random.seed(1)


class Graph():
    def __init__(self):


        self.edges = defaultdict(list)
        self.weights = {}
        self.time = {}
    "This function creates edges"
    def add_edge(self, from_node, to_node, weight, time):

        self.edges[from_node].append(to_node)
        self.edges[to_node].append(from_node)
        self.weights[(from_node, to_node)] = weight
        self.weights[(to_node, from_node)] = weight
        self.times[(from_node, to_node)] = time
        self.times[(to_node, from_node)] = time


"This function finds shortest path"
def dijsktra(graph, initial, end):

    shortest_paths = {initial: (None, 0)}
    current_node = initial
    visited = set()

    while current_node != end:
        visited.add(current_node)
        destinations = graph.edges[current_node]
        weight_to_current_node = shortest_paths[current_node][1]

        for next_node in destinations:
            weight = graph.weights[(current_node, next_node)] + weight_to_current_node

            if next_node not in shortest_paths:
                shortest_paths[next_node] = (current_node, weight)
            else:
                current_shortest_weight = shortest_paths[next_node][1]

                if current_shortest_weight > weight:
                    shortest_paths[next_node] = (current_node, weight)

        next_destinations = {node: shortest_paths[node] for node in shortest_paths if node not in visited}
        if not next_destinations:
            return -1
        current_node = min(next_destinations, key=lambda k: next_destinations[k][1])

    path = []
    while current_node is not None:
        path.append(current_node)
        next_node = shortest_paths[current_node][0]
        current_node = next_node

    path = path[::-1]
    return path
"This function finds shortest path"
def dijsktra_time(graph, initial, end):

    shortest_paths = {initial: (None, 0)}
    current_node = initial
    visited = set()

    while current_node != end:
        visited.add(current_node)
        destinations = graph.edges[current_node]
        weight_to_current_node = shortest_paths[current_node][1]

        for next_node in destinations:
            weight = graph.times[(current_node, next_node)] + weight_to_current_node

            if next_node not in shortest_paths:
                shortest_paths[next_node] = (current_node, weight)



            else:
                current_shortest_weight = shortest_paths[next_node][1]

                if current_shortest_weight > weight:
                    shortest_paths[next_node] = (current_node, weight)

        next_destinations = {node: shortest_paths[node] for node in shortest_paths if node not in visited}
        if not next_destinations:
            return -1
        current_node = min(next_destinations, key=lambda k: next_destinations[k][1])

    path = []
    while current_node is not None:
        path.append(current_node)
        next_node = shortest_paths[current_node][0]
        current_node = next_node

    path = path[::-1]
    return path
```

```
def id_generator(size=2, chars=string.ascii_uppercase):
    return ''.join(random.choice(chars) for _ in range(size))


"This function finds shortest edges between nodes"
def shortest_path_with_nodes(graph, primary, secondary, metric):


    if len(secondary) == 1:
        if metric == "db":
            shortest_path = dijsktra(graph, primary[0], secondary[0][0])[:-1]+\
                        dijsktra(graph, secondary[0][0], secondary[0][1])[:-1]+\
                        dijsktra(graph, secondary[0][1], primary[1])
        if metric == "tb":
            shortest_path = dijsktra_time(graph, primary[0], secondary[0][0])[:-1]+\
                        dijsktra_time(graph, secondary[0][0], secondary[0][1])[:-1]+\
                        dijsktra_time(graph, secondary[0][1], primary[1])
        if metric == "op":
            shortest_path = path_picker(graph, dijsktra(graph, primary[0], secondary[0][0])[:-1],\
                        dijsktra_time(graph, primary[0], secondary[0][0])[:-1])+\
                        path_picker(graph, dijsktra(graph, secondary[0][0],\
                        secondary[0][1])[:-1], dijsktra_time(graph, secondary[0][0],\
                        secondary[0][1])[:-1])+path_picker(graph, dijsktra(graph,\
                        secondary[0][1], primary[1]), dijsktra_time(graph, secondary[0][1], primary[1]))
    else:
        shortest_path = [primary[0]]
        current_position = primary[0]


        current_options = []
```

```
for x in range(0, len(secondary)):
    current_options.append(secondary[x][0])



while len(current_options) > 0:

    best_weight = 1000000
    for node in current_options:
        if metric == "db":
            d = dijsktra(graph, current_position, node)
        if metric == "tb":
            d = dijsktra_time(graph, current_position, node)
        if metric == "op":
            d = dijsktra(graph, current_position, node)
            t = dijsktra_time(graph, current_position, node)
            d = list(path_picker(graph, d, t))

        route_weight = 0
        for i in range(len(d)-1):
            route_weight += graph.weights[(d[i], d[i + 1])]
        if route_weight < best_weight:
            best_option = node
            best_weight = route_weight
            best_path = d

    shortest_path += best_path[1:]
    current_position = best_option

    if best_option != secondary[current_options.index(best_option)][-1]:
            current_options[current_options.index(best_option)]                                        =
secondary[current_options.index(best_option)][secondary[current_options.index(best_option)].index(b
est_option)+1]

        else:

            secondary.pop(current_options.index(best_option))

            current_options.pop(current_options.index(best_option))

      shortest_path += dijsktra(graph, current_position, primary[1])[1:]


   return shortest_path
"Thid function finds the best path"
def path_picker(graph, distance_path, time_path, tp=8, dp=3):

   if distance_path == time_path:
      return distance_path

   time_path_weight = 0
   time_path_time = 0
   for i in range(len(time_path)-1):
      time_path_weight += graph.weights[(time_path[i], time_path[i + 1])]
      time_path_time += graph.times[(time_path[i], time_path[i + 1])]

   distance_path_weight = 0
   distance_path_time = 0
```

```
    for i in range(len(distance_path)-1):
        distance_path_weight += graph.weights[(distance_path[i], distance_path[i + 1])]
        distance_path_time += graph.times[(distance_path[i], distance_path[i + 1])]

    time_path_penalty = time_path_weight * dp + time_path_time * tp
    distance_path_penalty = distance_path_weight * dp + distance_path_time * tp



    if distance_path_penalty > time_path_penalty:
        return list(time_path)

    else:
        return list(distance_path)



def get_graph(num1, num2):
    node_names = []
    while len(node_names) < num1 * num2:
        name = id_generator()
        while name in node_names:
            name = id_generator()
        node_names.append(name)

    G = nx.grid_2d_graph(num1, num2)
    E = G.edges()

    Elist = []
    for e in E:
        if random.random() > 0.9:
            Elist.append(e)

    for e in Elist:
        G.remove_edge(*e)

    labels = dict()
    for i, node in enumerate(G.nodes()):
        labels[node] = node_names[i]

    if not nx.is_connected(G):
        return get_graph(num1, num2)
    else:
        return G, labels


# -*- coding: utf-8 -*-
"""
Created on Mon Dec 23 16:01:06 2019

@author: monasrazadani
"""



random.seed(6)
from networkx import path_graph, random_layout

num1 = 10
num2 = 10
extra_edges = 0
```

```
edge_min = 1

edge_max = 3

speed_min = 10

speed_max = 60


"This section creates graph"

graph = Graph()


G, labels = get_graph(num1, num2)

h = []

a1 = []

b1 = []

c1 = []

t1 = []

for source, target in G.edges():

    a = labels[source]

    b = labels[target]

    c = random.randrange(edge_min, edge_max)

    x = random.randrange(speed_min, speed_max)

#    c=1

#     x=30

    t = round(60 * c / x)

    h.append((a, b, c, t))

    a1.append(a)

    b1.append(b)

    c1.append(c)

    t1.append(t)
```

```
backup = list(h)

h = list(h)

edges = h


for edge in edges:

   graph.add_edge(*edge)




labels={}

for edge in h:

   labels[(edge[0],edge[1])]=edge[2]

timeless = []

for edge in h:

   timeless.append((edge[0:-1]))

distanceless = []

for edge in h:

   distanceless.append((edge[0:-2]+tuple([edge[-1]])))


seed = 123

random.seed(seed)

np.random.seed(seed)

G=nx.Graph()

G.add_weighted_edges_from(timeless)

pos = nx.spring_layout(G, iterations=10000, weight='myweight')

plt.figure(figsize=(18, 16), dpi= 80, facecolor='w', edgecolor='k')

nx.draw(G,pos,edge_color='black',width=2,linewidths=2,\

node_size=1000,node_color='pink',alpha=0.9,\

labels={node:node for node in G.nodes()},font_size=20)
```

```
labels = {}
for edge in timeless:
    labels[(edge[0],edge[1])]=edge[2]

nx.draw_networkx_edge_labels(G,pos,edge_labels=labels,font_color='red',font_size=20)
plt.axis('off')
plt.show()




# -*- coding: utf-8 -*-
"""
Created on Mon Dec 23 16:01:06 2019

@author: monasrazadani
"""

# random.seed(3)

from matplotlib.pyplot import figure
"This section assigns number of vehicles and passengers"
number_of_vehicles =60
number_of_passengers = 120
vehicle_path_min = 5
passenger_path_min = 3

routes1=[]
vehicle_list = []
```

```
"This section generates vehicles"
for vehicle in range(1,number_of_vehicles+1):


    sent = False
    while sent == False:
      first_random = random.choice(a1)
      second_random = random.choice(b1)
      d = dijsktra(graph, first_random, second_random)
      t = list(d)


      if len(d) >= vehicle_path_min and len(t) >= vehicle_path_min:
         if len(d) >= 5:
            sent = True
            vehicle_list.append ([first_random,second_random])
            print (vehicle)
            print(d[0:-1])


            cg= d[0:-1]


            routes1.append(cg)
            print(routes1)


edges = []
for r in routes1:
   route_edges = [(r[n],r[n+1]) for n in range(len(r)-1)]
   G.add_nodes_from(r)
   G.add_edges_from(route_edges)
   edges.append(route_edges)
```

```
labels = {}
for edge in timeless:
    labels[(edge[0],edge[1])]=edge[2]

#           pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G,pos=pos)
nx.draw_networkx_labels(G,pos=pos)
nx.draw_networkx_edges(G, pos, alpha=0.5)
colors = ['G']
linewidths = [3]
for ctr, edgelist in enumerate(vehicle_list):
    nx.draw_networkx_nodes(G,pos=pos,nodelist=edgelist,node_color = 'limegreen', width=5,\
                    alpha=0.9,linewidths=3,labels={node:node for node in G.nodes()},font_size=10)
    nx.draw_networkx_edge_labels(G,pos,edge_labels=labels,font_color='r',font_size=10)


plt.gcf().set_size_inches(20, 10)
# plt.savefig('this.png')
# plt.figure(figsize=(20,10))


print (vehicle_list)
passenger_list = []

routes =[]
"This section generates passengers"
for passenger in range(1,number_of_passengers+1):
    sent = False
    while sent == False:
        pass_origin = random.choice(a1)
        pass_destination = random.choice(b1)
        d = dijsktra(graph, pass_origin, pass_destination)
        t = list(d)

        if len(d) >= passenger_path_min and len(t) >= passenger_path_min:
            if len(d) >= 3:
                sent = True
                passenger_list.append([pass_origin,pass_destination])
                print ( passenger)
                print(d)

                gg= [ d[index] for index in [0,-1] ]
                routes.append(gg)
                print (routes)
al=[]
edges2 =[]
edges = []
for r in routes:
    route_edges = [r[index] for index in [0]]
    G.add_nodes_from(r)
#    G.add_edges_from(route_edges)
    edges.append(route_edges)
al.append(edges)

for r in routes:
    route_edges = [r[index] for index in [-1]]
    G.add_nodes_from(r)
#    G.add_edges_from(route_edges)
    edges2.append(route_edges)
al.append(edges2)
print("al", al)
labels = {}
for edge in timeless:
    labels[(edge[0],edge[1])]=edge[2]

#            pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G,pos=pos)
nx.draw_networkx_labels(G,pos=pos)
nx.draw_networkx_edges(G, pos, alpha=0.5)
# colors = ['b','y']
# linewidths = [3]
for ctr, edgelist in enumerate(passenger_list):
    nx.draw_networkx_nodes(G,pos=pos,nodelist=edgelist,node_color = 'b', width=5,\
                    alpha=0.9,linewidths=3,labels={node:node for node in G.nodes()},font_size=10)
    nx.draw_networkx_edge_labels(G,pos=pos,edge_labels=labels,font_color='r',font_size=10)

plt.gcf().set_size_inches(20, 10)

# plt.savefig('this.png')
# plt.figure(figsize=(20,10))

print ("passenger_list =",passenger_list)
```

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 23 16:01:06 2019

@author: monasrazadani
"""

# random.seed( 4 )

"This section prints vehicles"

nx.draw_networkx_nodes(G,pos=pos)
nx.draw_networkx_labels(G,pos=pos)
nx.draw_networkx_edges(G, pos, alpha=0.5)

print (vehicle_list)
for ctr, edgelist in enumerate(vehicle_list):
   nx.draw_networkx_nodes(G,pos=pos,nodelist=edgelist,node_color = 'g', width=5,\
                  alpha=0.9,linewidths=3,labels={node:node for node in G.nodes()},font_size=10)
   nx.draw_networkx_edge_labels(G,pos=pos,edge_labels=labels,font_color='r',font_size=10)
# figure(num=None, figsize=(20, 10), dpi=80, facecolor='w', edgecolor='k')

plt.gcf().set_size_inches(20, 10)
# plt.savefig('this.png')
# plt.figure(figsize=(20,10))



# -*- coding: utf-8 -*-
"""
```

```
Created on Mon Dec 23 16:01:06 2019

@author: monasrazadani
"""



global number_of_vehicles, number_of_passengers, vehicle_distance_threshold, max_passengers, vehicle_time_threshold
global pass_time_threshold, pass_distance_threshold, distance_penalty, time_penalty

"This section assigns penalties"
distance_penalty = 10
time_penalty = 0
"This section sets the limitation on vehicle time"
vehicle_time_threshold = 100
"This section sets the limitation on vehicle distance"
vehicle_distance_threshold = 100
"This section sets the limitation on passenger time"
pass_time_threshold = 50
"This section sets the limitation on passenger distance"
pass_distance_threshold = 50
"This section sets the limitation on passenger waiting time"
time_metric = 1
pass_max_wait_time = 20
"This section sets the limitation on number of passengers"
max_passengers = 3

"This section sets the initial values for vehicles and passengers"
db_tot_passenger_weight = 0
```

```
db_tot_vehicle_weight = 0

db_tot_passenger_time = 0

db_tot_vehicle_time = 0

tb_tot_passenger_weight = 0

tb_tot_vehicle_weight = 0

tb_tot_passenger_time = 0

tb_tot_vehicle_time = 0

op_tot_passenger_weight = 0

op_tot_vehicle_weight = 0

op_tot_passenger_time = 0

op_tot_vehicle_time = 0

db_tot_passenger_weight_after = 0

db_tot_vehicle_weight_after = 0

db_tot_passenger_time_after = 0

db_tot_vehicle_time_after = 0

tb_tot_passenger_weight_after = 0

tb_tot_vehicle_weight_after = 0

tb_tot_passenger_time_after = 0

tb_tot_vehicle_time_after = 0

op_tot_passenger_weight_after = 0

op_tot_vehicle_weight_after = 0

op_tot_passenger_time_after = 0

op_tot_vehicle_time_after = 0

"This section creates the paths for vehicles"

db_vehicles = {}

db_vehicle_occupants = {}

tb_vehicles = {}

tb_vehicle_occupants = {}

op_vehicles = {}
```

```
op_vehicle_occupants = {}
for vehicle in range(1,number_of_vehicles+1):
    db_vehicle_occupants[vehicle]= []
    tb_vehicle_occupants[vehicle]= []
    op_vehicle_occupants[vehicle]= []
    sent = False
    while sent == False:
        vehicleli = vehicle-1


        first_random = vehicle_list [vehicleli][0]
        second_random = vehicle_list [vehicleli][1]


        d = dijsktra(graph, first_random, second_random)


#       t = list(d)
#        o = list(d)
        t = dijsktra_time(graph, first_random, second_random)
        o = list(path_picker(graph, d, t, time_penalty, distance_penalty))
        if len(d) >= vehicle_path_min and len(t) >= vehicle_path_min:
            sent = True


            route_weight = 0
            route_time = 0
            for i in range(len(d)-1):
                route_weight += graph.weights[(d[i], d[i + 1])]
                route_time += graph.times[(d[i], d[i + 1])]
            d.append(route_time)
            d.append(route_weight)
            db_vehicles[vehicle] = d
            db_tot_vehicle_weight+=route_weight
            db_tot_vehicle_time+=route_time

            print ( vehicle , "[", first_random , "," , second_random, "]" )

            route_weight = 0
            route_time = 0
            for i in range(len(t)-1):
                route_weight += graph.weights[(t[i], t[i + 1])]
                route_time += graph.times[(t[i], t[i + 1])]
            t.append(route_time)
            t.append(route_weight)
            tb_vehicles[vehicle] = t
            tb_tot_vehicle_weight+=route_weight
            tb_tot_vehicle_time+=route_time

            route_weight = 0
            route_time = 0
            for i in range(len(o)-1):
                route_weight += graph.weights[(o[i], o[i + 1])]
                route_time += graph.times[(o[i], o[i + 1])]
            o.append(route_time)
            o.append(route_weight)
            op_vehicles[vehicle] = o
            op_tot_vehicle_weight+=route_weight
            op_tot_vehicle_time+=route_time

db_original_vehicles=dict(db_vehicles)
tb_original_vehicles=dict(tb_vehicles)
```

```
op_original_vehicles=dict(op_vehicles)


"This section creates passengers' paths"
db_passengers = {}
tb_passengers = {}
op_passengers = {}


for passenger in range(1,number_of_passengers+1):
    sent = False
    while sent == False:
        passen_origin= passenger-1


        pass_origin = passenger_list [passen_origin][0]
        pass_destination = passenger_list [passen_origin][1]
        d = dijsktra(graph, pass_origin, pass_destination)
#         t = list(d)
#          o = list(d)
        t = dijsktra_time(graph, pass_origin, pass_destination)
        o = list(path_picker(graph, d, t, time_penalty, distance_penalty))
        if len(d) >= passenger_path_min and len(t) >= passenger_path_min:
            sent = True


            route_weight = 0
            route_time = 0
            for i in range(len(d)-1):
                route_weight += graph.weights[(d[i], d[i + 1])]
                route_time += graph.times[(d[i], d[i + 1])]
            d.append(route_time)
            d.append(route_weight)
```

```
db_passengers[passenger]=d
db_tot_passenger_weight+=route_weight
db_tot_passenger_time+=route_time

print (db_tot_passenger_time)
print ( passenger ,"[", pass_origin,",", pass_destination,"]")


route_weight = 0
route_time = 0
for i in range(len(t)-1):
    route_weight += graph.weights[(t[i], t[i + 1])]
    route_time += graph.times[(t[i], t[i + 1])]
t.append(route_time)
t.append(route_weight)
tb_passengers[passenger]=t
tb_tot_passenger_weight+=route_weight
tb_tot_passenger_time+=route_time


route_weight = 0
route_time = 0
for i in range(len(o)-1):
    route_weight += graph.weights[(o[i], o[i + 1])]
    route_time += graph.times[(o[i], o[i + 1])]
o.append(route_time)
o.append(route_weight)
op_passengers[passenger]=o
op_tot_passenger_weight+=route_weight
```

```
            op_tot_passenger_time+=route_time

db_original_passgeners = dict(db_passengers)
tb_original_passgeners = dict(tb_passengers)
op_original_passgeners = dict(op_passengers)

db_bad_passengers = []
tb_bad_passengers = []
op_bad_passengers = []

"This section assigns passengers to vehicles"

def assign_passengers_to_vehicles(vehicles, passengers, vehicle_occupants, bad_passengers, metric):

    original_vehicles = dict(vehicles)
    original_passgeners = dict(passengers)
    available_vehicles=[]
    for vehicle in range (1, number_of_vehicles+1):
        available_vehicles.append(vehicle)

    for passenger in range(1, number_of_passengers+1):
        print(passenger, end=" ")
        shortest_path = []
        closest_vehicle = 0
        shortest_distance = 100
        for vehicle in available_vehicles:

            passengers_for_vehicle = []
            for old_pass in vehicle_occupants[vehicle]:
```

```
          passengers_for_vehicle.append((passengers[old_pass][0],passengers[old_pass][-3]))
        passengers_for_vehicle.append((passengers[passenger][0],passengers[passenger][-3]))
        d10 = shortest_path_with_nodes(graph, (vehicles[vehicle][0],vehicles[vehicle][-3]),
passengers_for_vehicle, metric)
        route_weight = 0
        route_time = 0
        for i in range(len(d10)-1):
          route_weight += graph.weights[(d10[i], d10[i + 1])]
          route_time += graph.times[(d10[i], d10[i + 1])]

        passenger_tests = True
        for old_pass in vehicle_occupants[vehicle]+[passenger]:

          new_path = d10[d10.index(passengers[old_pass][0]):d10.index(passengers[old_pass][-3])+1]

          pass_route_weight = 0
          pass_route_time = 0
          for i in range(len(new_path)-1):
            pass_route_weight += graph.weights[(new_path[i], new_path[i + 1])]
            pass_route_time += graph.times[(new_path[i], new_path[i + 1])]

          if (1+pass_distance_threshold/100)*passengers[old_pass][-1] < pass_route_weight:
            passenger_tests = False
          if (1+pass_time_threshold/100)*passengers[old_pass][-2] < pass_route_time:
            passenger_tests = False



          new_wait = 0
          wait_path = vehicles[vehicle][0:d10.index(passengers[old_pass][0])]
```

```
            for i in range(len(wait_path)-1):
                new_wait += graph.times[(d10[i], d10[i + 1])]



            if time_metric == 1:
                if new_wait > (pass_max_wait_time/100)*passengers[old_pass][-2]:

                    passenger_tests = False
            elif time_metric == 2:
                if new_wait > pass_max_wait_time:

                    passenger_tests = False
            else:
                print("The provided value for time_metric is nonsensical!")

        if shortest_distance >= route_weight and route_weight <= (1+vehicle_distance_threshold/100)*(original_vehicles[vehicle][-1]) and \
                route_time <= (1+vehicle_time_threshold/100)*original_vehicles[vehicle][-2] and passenger_tests:
            shortest_distance = route_weight
            shortest_time = route_time
            shortest_path = list(d10)
            closest_vehicle = vehicle
        if closest_vehicle == 0:
            bad_passengers.append(passenger)
        else:
            shortest_path.append(shortest_time)
            shortest_path.append(shortest_distance)
            vehicles[closest_vehicle] = list(shortest_path)
            vehicle_occupants[closest_vehicle].append(passenger)
            for old_pass in vehicle_occupants[closest_vehicle]:
                new_path                                                                      =
shortest_path[shortest_path.index(passengers[old_pass][0]):shortest_path.index(passengers[old_pass][
-3])+1]
                pass_route_weight = 0
                pass_route_time = 0
                for i in range(len(new_path)-1):
                    pass_route_weight += graph.weights[(new_path[i], new_path[i + 1])]
                    pass_route_time += graph.times[(new_path[i], new_path[i + 1])]
                passengers[passenger] = new_path + [pass_route_time] + [pass_route_weight]
            if len (vehicle_occupants[closest_vehicle])==max_passengers:
                available_vehicles.remove(closest_vehicle)

    return None

assign_passengers_to_vehicles(db_vehicles, db_passengers, db_vehicle_occupants, db_bad_passengers,
"db")
assign_passengers_to_vehicles(tb_vehicles,  tb_passengers,  tb_vehicle_occupants,  tb_bad_passengers,
"tb")
assign_passengers_to_vehicles(op_vehicles, op_passengers, op_vehicle_occupants, op_bad_passengers,
"op")



for passenger in range(1,number_of_passengers+1):
    db_tot_passenger_weight_after += db_passengers[passenger][-1]
    db_tot_passenger_time_after += db_passengers[passenger][-2]
    tb_tot_passenger_weight_after += tb_passengers[passenger][-1]
    tb_tot_passenger_time_after += tb_passengers[passenger][-2]
    op_tot_passenger_weight_after += op_passengers[passenger][-1]
    op_tot_passenger_time_after += op_passengers[passenger][-2]



for vehicle in range(1,number_of_vehicles+1):
    db_tot_vehicle_weight_after +=  db_vehicles[vehicle][-1]
    db_tot_vehicle_time_after += db_vehicles[vehicle][-2]
    tb_tot_vehicle_weight_after +=  tb_vehicles[vehicle][-1]
    tb_tot_vehicle_time_after += tb_vehicles[vehicle][-2]
    op_tot_vehicle_weight_after +=  op_vehicles[vehicle][-1]
    op_tot_vehicle_time_after += op_vehicles[vehicle][-2]


# -*- coding: utf-8 -*-
"""
Created on Mon Dec 23 16:01:06 2019

@author: monasrazadani
"""

"This section analyzes the data"
def analyze_metric(original_vehicles, vehicles, original_passengers, passengers, vehicle_occupants, bad_passengers,metric):


    vehicle_weight_before = 1
    vehicle_weight_after = 0
```

```
vehicle_time_before = 1
vehicle_time_after = 0
for vehicle in range(1,number_of_vehicles+1):
    vehicle_weight_before += original_vehicles[vehicle][-1]
    vehicle_weight_after += vehicles[vehicle][-1]
    vehicle_time_before += original_vehicles[vehicle][-2]
    vehicle_time_after += vehicles[vehicle][-2]

passenger_weight_before = 0
passenger_weight_after = 0
passenger_time_before = 0
passenger_time_after = 0
for passenger in range(1,number_of_passengers+1):
    passenger_weight_before += original_passengers[passenger][-1]
    passenger_time_before += original_passengers[passenger][-2]
    passenger_time_after += passengers[passenger][-2]
for passenger in bad_passengers:
    passenger_weight_after += passengers[passenger][-1]

passenger_detour_weight = passenger_weight_after - passenger_weight_before
passenger_waiting_time = 0
for vehicle in vehicles:
    for old_pass in vehicle_occupants[vehicle]:
        passenger_detour_weight += passengers[old_pass][-1]
        wait_path = vehicles[vehicle][0:vehicles[vehicle].index(passengers[old_pass][0])]
        for i in range(len(wait_path)-1):
            passenger_waiting_time += graph.times[(wait_path[i], wait_path[i + 1])]

passenger_time_after += passenger_waiting_time
```

```
    tot_weight_before = vehicle_weight_before+passenger_weight_before
    tot_weight_after = vehicle_weight_after+passenger_weight_after
    tot_time_before = vehicle_time_before+passenger_time_before
    tot_time_after = vehicle_time_after+passenger_time_after



    tot_passenger_detour_time = passenger_time_after - passenger_time_before
    tot_vehicle_detour_time = vehicle_time_after - vehicle_time_before

    first_element_of_non_empty = [l[0] for l in vehicle_occupants.values() if l]
    num_non_empty = len(first_element_of_non_empty)
    num_empty = len(vehicle_occupants) - num_non_empty




    print("\nBefore Ridesharing")
    print("  Distance (mi)", tot_weight_before)
    print("  Time (min)", tot_time_before)

    print("After Ridesharing")
    print("  Distance (mi)", tot_weight_after)
    print("  Time (min)", tot_time_after)

    plot = {"vehicle travel (mi)": [vehicle_weight_before], "passenger travel (mi)":
[passenger_weight_before],\
      "vehicle + passenger (mi)": [tot_weight_before], "vehicles after sharing (mi)": [vehicle_weight_after],\
      "passengers without vehicles (mi)": [passenger_weight_after], "total travel after sharing (mi)": \
      [tot_weight_after]}
    df = pd.DataFrame(data=plot)
    sns.set(style="whitegrid")
    plt.figure(figsize=(16, 6))
    ax=sns.barplot(data=df).set_title(metric)

    plot = {"vehicle travel (mins)": [vehicle_time_before], "passenger travel (mins)": [passenger_time_before],\
      "vehicle + passenger (mins)": [tot_time_before], "vehicles after sharing (mins)": [vehicle_time_after],\
      "passengers after sharing (mins)": [passenger_time_after], "total travel after sharing (mins)": \
      [tot_time_after]}
    df = pd.DataFrame(data=plot)
    sns.set(style="whitegrid")
    plt.figure(figsize=(16, 6))
    ax=sns.barplot(data=df).set_title(metric)

    return None



print("################################")
print("####### DISTANCE BASED #######")
print("################################")
analyze_metric(db_original_vehicles, db_vehicles, db_original_passgeners, db_passengers, db_vehicle_occupants, db_bad_passengers, "db")



print("################################")
print("######### TIME BASED #########")
print("################################")
analyze_metric(tb_original_vehicles, tb_vehicles, tb_original_passgeners, tb_passengers, tb_vehicle_occupants, tb_bad_passengers, "tb")
```

```
print("##############################")

print("######## \"OPTIMIZED\" #########")

print("##############################")

analyze_metric(op_original_vehicles,        op_vehicles,        op_original_passgeners,        op_passengers,
op_vehicle_occupants, op_bad_passengers, "op")
```