



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA INDUSTRIAL

Especialidad de Organización Industrial

**ANALYSIS AND OPTIMIZATION OF
VEHICLE ROUTES AND DRIVER
ASSIGNMENT FOR TRANSPORTATION
NETWORK COMPANIES**

Autor: Álvaro Serrahima de Bedoya

Director: Ali Haghani

Madrid
Junio 2019

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESIS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Álvaro Serrahima de Bedoya DECLARA ser el titular de los derechos de propiedad intelectual de la obra: *Analysis and optimization of vehicle routes and driver assignment for Transportation Network Companies*, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

- a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar “marcas de agua” o cualquier otro sistema de seguridad o de protección.
- b) Reproducir la en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
- c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
- d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
- e) Asignar por defecto a estos trabajos una licencia Creative Commons.
- f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente*).

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

- a) Que la Universidad identifique claramente su nombre como autor de la misma
- b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
- c) Solicitar la retirada de la obra del repositorio por causa justificada.
- d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

- a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
- b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
- c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

- d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

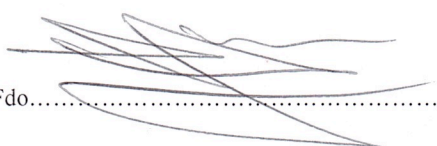
6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusiva del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 27 de junio de 2019

ACEPTA

Fdo.....

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título
ANALYSIS AND OPTIMIZATION OF VEHICLE ROUTES AND DRIVER
ASSIGNMENT FOR TRANSPORTATION NETWORK COMPANIES
en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el
curso académico 2018-2019 es de mi autoría, original e inédito y
no ha sido presentado con anterioridad a otros efectos. El Proyecto no es
plagio de otro, ni total ni parcialmente y la información que ha sido tomada
de otros documentos está debidamente referenciada.

Fdo.: Álvaro Serrahima de Bedoya Fecha: 27/ Junio / 2019



Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Ali Haghani

Fecha: 29 / 6 / 2019





ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA INDUSTRIAL

Especialidad de Organización Industrial

**ANALYSIS AND OPTIMIZATION OF
VEHICLE ROUTES AND DRIVER
ASSIGNMENT FOR TRANSPORTATION
NETWORK COMPANIES**

Autor: Álvaro Serrahima de Bedoya

Director: Ali Haghani

Madrid
Junio 2019

ANALYSIS AND OPTIMIZATION OF VEHICLE ROUTES AND DRIVER ASSIGNMENT FOR TRANSPORTATION NETWORK COMPANIES

Autor: Serrahima de Bedoya, Álvaro

Directores: Haghani, Ali

Entidad Colaboradora: ICAI - Universidad Pontificia Comillas

Los avances en tecnología han permitido que surja una nueva forma de movilidad, las empresas de vehículos de transporte con conductor (VTCs). Desde su llegada, muchos ámbitos del transporte han sufrido cambios. El servicio VTC permite a los usuarios obtener un transporte rápido, flexible y barato, y para los conductores es una fuente de ingresos sencilla. Esto ha hecho que otros servicios como el transporte público o los taxis se han visto afectados. Este último, en mayor medida. Allí donde llegan los VTC el sector del taxi sufre. Esto se debe principalmente a que las VTC ofrecen un servicio muy similar, pero más eficiente y mejor valorado. Esto lo consiguen mediante programas de optimización de rutas y asignación de conductores como el que se pretende diseñar en este proyecto. Además de ello, estas empresas aumentan su eficiencia gracias al uso de redes neuronales que analizan los datos obtenidos por los conductores para detectar patrones en la distribución de la demanda, permitiendo predecir dónde y cuándo se producirán altas demandas de viajes. Por otro lado, sistemas como los viajes compartidos, una opción mediante la cual los pasajeros se prestan a compartir servicio con otros usuarios, permiten a los programas de optimización una mayor variedad de rutas posibles, haciendo que los viajes sean más eficientes.

Sin embargo, no todo lo que ofrece este nuevo servicio son ventajas. En muchos casos el uso de VTCs trae consigo una disminución del uso de transporte público y un aumento del uso del coche, provocando un aumento de las emisiones y la congestión en ciudades. En otros casos, en cambio, el uso de VTC complementa el uso de transporte público y en ocasiones lleva a más personas a dejar de depender de sus vehículos privados. Si este servicio continúa mejorando, podría llevar a una reducción de la posesión de coches. Una forma en que este servicio podría alcanzar el nivel de fiabilidad suficiente para que la gente esté dispuesta a renunciar a tener coches privados es a través de los coches autónomos. Los coches autónomos, unidos con empresas VTC, podrían reducir substancialmente el número de coches en la carretera ya que, actualmente, los coches pasan alrededor del 95 por ciento del tiempo aparcados. El número de coches necesarios para proporcionar servicio a todo el mundo sería mucho menor gracias a los VTCs autónomos. También se reduciría el número de accidentes, las emisiones, y la congestión en las carreteras. Algunas VTC como Lyft o Uber ya han empezado a trabajar en este servicio. Pero para conseguir un servicio fiable, es necesario entrenar las redes neuronales que componen los coches autónomos a través de datos en carretera. Cuanto mayor y más variado sea el número de datos más eficiente será el servicio. Esto, además, contribuiría a la hora de entrenar las redes neuronales de predicción de la demanda, aumentando su precisión.

Utilizando el programa de optimización FICO Xpress, se ha elaborado un programa de diseño de rutas y asignación de pasajeros. Tras revisar estudios anteriores realizados sobre problemas parecidos, se diseña una formulación que obtiene resultados de manera eficiente, proponiendo formas de mejorar el código para que el tiempo de computación necesario sea pequeño. También se plantean distintos enfoques a la minimización de las rutas. Estos enfoques varían en cuál es la variable que se desea minimizar. Hay dos enfoques principales: el primero, minimizar el coste

para la empresa diseñando las rutas más cortas posibles; el segundo, minimizar el malestar de los usuarios diseñando rutas que proporcionan un servicio más rápido para los usuarios. Utilizando el programa y una serie de datos generados aleatoriamente se estudian las diferencias que estos dos enfoques crean en el diseño de las rutas, comparando la distancia, el tiempo de espera, y la eficiencia del servicio. Además, variando los parámetros básicos del programa, se estudiará de qué forma distintas variaciones en la demanda y el servicio prestado modifican las rutas diseñadas. Por ejemplo, que cambios se producen si uno de los coches disponibles rechaza proporcionar el servicio. O cuánto mejora la eficiencia de las rutas al añadir la opción de que los usuarios compartan servicio. Por último, se propone un tercer enfoque de minimización que mezcla los dos enfoques anteriores, de forma que el diseño de rutas se ajusta a los deseos de los usuarios mientras que las rutas no son excesivamente largas. La comparación de los resultados obtenidos con este enfoque respecto a los enfoques anteriores muestra que, en ocasiones, es posible diseñar rutas que reduzcan significativamente la distancia de las rutas mientras que siguen un servicio prácticamente igual a los pasajeros.

ANALYSIS AND OPTIMIZATION OF VEHICLE ROUTES AND DRIVER ASSIGNMENT FOR TRANSPORTATION NETWORK COMPANIES

Author: Serrahima, Alvaro

Directors: Haghani, Ali

Collaborating Entity: ICAI - Universidad Pontificia Comillas

New advances in technology have allowed for a new mobility service to emerge, the Transportation Network Companies (TNCs). Since their introduction, the transportation has experienced some changes. TNCs services provide users with fast, flexible and cheap trips, and it's also for many users a simple source of income. This has affected other services like public transportation or taxis. Studies have shown that the introduction of TNCs in a market brings a reduction of taxi demand. The main cause for that is that TNCs provide a similar service, but in a more efficient way. Users generally rate TNCs service higher than taxi. One of the reasons for that is the use of optimization programs for the design of routes and assignation of drivers to passengers, like the one which will be designed in this project. In addition to that, service efficiency is increased through the use of neural networks, which analyze data obtained by drivers in order to detect patterns in the distribution of the demand, allowing companies to predict where and when there will be high demands for rides. On top of that, the introduction of shared trips, where users accept to share their service with other individuals in exchange for a lower fare, allows this programs to have a higher variety of possible routes, making trips even more efficient.

However, not everything the TNCs offer are advantages. In many cases, the use of TNCs brings a reduction in the use of public transportation and an increase in car usage, causing carbon emissions and congestion in cities to rise. In other cases, instead, TNCs complement the use of public transportation by providing first and last-mile trips, and causing more people to reduce their private car dependence. If TNCs service continues to improve, it could lead to a reduction in private car ownership. One of the ways this service could reach the reliability enough to convince people to give up their car ownership is through the use of autonomous cars. Autonomous cars, used by TNCs, shape what is called autonomous mobility-on-demand. A service where users can request an autonomous car. With a big enough fleet of these cars, service could be reliable and fast enough to substantially reduce the number of cars owned. Right now, cars spend about 95 percent of the time idle. With autonomous mobility-on-demand, the number of cars needed to provide service for all the people would be much lower. In addition, there would be a reduction in the number of accidents, carbon emissions and congestion in cities. Some TNCs like Uber or Lyft have already started working on this service. But to reach a reliable service, it's necessary to train the neural networks used by autonomous cars through on-road data. The bigger and more varied the data is, the more efficient the cars will be. All the data collected by these cars could also be used to further train demand prediction neural networks to increase their accuracy.

Using the FICO Xpress optimization software, we elaborated a program for the design of TNCs routes and assignation of passenger to users. After reviewing previous studies carried on problems similar to the TNCs case, we developed a formulation that solves the problem in an efficient way by proposing methods to reduce the time needed by the program to reach an

optimal solution. Also, the previous studies were used to present different approaches to the minimization of routes. There are two main approaches: the first one, to minimize the total costs for the company by designing the shortest possible routes; the second, to minimize user discomfort through the design of fast routes for the users. Using the program and some different data sets randomly generated, we studied the differences that these two different approaches produce in the design of routes, comparing the distance, the waiting times and the efficiency of the service. In addition, through the variation of the basic parameters of the program, we studied how different variations in the demand and service provide modify the route designs. For example, what changes if one of the drivers decides to reject the trip request. Or also, how route efficiency improves when the sharing option is included in the service. Lastly, we propose a third minimization approach that mixes the two previous approaches. This means that route designs aim for reduced waiting times but also for short trips. The comparison of the results with respect to the previous two approaches shows that, in some occasions, it's possible to design routes that significantly reduce the length of the trip while barely affecting user experience.

Content

Introduction.....	15
PART 1 – ANALYSIS OF TNCs IMPACT.....	17
1.1 Impact on user experience	17
1.2 Impact on drivers	19
1.3 Impact on other mobility services.....	19
1.4 Impact on environment and congestion	20
PART 2 – PROGRAM DESIGN.....	23
2.1 Introduction.....	23
2.2 Main features of the program.....	23
2.3 Literature review	25
2.3.1 The Traveling Salesman Problem (TSP)	25
2.3.2 Dial-a-Ride Problem(DARP).....	26
2.4 Formulation.....	27
2.4.1 Variables	27
2.4.2 Constrains	28
2.4.3 Minimization function	35
2.5 XPRESS Code	39
2.6 Observations	49
2.6.1 Comparing minimization approaches	49
2.6.2 Modifying number of cars available	53
2.6.3 Comparing effect of shared trips.....	54
CONCLUSION.....	55
Appendix.....	56
Full code.....	56
Results obtained from the program tests.....	61
Bibliography	63

Introduction

The concept of shared mobility is not completely new. In the past, when owning a vehicle was not as common as today, carpooling and hitchhiking were common sources of transportation. However, their use had always been limited by the lack of communication methods, which made it complicated for people to reliably find rides. With that, the widespread of self-owned vehicles brought the decline of shared mobility. However, in the last few years, the advances in technology have created new opportunities for this kind of service, making it much easier for users to request safe, reliable and on-demand rides, and also for ride providers to efficiently provide the service. With these advances, a new kind of service emerged and it has been growing since then: Transportation Network Companies.

Transportation Network Companies(TNCs) are on-demand ride services that connect drivers with passengers through online-enabled applications or platforms. Drivers provide the service using their personal vehicle and they don't need to be licensed vehicle-for-hire drivers. TNCs are also referred to as ridesourcing, ride-hailing, or ride-sharing, although this last name is not precise. In comparison to ride-hailing, ride-sharing drivers usually share the same destination with the passenger and the incentive is not always the money. Drivers that operate with TNC services work in exchange of money, which is paid by the users through the app. Drivers often receive 80% of the price while the company keeps the rest. These services usually include a rating system where users can review the service obtained.

The two biggest TNCs are Uber and Lyft, and in the last years they have considerably grown. In June 2015, the number of daily trips provided by Uber was more than 1 million worldwide (Geier, 2015). The last data provided from Uber states that, as of December 2018, the number of daily trips was 14 million (Uber Technologies Inc. , 2019). While Uber operates worldwide, Lyft only provides service in the USA and in some cities in Canada. In the USA, its market share is figured at 29-35%. There are many different TNCs all over the world with many different characteristics. However, Uber and Lyft are the two most representative examples of the current TNCs network and, in this project, they'll be used to represent the way these companies operate.

This project is divided into two main sections. The first one consists on a research and analysis of the current TNCs service, reviewing the impacts and the ways companies like Uber and Lyft are improving the service. The second part consists on the elaboration of a computer program that optimizes the assignation of drivers to customers and the design of routes. This program will be used to study the results of using different optimization approaches with different sets of data. The aim of both sections is to comprehend the operation, impact and opportunities of TNCs and consider the ways this service could be improved in the future.

PART 1 – ANALYSIS OF TNCs IMPACT

The presence of TNCs has grown in the last years and their impacts in the modern society are bigger every day. Some people criticize that TNCs increase the number of cars in the cities and that this service is substituting public transportation and inducing people to use cars more, increasing pollution and congestion. Moreover, critics argue that TNCs risk the security of the users and harm the current transportation economy. Others, however, see it as reliable, flexible, convenient and fast alternative that provides service for a previously unmet demand. Some even consider it to be a revolutionary service that will bring many benefits to the way transportation is conceived. All the impacts caused by ride-hailing services will be reviewed in this section.

1.1 Impact on user experience

The benefits that TNCs service provide are multiple. It allows users to request a ride from almost any place, with low waiting times, relatively low prices and a good user experience. Compared with public transport, ride-hailing is much more flexible and it works at any time of the day. It is easy to use, since it only requires internet connection, and easy to pay, since the payment can be done online through the app. It can even be requested by a different person than the one riding the car. This is useful, for example, for parents that want to request trips for their children and pay for them, or also for business that can hire trips for their employees.

When comparing it to traditional taxi companies, the general opinion mostly indicates that the service provided by TNCs is better. Research and data from Uber shows that the company provides faster, cheaper and better rated service than taxi companies.

Time

A study made in San Francisco found that two thirds of the TNC users waited less than five minutes, while, in the case of the taxis, only 35 percent waited less than ten minutes during the week. This number is reduced to 16 percent on the weekends. (Shaheen, Chan, & Rayle, 2017). This gap in time waited is caused by the higher efficiency of the service provided by TNCs. These companies spend a lot of resources on providing a fast and reliable service for both users and drivers. Some of the ways they achieve that is through the use of complex programs that design optimal routes. Also, they put a lot of effort on analyzing the huge quantities of data they obtain from the records of their drivers. This data is used to train neural networks whose aim is to look for patterns in customer's behavior. Having a deep understanding of user's movement through the cities allows them to develop precise predictions about where and when demands will be located.

Price

Data provided from the company Certify, in USA, showed that the average price for Lyft was \$22.51 and for Uber \$30.03, while for the taxis it was \$34.48 (Certify, 2015). The way Uber and Lyft set the prices for their trips is very similar. There is a base charge and a cost per mile and per minute costs. In addition, both companies include higher prices if the number of requests is higher than normal at that moment of the day. Users also have the chance to

request shared trips for a lower price. In these rides, the user agrees to share the service with other users, which sometimes means longer trips to pick up or drop off other users. In addition to that, users can receive an even lower price if they agree to walk a small distance to the point indicated by the app.

While in traditional taxis the price is not known until the end of the ride, TNCs generally provide the price when the trip is requested, and that price is the final price unless something unexpected happens. Users usually value being able to know the price before the trip is made. It also allows them to compare the prices and length of the trips provided by different TNCs before requesting them, increasing competitiveness between companies.

Wider service

The coverage of the service provided by TNCs is also wider, allowing for people from underserved areas to make use of it. It also provides service to senior and disabled people. For example, Uber has a partnership with wheelchair accessible transportation providers to ensure that part of his fleet can provide this service. Also, thanks to UberAssist, some Uber drivers receive training in how to provide assistance to these people. (Ngo, 2015)

Social impact

The flexibility provided by TNCs allows individuals to use them in many different situations. A study made in San Francisco with 380 users found that 67% of the trips were social and 16% were work related. 40% of the trips began at home. In case of social trips, Uber has a positive impact in the reduction of driving under the influence of alcohol and other substances. A study showed that 20% of Uber users avoided drinking and driving thanks to the service (Rayle, Shaheen, Chan, Dai, & Cervero, 2014). Two other studies found that, since the introduction of Uber, the number of driving accidents caused by alcohol and drugs has been reduced. Lyft has also taken action into reducing drunk driving incidents. In 2017 Lyft and WRAP partnered to offer free rides and discounts during major drinking holidays (Lyft, 2019). However, although these apps reduce the number of accidents caused by substances, TNCs often cause an increase in distracted drivers (Greenwood & Wattal, 2015). The app is the only way drivers have to accept user requests, view the routes and communicate with them. This causes a need for the drivers to constantly use electronic devices while driving, which is the main source of distractions. To reduce this problem, Uber and Lyft design their apps giving a high priority to simplicity and utility. All the information is presented to the driver in a simple and intuitive way and with minimal interaction, reducing the distraction of the drivers.

Safety

Regarding safety, some users mistrust the reliability of the drivers. TNC drivers are not employees of the company. TNCs don't hire their drivers, but instead, they provide them with a service. However, that doesn't mean that companies don't take an active place on ensuring the user safety. Uber, for example, goes through a screening check of their driver's skills and criminal history before authorizing trips through the app. It also counts with a support team, coverage on the trips, and an app that includes an emergency button and the

option to share the trip details with other people. In addition to that, to enforce user confidence, they count with a rating system that prioritizes trips with drivers that have high ratings. Lyft also counts with similar measures too. On top of that, some Governments have also started to take measures to enforce the safety of the service. In California, for example, TNCs drivers must get a specific license, go through training programs, and receive a criminal check (Shaheen, Chan, & Rayle, 2017).

1.2 Impact on drivers

Many drivers have found in TNCs a reliable source of income. As explained before, drivers are considered independent contractors. They obtain the possibility to use the TNC resources and, in exchange, the company keeps a percentage of the fare. In the case of Uber, drivers keep the 80%, but they are responsible for fuel, maintenance and part of the insurance (Ngo, 2015). In most companies, drivers must receive a training before they can start providing the service. Apart from that, the service is very flexible for drivers. They can use their own car and have the schedule they prefer. There are no conditions to when drivers can start and stop providing the service and drivers can reject the trip requests. Most drivers prefer to work part-time. An example of that is the situation in NYC, where the number of TNCs drivers is higher than the number of taxis, but taxis still make ten times more trips, suggesting that TNCs drivers work less hours. For many drivers, it is a relatively easy way to earn money because the requirements are not high. However, there are some TNCs, like Cabify, who have higher standards for their drivers. Cabify requires drivers to wear a suit, to provide users with water, and to be very polite and keep the car clean.

The number of drivers continues to increase every day. In 2013 and 2014, the number of drivers from Uber doubled every 6 months, and increased by tens of thousands annually (National Academies of Sciences, Engineering, and Medicine, 2016)

1.3 Impact on other mobility services

The arrival of TNCs to the transportation landscape has affected the use of other mobility services.

Public transport

Regarding public transportation, there is some concern about whether ride-hailing is used as a substitution for public transportation. If that is the case, that would mean an increase in traffic, pollution and congestion in cities. On the other side, ride-hailing services might be providing a service to people that otherwise would have used their personal vehicle. It could also be complementing public transportation. There is multiple research on this subject. One survey made in San Francisco to 380 TNCs users asked them, after they completed a ride, what they would have done if TNCs hadn't been available. Eight percent of people answered that they wouldn't have made the trip. Thirty percent of the people would have used the public transit and 46 percent would have used a car. Also, 4% of the people had used the TNC to travel from or to a public transit station, suggesting that, in those cases, ride-hailing complemented public transportation by providing first or last-mile trips (Shaheen, Chan, & Rayle, 2017). That is the case of passengers to use the service to access places

where public transport doesn't reach or during after-hours, when there is no available public transportation. In response to this problem, some jurisdictions have arranged partnerships with TNCs to provide a solution for these trips. For example, the City of Rockford is redirecting federal funds to fill after-hour gaps or assist places with poor bus service with the use of TNCs services (Ngo, 2015). In Monrovia, Lyft partnered with the city to increase the transit opportunities and to provide data that helps city officials to identify hotspots and design a better public infrastructure. (Lyft, 2019)

Taxi

As presented before, TNCs present a better service than traditional taxis in almost every aspect. This, added to the fact that the target market for both services is very similar, explains why taxis decline while TNCs customers increase.

A study conducted in Vancouver showed that the growth of Uber in that market consisted primarily on the substitution of taxi trips. There, taxi companies were generally cited to offer a bad service. Since the entry of TNCs, there has been a reduction of 10 to 40 percent in taxi market share. The study also found that the average occupancy of taxis was of 1.1, in comparison to 1.8 of TNCs, which suggests that TNCs are more efficient (Ngo, 2015). There is also another study that supports the idea that TNCs are more efficient than taxis. This study compared the utilization rate of both services and observed that TNCs service was more efficient. The study attributes it to more efficient matching technology, to their larger scale, inefficiency of taxi regulation and the flexibility of Uber's model (Cramer & Krueger, 2016).

More studies relate the decline in taxi usage with the introduction of TNCs. For example, the number of taxi rides in NYC decreased 8% per hour in 2014 and 2015 after the introduction of Uber in 2011 (Brodeur & Nield, 2016). Another study made in San Francisco indicated that 39 percent of the users would have taken a taxi if the TNC service hadn't been available (Shaheen, Chan, & Rayle, 2017). In total, it's estimated a reduction in taxi market share of 10-25% after two to three years of the introduction of a TNC company. That could explain why, since 2014, there have been all over the world multiple protests from taxi drivers against TNCs (Taylor, 2014)

Healthcare

Every year 3.6 million Americans miss their medical appointments due to lack of transportation. This costs about \$150 billions per year. Twenty five percent of lower income patients miss or change an appointment due to lack of transportation. To solve this problem, Lyft launched Lyft Concierge to help healthcare clients get to their appointments (Lyft, 2019). Similarly, Uber has introduced Uber Health.

1.4 Impact on environment and congestion

One of the biggest concerns about TNCs services is about whether their use leads to an increase in emissions, which can be caused by an increased number of cars and by the deadheading. Deadheading refers to the movement of a vehicle with no passengers on it. One study indicates that rides-hailing drivers deadhead a lot, increasing the miles travelled and

therefore the carbon emissions (Anderson, 2014). The reason of that is that TNCs deadhead to look for customers before and after trips. Out-of-service driving is estimated to account for about fifty percent of total distance travelled in New York, and twenty percent in San Francisco. In addition, between 43 and 60 percent of trips are in substitution to other methods that wouldn't include cars, increasing the number of cars. The results of the study made in San Francisco concluded that the waiting time increased in roads with the presence of TNCs as a result of increased congestion, and the disruptive effect of picking and dropping off customers (Erhardt, y otros, 2019). Low prices offered by ride-hailing services might induce people who currently use more sustainable services to request these services.

On the other side, there are many studies that support that the increase of TNCs usage will eventually cause a reduction in driving and in the number of cars owned per person. Forty percent of the people from the San Francisco study declared having reduced their driving because of the service. (Shaheen, Chan, & Rayle, 2017). If TNCs continue to develop to the point where it is easy for individuals to travel without owning a car, car ownership could decrease. However, that doesn't necessarily mean a decrease in distance traveled, but it would probably encourage users to reduce their car usage or increase the occupancy. Some consider that, in the future, it won't be that common for families to own two cars as TNCs provide for better and cheaper alternatives. The truth is that cars spend around 95 percent of the time parked (Barter, 2013). If ride-hailing is used efficiently, this number could be reduced to only 60% (KPMG, 2014). Owning less vehicles would result in a decrease of vehicle emissions, congestion and a decrease of the space needed for parking. However, not all the people is willing to give up owning a car without a completely reliable transportation source, which is very hard to reach using traditional TNCs services. Some people think that the solution to this problem are autonomous cars:

Autonomous cars

Not that many years ago, it was unconceivable to think about self-driven cars. Even for short trips, the number of lines of code that a program would need to be able to solve all the different possible permutations would be too high. However, thanks to deep learning, it became possible. Instead of having to program each of those permutation, the machine builds the data based on its experience.

With the introduction of autonomous cars, it also emerged a new opportunity for transportation services, the Autonomous mobility-on-demand. With it, users can request self-driven car trips. This could eventually reduce the cost of TNCs services. With a big enough fleet of autonomous cars, TNCs services could be available for anyone in a very short time. It would be possible to order a car where and whenever the user desires. Without the need of the passenger to focus on the road, cars interiors could experience a complete remodeling, turning them into work, entertainment or resting spaces.

This possibility is not as far as people might think. The two most popular TNCs, Lyft and Uber, are already working on this kind of service. In December of 2017 Lyft had the first successful test of an autonomous car in a private road, and by November of the next year it launched its first pilot employee car. They have already provided self-driving rides to over 50,000 people in the US. And according to a study, self-driving cars could take off the road 203 million cars by 2030 (Lyft, 2019). However, Lyft states that they will never stop from also providing human-driven service in the future. Uber is also working on this service.

Eventually, if Autonomous mobility-on-demand becomes reliable enough, it could replace car ownership. A survey made to 32 people showed that more than half of them would consider giving up their second car if a self-driving car could be available in less than 15 minutes. Autonomous cars would not only reduce the number of cars needed to supply the demand for mobility, they would also decrease congestion. Just implementing a special lane for self-driven cars in a congested highway could increase its capacity by 500 percent, making commutes shorter (KPGM, 2013). The use of these cars could also lead to a 90 percent reduction in accidents, and carbon emissions could be reduced by a gigaton every year (Bertoncello & Wee, 2015). However, in order to train reliable autonomous car, there is a need to acquire traveling data. The larger and varied the data is, the more effective the car will be. This data could be used not only to train the cars but also to identify user behaviors and patterns in order to create more accurate predictive models, increasing the effectiveness of TNCs even more.

PART 2 – PROGRAM DESIGN

2.1 Introduction

This part of the project consists on the elaboration of a computer program capable of optimizing the assignation of drivers and the design of routes for customers of TNCs. Given a number of customers who formulate requests for transportation from an initial origin (pick-up point) to a specific destination (drop-off point) and a number of available cars to provide the service, the aim of the program will be to design an optimal set of routes capable of accommodating all user requests under a set of constrains.

The design of the optimal routes is characterized by the presence of two conflicting objectives: the minimization of the operating costs and the minimization of user arriving time. Operating costs depend on the distance driven while arriving time depends on the time difference between the car request and the drop-off. The differences between both approaches will be studied as well as the combination of both. Finally, the impact on environment and traffic will also be considered.

2.2 Main features of the program

The program is designed to do the function of what a TNC matching system does. These companies receive the information of the position of the users that request their service as well as the destination they want to go. TNCs use this data to match drivers with requests and to calculate the best routes that take the users to their destinations. They also indicate the price of each trip and the estimated time of it. The aim of the program will be to emulate this service.

In practice, the number of customer requests varies a lot depending on the time of the day, the city and many other factors that make it very hard to predict. TNCs must be able to satisfy all the demand independently of the number of requests. On the other side, the number of cars available to provide the service and their position is also unpredictable. It depends on the number of available drivers at that moment and on the route they decide to make while waiting for a pick-up request. In most companies like Uber, Lyft or Cabify, drivers drive around waiting for a driver request nearby. For these reasons, the program created must be able to operate with any set of data regardless of the number of users, drivers or their positions.

The software used to optimize the program is FICO Xpress Solver, a commercial optimization solver which uses Mosel programming language. The approach used for the problem will be a static model. That is, the program will receive the data of the drivers and customers at a certain point of time and it will optimize the routes and assignation of drivers only for that specific moment. All the information is known beforehand. This approach is a simplification of a real assignation system. In practice, TNCs systems receive real-time information and are able to adjust routes in order to meet demand.

To simulate the conditions of a real situation, the program will have a series of parameters which will be set before running the program. These parameters are the number of customers requesting

cars and the number of cars available at a certain point of time. These numbers don't have to meet any conditions. As it could happen in practice, the number of requests can be higher than the number of cars available, making it necessary for a car to provide service for more than one user. On the contrary, the number of requests can be lower than the number of cars available, which means that some cars won't provide any service.

Regarding the position of cars, instead of having a fixed departure point for the cars, as it would be the case of the bus depot for a bus transportation network, each car has a different departure point which depends of their position when the request was made. The same happens with customer pick-up and drop-off points. There aren't any specified places for that. The pick-up point is the initial position of customer when it requests the car and the destination can be any point. To address these conditions, the program receives the data regarding the position of every car and customer and their respective destinations and operates with that.

For more realism, the program also includes the option for passengers to request a shared trip. Shared trips are a feature included by some TNCs like Uber or Lyft by which users accept to share their ride with other users, usually in exchange of a reduced price. With this feature drivers have the chance to pick up multiple customers in the same trip. Here is an example to illustrate it. In the figure 1 below, users B and D have requested a non-shared trip, so they are taken directly from their request point to their respective destinations C and E. In figure 2, both passengers requested a shared trip allowing the driver to pick up both of them before dropping them off, resulting in a shorter trip. For the second route to be possible, it's necessary that both passengers requested the shared trip.

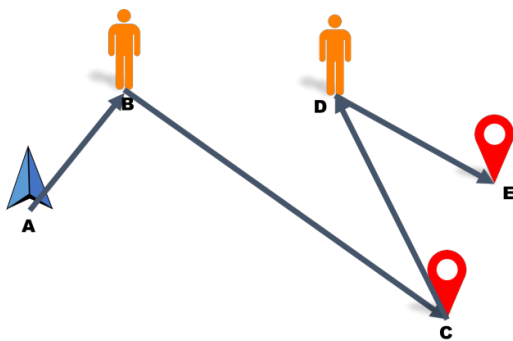


Figure 1 Route without trip sharing

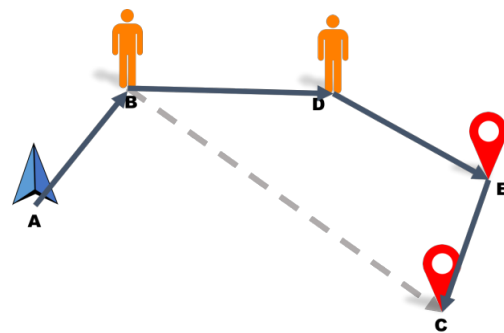


Figure 2 Route with trip sharing

Something to highlight about the pictures above is that the driver doesn't need to drop off the customers in the same order they were picked up. Comparing figures 1 and 2 we can observe that this resulted in a longer trip for user B while user D still had the same route. At one point of the shared trip, between points D and E, both users shared the car. For the development of this program it will assumed that all the cars providing the service are similar and have a maximum capacity of five people inside the car. This means that the driver can only take up to four customers at the same time. It's also important to mention that the fact that a user requests a shared trip doesn't necessary imply that it will share a trip with someone. The program assigns

the optimal route based on the minimization standards and it only makes the customers share the car it that results in a more optimal solution.

Finally, regarding time and schedules, it is a static model, so it will only be considered the positions at a certain point of time. It will also be assumed that all users have requested a trip at that specific moment of time. No scheduled trips are included in this approach. Users cannot decide at what time they want the car to arrive or when they want to be delivered. This means that users can be picked up at any moment and they want to arrive at their destination as soon as possible.

2.3 Literature review

In this section, we will review some relevant past researches about optimization formulation that have some similarities with the assignment problem for TNCs. Two problems will be reviewed: the Dial-a-Ride and the Traveling Salesman problems, providing some insight about different approaches and how that is related to our problem.

2.3.1 The Traveling Salesman Problem (TSP)

The Traveling Salesman Problem(TSP) consists on designing a route for a salesman that wants to visit n cities and then go back to the origin. The aim is to minimize the traveling distance so that every city is visited exactly once, starting and finishing at the same point. The Figure 3 below illustrates an example with eight different locations: the salesman origin point and seven destination cities.



Figure 3 Solution of a TSP

The TSP is one of the most studied optimization problems. It's a much simpler problem than the TNC assignment system but the basic concepts are very similar and there is a very large research on this problem. The paper published by Orman, A. J. and Williams, H. Paul (2004) surveys eight different formulations of the TSP as an Integer Program (IP). The survey proposes different ways to minimize the complexity of the problem as well as ways to ensure the continuity of flow of the solution and the staging of time. The continuity of the flow means that the movement of

people or objects though the route is continuous and logical, without any subtours. The staging of time refers to events occurring in the order desired. These two conditions are also relevant for the TNC problem to ensure that car routes are viable and that pick-up and drop-off points are visited in a logical order.

2.3.2 Dial-a-Ride Problem(DARP)

The Dial-a-Ride Problem (DARP) has many similarities with the TNC problem. In the DARP, users request transportation from specified pickup points to specified destinations. The aim is to plan a minimum-cost set of routes and schedules that provide transportation for all requests under a number of constraints. The most common example of it is the door-to-door transportation for elderly and disabled people case. In the DARP, users often request two trips per day, an inbound and an outbound, specifying the desired time windows for pickup and/or delivery. The main difference between the DARP problem and the TNCs assignment problem is the time span between the request and the pick-up desired time. In the DARP, users request trips in advance, providing the information of when they want to be picked up or arrive. Differences between the desired and actual times create inconvenience for the users. For the TNCs requests, however, users rarely schedule their trips. Instead, users usually request immediate trips and plan to get to their drop-off point as soon as possible. This means that the inconvenience is created by the length of the time they take to arrive at their destination. This distinction is what causes the main differences between the formulations of both cases.

There are two main cases for the DARP: the static case and the dynamic case. In the static case, all requests are known beforehand and no changes can be made once the routes have been designed. In the dynamic case, on the contrary, information is received in real time and there is no information about future requests. When a new request is received at time t , all requests planned before t have already occurred and therefore they can't be modified. The program then has to adapt the routes of all request planned after t to include the new one. In this section, we will not review the literature about this second case because the TNCs formulation proposed is static and therefore has more similarities with the first case.

There are multiple previous researches on the DARP static case that include many variations. The earliest research was carried by Psaraftis (1980) who developed a program referring the single-vehicle case. In Psaraftis DARP there is only one vehicle which provides all the service and there aren't time windows. The objective function is the minimization of a weighted sum between route total completion time and customer dissatisfaction. The author later updated this algorithm in (1983) adding time constraints. The high complexity of this algorithm ($O(n^23^n)$) causes that it can only be solved with relatively small instances. The largest solved contained nine users (Cordeau & Laporte, 2007). The main differences between Psaraftis approach and the one presented in this project is that Psaraftis algorithm only had one vehicle and it included time constraints. Our algorithm includes multiple cars and there aren't time window constraints. However, one of the approaches of the minimization function that will be studied is a weighted sum of the total distance and customer satisfaction, which is similar to Psaraftis approach.

Other approaches to the DARP problem were carried by Sexton (1979) and Sexton and Bodin (1985b). Their solutions included clustering users in groups before solving the routes. In their algorithms, they minimize user inconvenience as a weighted sum of two terms. The first one

being the difference between actual travel time and the direct travel time, and the second one the positive difference between desired and actual times of arrival. With their formulation, they were able to solve sets of data varying between seven and twenty. Another study on the DARP was formulated by Desrosiers et al. (1986) as an integer program including time windows and vehicle capacity, managing to get solutions for up to forty users. (Cordeau & Laporte, 2007).

A study by Borndörfer et al. (1997) uses a different approach, dividing the process in two phases. First, it creates groups of users connected by subtours and then it connects these groups together. Other examples like Rekiek et al. (2006) also cluster the users before creating the routes. Both authors try to minimize the operating costs by also minimizing the number of cars. On the formulation proposed in our project, the number of vehicles can't be minimized as it depends on the number of available drivers at that moment.

2.4 Formulation

The formulation for the TNCs assignment system is defined by two parameters n and m , being n the number of users requesting the service and m the number of cars available to provide it.

In our formulation, we will take the set of total vertices as $T = \{1, 2, \dots, 2n+m\}$ that includes the total number of points given in the initial data. The size $2n+m$ results from n pickup points, n drop-off points and m car initial positions. The set T is partitioned into $\{P, D, C\}$ which refers to: $P = \{1, \dots, n\}$ the set of pickup vertices, $D = \{n+1, \dots, 2n\}$ the set of drop-off vertices and $C = \{2n+1, \dots, 2n+m\}$ the set of car initial positions. For every user with a pickup point $\{i\}$, the corresponding drop-off point is the vertex $\{i+n\}$

We also define the sets $M = \{1, \dots, m\}$ to refer to the number of vehicles available and $N = \{1, \dots, n\}$ for the number of user requests.

2.4.1 Variables

We define the following decision variables:

x_{ij}^k = 1 if arc (i, j) is traversed by vehicle k

= 0 if arc (i, j) is not traversed by vehicle k

p_{ij}^k is the number of passengers that car k carries when traversing the arc (i, j)

f_{ij}^k if the 'flow' of the car k in the arc (i, j)

Other variables included in the formulation:

d_{ij} is the distance between points i and j

s_i = 1 if the user i requested a shared trip

= 0 if the user i requested a non-shared trip

2.4.2 Constrains

We will now define the constrains to which the minimization is subject to and that define the conditions of the problem. The constrains are divided into three groups: Basic, flow and simplification constrains.

A. Basic constrains

These constrains express the basic rules of the problem:

(1)

$$\sum_{i \in T} \sum_{k \in M} x_{ij}^k = 1 \quad \forall j \in P \cup D$$

This constrain states that one car must arrive at every pickup and drop-off point. This ensures that every point is visited once.

(2)

$$\sum_{j \in T} x_{ji}^k = \sum_{j \in T} x_{ij}^k \quad \forall i \in P \cup D, \forall k \in M$$

Every car that goes to a pickup or drop-off point must leave from that point.

(3)

$$\sum_{j \in T} \sum_{k \in M} x_{ij}^k \leq 1 \quad \forall i \in C$$

From each car's initial position only one car can depart.

(4)

$$\sum_{j \in T} x_{(2n+k),j}^k = 1 \quad \forall k \in M$$

This constrain forces each car k to start from its respective initial position. Car positions are placed in set $C = \{2n+1, \dots, 2n+m\}$ so for every car k corresponds the initial position $2n+k$.

(5)

$$\sum_{j \in T} x_{j,(2n+k)}^k = 1 \quad \forall k \in M$$

It forces each car to return to its initial position. This constrain is only used to simplify the formulation. In practice, TNC cars don't need to go back to their initial position, instead they usually just drive around waiting for the next trip request. For that reason, the distance driven by the car from the last drop-off point back to the initial position is never included in the minimization function so it doesn't affect the final solution. The reason why the formulation forces cars to go back to the initial position is that it is easier to formulate a closed route for each car than open routes that start and end at different points.

(6)

$$\sum_{i \in T} x_{ij}^k = \sum_{i \in T} x_{i,(j+n)}^k \quad \forall j \in P, \forall k \in M$$

If a car picks up a customer it must also go to that customer's drop-off point. Remember that for a customer j the corresponding drop-off point is $j+n$.

(7)

$$\sum p_{(k+2n),j}^k = 0 \quad \forall j \in T, \forall k \in M$$

The number of passengers when the car departs from its initial position must be 0.

(8)

$$\sum_{i \in T} p_{j,i}^k - \sum_{i \in T} p_{i,j}^k = \sum_{i \in T} x_{i,j}^k \quad \forall j \in P, \forall k \in M$$

When cars go to a pickup point, they leave with one more passenger than when they arrived.

(9)

$$\sum_{i \in T} p_{i,j}^k - \sum_{i \in T} p_{j,i}^k = \sum_{i \in T} x_{i,j}^k \quad \forall j \in D, \forall k \in M$$

When cars go to a drop-off point, they leave with one less passenger than when they arrived

(10)

$$\sum p_{i,j}^k \leq 4 * x_{i,j}^k \quad \forall i, j \in T, \forall k \in M$$

This constrain has two functions. First, the number of passengers in trips that are not travelled must be 0. Secondly, the number of passengers in the car must be less than 5 at any point.

(11)

$$\sum_{i \in T} p_{i,j}^k = 0 \quad \forall j \in P, \forall k \in M, s_j = 0$$

For customers that don't want to share, which are the ones with $s_j = 0$, the number of passengers when the car picks them up must be 0.

(12)

$$x_{j,j+n}^k = \sum_{i \in T} x_{i,j}^k \quad \forall j \in P, \forall k \in M, s_j = 0$$

Customers that have requested non-shared trips must be taken directly from their pick-up point to their drop-off location.

B. Flow constrains

The "flow" is a concept commonly used in transportation problems that refers to the movement of vehicles between destinations. It's important in these problems to ensure the continuity of flow. This means that the vehicles follow continuous routes. Without flow constrains, subtours are created and the routes become discontinuous. To illustrate it we can use a TSP example:



Figure 4 Subtour example with TSP

Figure 4 above shows the solution that would be obtained in this TSP example if flow constrains weren't included. It can be observed how the basic constrains are satisfied: every city is visited once, there is one arc entering each city and one leaving it, and the route is minimal. However, the route followed by the salesman is not feasible. In this example, the flow is not continuous and two sub tours have been formed. To solve this problem, algorithms usually include this flow conservation constrain:

$$\sum_{i \in T} f_{ij} - \sum_{i \in T} f_{ji} = 1 \quad \forall j \in N - \{1\}, i \neq j$$

This constraint forces that, for every node except the node 1, the flow of the arc that enters a node must be one unit smaller than the flow of the arc that leaves from that node. This causes that tours can only close at the node 1, because it's the only node that doesn't need to follow that rule. The image below shows a visual example of this.

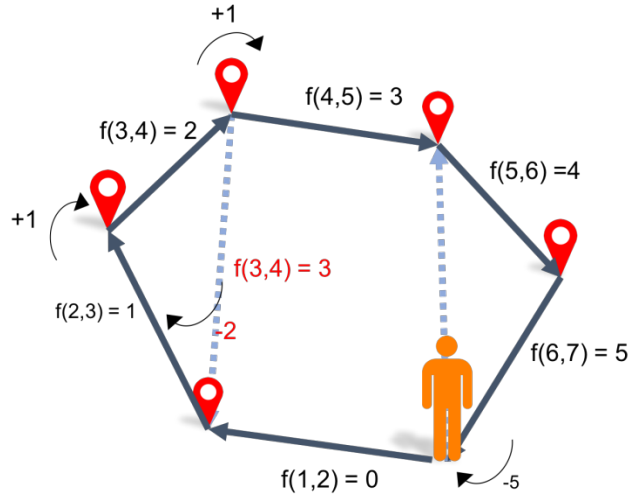


Figure 5 Subtour elimination in TSP

We can observe in the figure above that in every node the value of f increases by 1, so the flow continues to increase. But to close the route, it must reduce its value. For that reason, the only way to close the route is in the initial point, where it can decrease back to 0. If the program tried to create a subtour like in the figure 4 through the arc (3,4), then the flow difference between arc (3,4) and (2,3) would be -2, which wouldn't be allowed by the constraint as it is not the initial node. In conclusion, this avoids that any subtours are created.

In the algorithm designed for the TNC assignment program we have included the same flow variable explained, but with some variations. The constraint we included is:

(13)

$$\sum_{i \in T} f_{j,i}^k - \sum_{i \in T} f_{i,j}^k = \sum_{i \in T} (x_{j,i}^k * d_{j,i}) \quad \forall j \in P \cup D, \forall k \in M$$

The basics behind the constraint are the same: the flow has to increase in every consecutive arc of the tour except in the car initial points, forcing car routes to be closed paths starting and ending at those points. However, by changing the right side of the constraint, the flow variable obtains a second function. In this variation, the flow variable also measures the accumulated distance driven by a car from the initial point to a given point in its route. $f_{i,j}^k$ is the total distance driven by car k when it arrives at node j coming from i . Instead of making that the flow must increase by one on every node, the constraint now states that the flow difference from two consecutive arcs is the distance of the second arc. Here is a visual example:

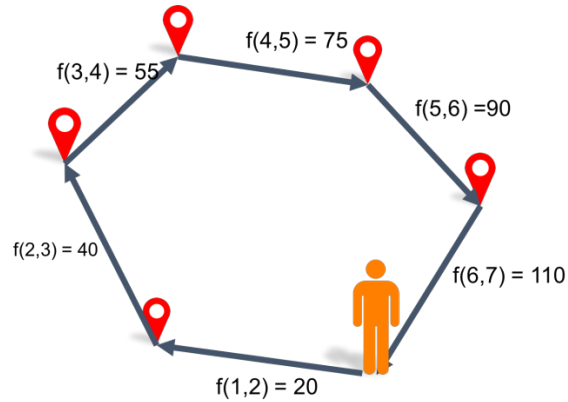


Figure 6 Flow variable measuring accumulated distance in TSP

In the figure above, we can see how the flow variable measures the accumulated distance. At the start of the route the value is the distance between nodes 1 and 2, which is 20. Then, for the arc (2,3), the value is the sum of the initial 20 plus the extra 20 that separate nodes 2 and 3. And it keeps adding with every node. We can appreciate that the only point where the distance doesn't increase is in the node 1.

This new change allows us to keep track of the distance driven by a car at the moment it gets to pickup and drop-off points, letting us know how long it took the driver to get there. Without this change, it would be necessary to include a new decision variable to measure the accumulated distance, which would need to have the same size as the flow variable. That, in addition to variables $f(i, j, k)$, $p(i, j, k)$ and $x(i, j, k)$, would add up to four decision variables, which would increase the complexity and make it much harder to find a solution. The reason it was possible to include the accumulated distance into the flow variable is because it's a value that increases with every new arc and it never decreases. Other variables like, for example, the number of passengers, can't be included in the flow variable because it can either increase or decrease, which would allow for subtours to be created.

Complementing constrain (13) there are other flow constrains needed:

(14)

$$\sum_{j \in T} f_{(2n+k),j}^k = \sum_{j \in T} (x_{(2n+k),j}^k * d_{(2n+k),j}) \quad \forall k \in M$$

When cars depart, their initial traveled distance is 0 so, when they arrive to their first customer, the distance driven must be equal to the distance between those two points.

(15)

$$\sum f_{i,j}^k \leq x_{i,j}^k * W \quad \forall i, j \in T, \forall k \in M$$

This constrain does two functions. First, it sets $f_{i,j}^k = 0$ for every path (i, j) that is not traversed by car k . And secondly, it sets a maximum value W for the flow variable. This maximum value must be set so it's as small as possible without affecting the final solution. The smaller it is the easier it is to find a solution. As explained before, the flow variable represents the total distance driven by a car from its initial position. Then, to select W , it must be the smallest possible value of f that we are sure it's never going to be surpassed. Higher values of W would still work, but the smaller it is without interfering with the solution the faster the program is going to find the routes. Now we will explain how it was selected.

First, in order to never interfere with the solution, it must be considered the scenario that would result in the longest route possible. Taking into account that cars can only visit each location once, with n users and m cars, and assuming that every car is used, the maximum number of arcs possible is $2n+m$. Knowing that, we would need to know the longest possible route that travels to every point once. However, that's a minimization problem on its own. It would be like doing the Travelling Salesman Problem but looking for the longest route, instead of the shortest one. To avoid that, we can approximate it by assuming that every arc could be as long as the longest of them. The value of W would then be:

$$W = (2n + m) \max_{\substack{i \in T, \\ j \in P \cup D}} (d_{ij})$$

However, there is a better way of getting a value for W , and it is to calculate the sum of maximum incoming arcs for every point. In other words, to assume that the arc incoming to each point comes from the furthest point from it. This results in a smaller W than in the previous approach. The formulation for that would be:

$$W = \sum_{i \in T} \max_{j \in P \cup D} (d_{ij})$$

*Note that $j \in P \cup D$ because trips between car locations are not possible

(16)

$$\sum x_{i,i}^k = 0 \quad \forall i \in P \cup D, \forall k \in M$$

The main drawback of having mixed the accumulated distance and the flow distance in one is that the conservation of flow constrain only works for nodes whose distance is greater than 0. However, the arcs that connect nodes to themselves have a distance of 0, allowing for loops to be created with them. For that reason, this constrain forbids travels from one point to itself. This constrain is only applied to pickup and drop-off nodes because, in the case that the optimal route doesn't include all the cars, cars that are not needed are represented with just one trip from their starting point to that same point. This matches the constrain that every car must depart from their initial point and must arrive to their initial point.

But there is still one problem, arcs that connect nodes to themselves are not the only case in

which the distance can be 0. If two different nodes result to be in the exact same place, then subtours could be created between them. This is the main drawback of using this formulation. That being said, we will not contemplate that possibility in our formulation. If nodes result to be in the same exact spot, they can be modified so that they are separated by a negligible distance, and that would be enough to avoid subtours.

(17)

$$\sum_{i \in T} f_{ij}^k - \sum_{i \in T} f_{i,(j-n)}^k \geq d_{(j-n),j} \sum_{i \in T} x_{i,j}^k \quad \forall j \in D, \forall k \in M$$

This constrain ensures that the user drop-off only occurs after the pick-up. We do it by stating that the total distance driven by a car when it arrives at a customer drop-off point must be higher than when it picks up that customer. For that to happen, it would be enough to place a 0 on the right side of the equation. However, we know that the difference will be at least the distance connecting both points so, by including that, we reduce the time needed to find a solution.

C. Simplification constrains

These constrains are not indispensable for the formulation to work. Their function is to help the program find a solution faster by taking away any routes that we know for sure are not going to happen and assigning values to variables which we already know. These constrains should never change the final result obtained in the minimization.

(18)

$$\sum x_{i,j}^k = 0 \quad \forall i, j \in C, \forall k \in M, i \neq j$$

Restrains cars from driving to another car initial position. There is no reason why a car would take that route.

(19)

$$\sum_{k \in M} x_{i,i+n}^k = 1 \quad \forall i \in P, s_i = 0$$

We know that the arc connecting users that don't want to share and their destinations is always going to be traversed by one of the cars. It's similar to constrain (12), but not the same.

(20)

$$x_{i,j}^k = 0 \quad \forall i \in P, \forall j \in T, \forall k \in M \quad j \neq (i + n), \quad s_{(i)} = 0$$

Complementing the previous constrain (19), every arc that connects non-sharing customers with any points apart from its destination are never going to be traversed.

(21)

$$x_{i,j}^k = 0 \quad \forall i \in P, \forall j \in C, \forall k \in M$$

There is no reason why a car would travel from a customer to a car's initial position because, before going back to the starting point, it must always drop-off that customer first.

(22)

$$x_{i,j}^k = 0 \quad \forall i \in C, \forall j \in D, \forall k \in M$$

There is no reason why a car would travel straight from the initial point to a drop-off point. It must go to a pickup point first.

2.4.3 Minimization function

For the minimization function we will review three different approaches that depend on what are our preferences at calculating the optimal routes.

A. Minimization of costs

For this approach, we will assume that all costs are operational and proportional to the distance driven by the cars (longer routes consume more energy and therefore have a higher cost). Therefore, the aim of this minimization will be to develop the shortest route possible. This usually includes not using the entire fleet of cars available. If one car is capable of providing service to all users using the shortest route, then there is no need to use more cars. This approach doesn't consider the holding cost of having a car or the opportunity cost of having unused cars. Neither it takes into account the discomfort of the users created by long waiting times. This minimization approach is the one used in the Traveling Salesman problem. In that problem, the only concern is to create the shortest route possible. It's also commonly used in the DARP in algorithms by Dumas et al. (1989a), Desrosiers et al. (1991), Ioachim et al. (1995) or Cordeau and Laporte (2003a) and many others (Cordeau & Laporte, 2007). However, although their algorithm's minimization function also consisted in minimizing route length, their results were also constrained by the time windows that characterize the DARP, while in our algorithm there's nothing that restricts the amount of time a person can wait. This might lead in some occasions to very short routes but where some users take a long time to reach their destination. The results of this approach and the others proposed are reviewed later in this paper.

The minimization function would then be:

$$\text{Minimize } \sum_{i \in T} \sum_{j \in P \cup D} \sum_{k \in M} x_{i,j}^k d_{i,j}$$

*Note that arcs entering car destinations are not included in the minimization. That is because in practice TNCs don't have to go back to their initial point.

B. Minimization of user discomfort

In this approach, the company's intention is to maximize user satisfaction by minimizing discomfort. For this algorithm, we will assume that user discomfort is proportional to the time waited since they request the trip to the time they arrive at their drop-off point. The later they arrive the more discomfort. We will review three different variations of this minimization that differ in the way the users value their time. The three ways are related with the time waited by the user.

In order to simplify the code, we will presume that all cars drive at the same constant velocity between vertices and that the time spent picking up and delivering customers is negligible. This simplification allows us to conceive time as distance divided by velocity. The minimization of time is therefore reduced to a minimization of distance. There is no need to include the velocity term in the formulation as it is constant and it doesn't affect the minimization.

B1. Minimizing total time

In this algorithm, we don't distinguish between the discomfort caused by waiting for the car, spending time on the car or sharing car with other people. We will assume that users are equally affected by a long wait for the car to arrive as for the route from the pick-up point to the drop-off to be long. They just want to arrive as soon as possible. The objective of the minimization is, therefore, to reduce the total time needed to take each customer to their destination.

The minimization function is:

$$\text{Minimize } \sum_{i \in T} \sum_{j \in D} \sum_{k \in M} f_{i,j}^k$$

In this minimization function the objective function is the total sum of arrival times at the final destinations. We use the flow variable because it contains the distance driven by a driver when the car gets to a point j coming from i . By using the simplification explained before, we can approximate this to the time it took the driver to get to that point. By adding up the values of f for every drop-off point(D) we get the total time users needed to get to their destinations.

B2. Minimizing weighted sum of times

Another possible variation is to add a weighted minimization that differentiates between waiting time and riding time. By splitting both times and adding weights we can more accurately reflect user preferences. The weights could be balanced, for example, by conducting a survey between TNC users in order to know their priorities. Some authors included this type of distinctions in their algorithms like for example Psaraftis (1980). His minimization equation included the notation $[\alpha * WT + (2 - \alpha) * RT]$, being WT the waiting time, RT the riding time and α the customers' time preference constant.

To include this differentiation, the total time has to be divided into WT and RT. In our formulation it could be done with the following formulation:

WT The waiting time for each user is the value of the flow variable when the car

arrives at the pickup point. Then, the total waiting time is the sum of values of flow for every pick-up point.

$$WT = \sum_{i \in T} \sum_{j \in P} \sum_{k \in M} f_{i,j}^k$$

RD The riding time is the difference between arrival time and waiting time. To obtain the total riding time we take the total values of the flow variable at the drop-off points and subtract the total waiting times.

For each user j

$$RT_j = \sum_{i \in T} \sum_{k \in M} f_{i,(j+m)}^k - \sum_{i \in T} \sum_{k \in M} f_{i,j}^k$$

In total

$$RT = \sum_{j \in P} \left(\sum_{i \in T} \sum_{k \in M} f_{i,(j+m)}^k - \sum_{i \in T} \sum_{k \in M} f_{i,j}^k \right)$$

And since D includes the set of vertices of P+m we can rearrange that into

$$RT = \sum_{i \in T} \sum_{j \in D} \sum_{k \in M} f_{i,j}^k - \sum_{i \in T} \sum_{j \in P} \sum_{k \in M} f_{i,j}^k$$

Which is the subtraction of arrival time and waiting time

Putting it all together and adding the time preference constant α we get:

$$\text{Minimize } \alpha \sum_{i \in T} \sum_{j \in P} \sum_{k \in M} f_{i,j}^k + (1 - \alpha) \left(\sum_{i \in T} \sum_{j \in D} \sum_{k \in M} f_{i,j}^k - \sum_{i \in T} \sum_{j \in P} \sum_{k \in M} f_{i,j}^k \right)$$

Which can also be expressed as:

$$\text{Minimize } \alpha WT + (1 - \alpha) RT$$

Being $0 \leq \alpha \leq 1$ the user preference of riding over waiting. If $\alpha > 0.5$ users prefer spending time in the car and if $\alpha < 0.5$ users prefer waiting for it.

B3. Minimizing difference between shortest and actual ride times

There is also another third variation that minimizes the differences between actual and shortest

possible ride times. In other words, users are discomforted if the route taken by the car is longer than the shortest possible route. This type of minimization was used in algorithms like Bodin and Sexton (1986) and Diana and Dessouky (2004). It is a very logical minimization objective and it's very applicable for the TNCs problem. If the actual ride time is much higher than time it would take with a direct trip, users might prefer to use other ways of transportation. What is more, that also increases the probability that other TNC competitors will be offering shorter routes for the users. This might affect the company by causing a reduction of user requests. In order to account for this, the formulation necessary is the following:

$$\text{Minimize} \quad (\text{Actual time} - \text{Shortest possible time})$$

The total actual time for arrival is

$$\sum_{i \in T} \sum_{j \in D} \sum_{k \in M} f_{i,j}^k$$

The total shortest possible time is the sum of distances between pickup and their respective drop-off points

$$\sum_{i \in P} d_{i,(i+n)}$$

Subtracting both terms we get to the minimization equation:

$$\text{Minimize} \quad \sum_{i \in T} \sum_{j \in D} \sum_{k \in M} f_{i,j}^k - \sum_{i \in P} d_{i,(i+n)}$$

C. Minimization of both discomfort and cost

Once reviewed the formulation for each of the approaches for minimization, we propose an objective function that is a linear combination of cost and discomfort. The aim of this function is to minimize user discomfort while keeping the operational costs low. We obtain this by simply adding a weighted sum of both time and total distance.

$$\text{Minimize} \quad w_1 \sum_{i \in T} \sum_{j \in PUD} \sum_{k \in M} x_{i,j}^k d_{i,j} + w_2 \sum_{i \in T} \sum_{j \in D} \sum_{k \in M} f_{i,j}^k$$

Where w_1 and w_2 are the weights given to the minimization of cost and discomfort, respectively. For this time term, we have selected the first of the time minimization functions proposed earlier to keep the function simple and to not add too much complexity to the algorithm.

A similar approach was made in DARP algorithms like Melachrinoudis et al. (2007) which minimized a convex combination of total vehicle transportation costs and total clients' inconvenience time.

2.5 XPRESS Code

The software we used to solve the algorithm is FICO Xpress Solver. It uses Mosel programming language so it is necessary to translate the mathematical notation into Mosel language. In order to test the program, we included in it an algorithm that creates random data depending on the parameters. Every time the program is run the positions of every car, pickup and drop-off point are randomly set, allowing for us to test many different scenarios. We will now present the code and explain its different parts. To facilitate the lecture, the code is written in different colors. Parts of the code presented in blue represent commands and parts in green are comments included in the code.

The program starts calling the Xpress-Optimizer solver and a graphic tool.

```
!@encoding CP1252
model TFG_Transportation_Network_Companies_Assignation_System
uses "mmsxprs"; !gains access to the Xpress-Optimizer solver
uses "mmsvg" !gains access to the graphs plotter
```

First, we introduce the parameters in the code. Parameters are selected by the user. Apart from the basic parameters presented in the mathematical notation: the number of cars(N_CARS) and the number of customers(N_CUSTOMERS); there are other parameters included here:

SIZE: Refers to the scale at which the user wants to run the program en km². The bigger the size is the longer the distances will be. It's relevant because waiting times and distances are not the same in a small area than in a big one, even if there is the same number of users and cars.

CAR_SPEED: It's the speed used to approximate the time it takes the drivers to traverse the arcs. It's the same for every car. For the program we used 50km/h, which is a usual speed limit inside a town.

SHARE_PROB: This parameter refers to the probability of the random data generator to assign a ride to be shared. If this variable is set to 1, all the requests will be of shared rides, and if it is set to 0, none of them will. Changing the parameter allows to get different sets of data and observe how routes change depending on the portion of people sharing.

BASE and COST parameters: They are used to estimate the price each user will have to pay for the trip. To estimate the prices, the pricing strategy is similar to the one used by Uber and Lyft, which includes a base price, a cost per minute and a cost per km driven. Also, there are different fares and costs for shared and non-shared trips, as the first ones are usually cheaper. The specific fares used below and used to test the program are the Uber prices in Washington DC.

```
parameters          !Customizable variables
N_CARS = 3          !Number of cars
N_CUSTOMERS = 6    !Number of passengers

SIZE = 25  !Size of the map in km^2
CAR_SPEED = 50 !Average car speed in km/h
```

```

SHARE_PROB = 0.5 !Probability to share ride
BASE_FARE = 3.21 !Base fare for normal car
COST_KM = 0.5 !Cost per KM
COST_MINUTE = 0.3 !Cost per minute
BASE_FARE_SHARE = 2.74 !Base fare for shared cars
COST_KM_SHARE = 0.5 !Cost per KM on shared cars
COST_MINUTE_SHARE = 0.25 !Cost per minute with shared car

```

end-parameters

After the parameters, the program receives the declaration of sets and variables. Next to each declaration is, in parenthesis, the corresponding variable of the ones presented in the formulation.

declarations

```

CUSTOMERS = 1..N_CUSTOMERS      !(Corresponds to set P of pickup points)
DESTINATIONS = ((N_CUSTOMERS+1)..(2*N_CUSTOMERS))  !(Set D of drop-off points)
CAR_POSITIONS = ((2*N_CUSTOMERS+1)..(2*N_CUSTOMERS+N_CARS))  !(Set C of car initial positions)
TOTAL_NO_CAR = 1..(2*N_CUSTOMERS)      !(P+D of pickup and drop-off points)
TOTAL = 1..(N_CARS+2*N_CUSTOMERS)      !(Set T of total data)

```

```

CARS = 1..N_CARS      !(Set M of number of cars)

```

```

COORD: array(TOTAL, 1..2) of real !coordinates x,y of each driver, customer and final destination
DIST: array(TOTAL, TOTAL) of real !(d)driving distance from i to j
SHARE: array(CUSTOMERS) of real !(s)Whether a customer wants to share ride or not

```

```

FLOW: array(TOTAL, TOTAL, CARS) of mpvar !(f) Variable that controls: 1.That routes only start at cars and end up at cars(no impossible loops). 2.Time passed since the start of the model to when the car arrives at a certain point

```

```

X: array(TOTAL, TOTAL, CARS) of mpvar !(x) Whether the car 'k' goes from 'i' to 'j' or not
PASSENGERS: array(TOTAL, TOTAL, CARS) of mpvar !(p) Number of passengers car k has when travelling from i to j

```

end-declarations

After that, some formulation is needed before the constraints are introduced. First, the program creates random coordinates for each of the car initial positions and for each pick-up and drop-off points. Then, it saves the distances between each node in the variable DIST. It also assigns whether users requested a shared ride or not depending on the SHARE_PROB parameter. Finally, the X variable is set binary so that it can only take the values of 1 and 0.

```

forall (i in TOTAL, j in 1..2) !Randomly assigns coordinates to customers, cars and destinations
    COORD(i,j) := 100*random

```

```

forall (i in CUSTOMERS) do !Randomly assigns whether customers want to share or not
    SHARE(i) := random
    if(SHARE(i) <= (1-SHARE_PROB)) then
        SHARE(i) := 0 !If share is 0 then the customer doesn't want to share
    else

```



```

        SHARE(i) := 1                !If share is 1 then the customer wants to share
    end-if
end-do

forall (i in TOTAL, j in TOTAL)
    DIST(i,j) := sqrt((COORD(j,1)-COORD(i,1))^2+(COORD(j,2)-COORD(i,2))^2)
    !calculates all distances between points and adjusts it to the size of the map if the parameters

forall (i in TOTAL, j in TOTAL, k in CARS)
    X(i,j,k) is_binary                !makes X binary

SPEED := CAR_SPEED/60    !Speed in kilometers per minute

```

If instead of generating random data the intention is to use a specific set of data, the user can do it by substituting the section of the code above with the one below. The data is taken from a .txt document that must include the coordinates of every node and the SHARE matrix.

This formulation was used to perform different tests with the program to observe the results obtained with different sets of data. The results are presented later in this document.

```

SPEED := CAR_SPEED/60    !Speed in kilometers per minute

initializations from 'T11_3-6_1.txt'    ! Gets the information from the data document
    COORD as 'COORDINATES'
    SHARE as 'SHARE'
end-initializations

forall (i in TOTAL, j in TOTAL)
    DIST(i,j) := sqrt((COORD(j,1)-COORD(i,1))^2+(COORD(j,2)-COORD(i,2))^2)
    !calculates all distances between points and adjusts it to the size of the map if the parameters

forall (i in TOTAL, j in TOTAL, k in CARS)
    X(i,j,k) is_binary                !makes X binary

```

Then, the constrains are introduced into the program. These constrains are the same ones presented in the mathematical notation. Next to each constrain is, in parenthesis, the number assigned to their respective constrain from the section **4. Formulation**.

```

!Constrains

forall (i in CUSTOMERS+DESTINATIONS) do
    sum(j in TOTAL, k in CARS)(X(i,j,k)) = 1    !(1) a car must arrive to every customer
    forall(k in CARS)
        sum(j in TOTAL)(X(i,j,k)) = sum(j in TOTAL)(X(j,i,k))    !(2) if a car goes to a customer it
        must leave from that customer
    end-do

forall (i in CAR_POSITIONS)

```

$\text{sum}(j \text{ in TOTAL}, k \text{ in CARS})(X(i,j,k)) \leq 1$!(3) From every car location only one car can depart. No cars will depart if that car is not needed

$\text{forall}(k \text{ in CARS})$
 $\text{sum}(j \text{ in TOTAL})(X(k+2*N_CUSTOMERS,j,k)) = 1$!(4) each car has to leave from its initial position

$\text{forall}(k \text{ in CARS})$
 $\text{sum}(j \text{ in TOTAL})(X(j, k+2*N_CUSTOMERS,k)) = 1$!(5) each car has to end at its initial position

$\text{forall}(j \text{ in DESTINATIONS}, k \text{ in CARS}) \text{ do}$
 $\text{sum}(i \text{ in TOTAL})(X(i, j, k)) = \text{sum}(i \text{ in TOTAL})(X(i, j-N_CUSTOMERS, k))$!(6) if a car picks up a customer it must deliver that customer
 $\text{sum}(i \text{ in TOTAL})(\text{FLOW}(i,j,k)) - \text{sum}(i \text{ in TOTAL})(\text{FLOW}(i,j-N_CUSTOMERS,k)) \geq \text{DIST}(j-N_CUSTOMERS,j) * \text{sum}(i \text{ in TOTAL})(X(i,j,k))$!(7) we make sure that a car doesn't deliver a customer before picking it up
 end-do

$\text{forall}(i \text{ in CAR_POSITIONS}, j \text{ in CAR_POSITIONS}, k \text{ in CARS} | i <> j)$
 $X(i,j,k) = 0$!(8) Cars should never drive to another car's position. This constrain is not necessary but helps finding a solution faster

$W := \text{sum}(i \text{ in TOTAL})(\text{max}(j \text{ in TOTAL_NO_CAR})(\text{DIST}(i,j)))$
 $\text{forall}(i \text{ in TOTAL}, j \text{ in TOTAL}, k \text{ in CARS})$
 $\text{FLOW}(i,j,k) \leq X(i,j,k) * W$!(9) The flow of a route that is not taken by the car is 0. Also, flow should never be bigger than the sum of biggest arc incoming to each point (W)

$\text{forall}(k \text{ in CARS})$
 $\text{sum}(j \text{ in TOTAL})(\text{FLOW}(k+2*N_CUSTOMERS,j,k)) = \text{sum}(j \text{ in TOTAL})(X(k+2*N_CUSTOMERS,j,k) * \text{DIST}(k+2*N_CUSTOMERS,j))$!(10) When cars depart their initial distance driven is 0 so the distance driven when they arrive to their first customer has to be the distance of that arc.

$\text{forall}(k \text{ in CARS}, j \text{ in TOTAL_NO_CAR}) \text{ do}$
 $\text{sum}(i \text{ in TOTAL})(\text{FLOW}(j,i,k)) - \text{sum}(i \text{ in TOTAL})(\text{FLOW}(i,j,k)) = \text{sum}(i \text{ in TOTAL})(X(j,i,k) * \text{DIST}(j,i))$
 !(11) We use the variable FLOW not only to make sure no loops are created but also to account for the time it has taken the car to arrive at one location.
 end-do

$\text{forall}(j \text{ in TOTAL_NO_CAR}, k \text{ in CARS})$
 $X(j,j,k) = 0$!(12) Forbids travels from one point to itself. Without this constrain loops are formed between points and themselves as, because distance from i to i is 0, the previous condition doesn't restrain it

$\text{forall}(k \text{ in CARS}, j \text{ in TOTAL})$
 $\text{PASSENGERS}(k+2*N_CUSTOMERS, j, k) = 0$!(13) The number of passengers a car has when it starts is 0

$\text{forall}(j \text{ in CUSTOMERS}, k \text{ in CARS})$
 $\text{sum}(i \text{ in TOTAL})(\text{PASSENGERS}(j, i, k)) - \text{sum}(i \text{ in TOTAL})(\text{PASSENGERS}(i, j, k)) = \text{sum}(i \text{ in TOTAL})(X(i,j,k))$!(14) When a car picks up a customer the number of passengers adds 1

$\text{forall}(j \text{ in DESTINATIONS}, k \text{ in CARS})$
 $\text{sum}(i \text{ in TOTAL})(\text{PASSENGERS}(i, j, k)) - \text{sum}(i \text{ in TOTAL})(\text{PASSENGERS}(j, i, k)) = \text{sum}(i \text{ in TOTAL})(X(i,j,k))$!(15) When a car drops a customer the number of passengers subtracts 1

$\text{forall}(i \text{ in TOTAL}, j \text{ in TOTAL}, k \text{ in CARS})$
 $\text{PASSENGERS}(i,j,k) \leq 4 * X(i,j,k)$!(16) There are passengers only in the routes that the car drives. Also number of passengers must always be 4 or less.

```

forall(k in CARS, j in CUSTOMERS | SHARE(j)=0) do
    sum(i in TOTAL)(PASSENGERS(i,j,k)) = 0
    X(j,i+N_CUSTOMERS,k) = sum(i in TOTAL)(X(i,j,k))
end-do

forall(i in CUSTOMERS | SHARE(i)=0) do
    sum(k in CARS)(X(i,i+N_CUSTOMERS,k))=1
    forall(j in TOTAL, k in CARS | j<>(i+N_CUSTOMERS))
        X(i,j,k) = 0
end-do

forall(i in CUSTOMERS, j in CAR_POSITIONS, k in CARS)
    X(i,j,k)=0

forall(i in CAR_POSITIONS, j in DESTINATIONS, k in CARS)
    X(i,j,k)=0

```

In reference to the constrains 18-22, which are the simplification constrains, they don't affect the final solution. Their only function is to avoid the program from looking into routes that are known beforehand that will not be taken. However, it is important to check how the program runs without these constrains to ensure that basic constrains are well designed. If the solutions with and without constrains 18-22 are different, it could mean that the route selected by the program includes illogical or non-optimal arcs, like sending cars to other car initial positions, for example. If that is the case, knowing that no optimal route would include arcs like that, it means that basic constrains are wrong.

Once all the constrains have been included, the program looks for the optimal route. The object that the program will try to minimize is indicated in the objective function. In section 4.2 **Minimization function** we presented three different approaches for the objective function: minimization of cost, discomfort and a weighted sum of both. For each approach, the corresponding minimization function in XPRESS language is:

- (a) `minimize(sum(i in TOTAL, j in TOTAL_NO_CAR, k in CARS)(X(i,j,k)*DIST(i,j)))` ! A. Minimization of cost
- (b) `minimize(sum(i in TOTAL, j in DESTINATIONS, k in CARS)(FLOW(i,j,k)))` ! B1. Minimizing total time
- (c) `minimize (alpha*sum(i in TOTAL, j in CUSTOMERS, k in CARS)(FLOW(i,j,k))+(1-alpha)*(sum(i in TOTAL, j in DESTINATIONS, k in CARS)(FLOW(i,j,k)) - sum(i in TOTAL, j in CUSTOMERS, k in CARS)(FLOW(i,j,k))))`
! B2. Minimizing weighted sum of times
- (d) `minimize(sum(i in TOTAL, j in DESTINATIONS, k in CARS)(FLOW(i,j,k))- sum(i in CUSTOMERS)(DIST(i,i+N_CUSTOMERS)))`
! B3. Minimizing difference between shortest and actual ride times
- (e) `minimize(W1*sum(i in TOTAL, j in TOTAL_NO_CAR, k in CARS)(X(i,j,k)*DIST(i,j))+W2*sum(i in TOTAL, j in DESTINATIONS, k in CARS)(FLOW(i,j,k)))` ! C. Minimization of both discomfort and cost

Only one minimization can be done at a time so the program must only contain one of the equations (a)-(e) when running, depending on what the user wants minimize.

For the approach (c) it would be necessary to declare *alpha* in the parameter declaration section, selecting the desired value $\in [0,1]$

Likewise, for the approach (c) it would be necessary to declare W1 and W2. The values selected for these variables to run the program where:

```
W1 := 1
W2 := 1/SPEED
```

By choosing these values we are stating that the value of saving 1 minute is equivalent to the value of saving 1 kilometer. In this program we approximate the time to the distance driven with the formula $\text{Time} = \text{Distance} / \text{Speed}$. So by setting $W2 = 1 / \text{SPEED}$ we are converting the second term of the minimization into time.

After the minimization, the program shows the results thanks to the following formulation:

```
!Writes results
```

```
T := sqrt(SIZE)/(SPEED*100)*sum(i in TOTAL, j in DESTINATIONS, k in CARS)(getsol(FLOW(i,j,k)))
```

```
!Time= Distance/Speed and then multiplied by sqrt(SIZE)/100 to scale the coordinates to the size of the map selected
```

```
D := sqrt(SIZE)/100*sum(i in TOTAL, j in TOTAL_NO_CAR, k in CARS)(getsol(X(i,j,k))*DIST(i,j))      !Total distance driven by cars to customers and destinations and scaled to the size of the map
```

```
forall(j in CUSTOMERS) do
```

```
  writeln("Passenger ", j, ":")
```

```
  KM_PICKUP := sum(k in CARS, i in TOTAL)(getsol(FLOW(i,j,k)))*sqrt(SIZE)/(100) !Calculates the distance from car's initial position to pickup and scale it to KM. Coordinates go from 1 to 100 so we divide by 100 to get a percentage of the map. Then we multiply by map size to make distances proportional to the map size.
```

```
  KM_ARRIVAL := sum(k in CARS, i in TOTAL)(getsol(FLOW(i,j+N_CUSTOMERS,k)))*sqrt(SIZE)/(100) !Calculates distance driven from car's initial position to arrival
```

```
  WAIT := KM_PICKUP/SPEED !Time passed since the car is ordered to when it picks up the customer
```

```
  ARRIVAL := KM_ARRIVAL/SPEED !Time passed since the car is ordered to when it drops the customer off
```

```
  PRICE := (1-SHARE(j))*(BASE_FARE + COST_KM*(KM_ARRIVAL-KM_PICKUP)+ COST_MINUTE*(ARRIVAL-WAIT)) + (SHARE(j))*(BASE_FARE_SHARE + COST_KM_SHARE*(KM_ARRIVAL-KM_PICKUP)+COST_MINUTE_SHARE*(ARRIVAL-WAIT))
```

```
  writeln("Wait " , round(WAIT*100)/100, " min")
```

```
  writeln("Arrive in " , round(ARRIVAL*100)/100, " min")
```

```
  writeln("Price $" , PRICE)
```

```
end-do
```

```
writeln("")
```

```
writeln("Total time to arrival = ", T, " min")
```

```
writeln("Average time = ", (T)/N_CUSTOMERS, " min")
```

```
writeln("Total distance driven = ", sqrt(SIZE)/100*sum(i in TOTAL, j in TOTAL_NO_CAR, k in CARS)(getsol(X(i,j,k))*DIST(i,j)), " km")
```

```
writeln("Total deadheading distance = ", sqrt(SIZE)/100*sum(i in TOTAL, j in TOTAL_NO_CAR, k in CARS| getsol(PASSENGERS(i,j,k)=0)(getsol(X(i,j,k))*DIST(i,j)), " km")
```

The results include the time waited by each passenger for the car to arrive, the total time to arrival and the estimated price of the trip. This price is calculated with the formula:

$$\text{Price} = \text{Base fare} + \text{Cost per km} * \text{Km} + \text{Cost per Minute} * \text{Minutes}$$

It also shows the total time to arrival, the average time per user, the total distance driven and the total deadheading distance. Deadheading distance refers to the distance driven by the car with no users inside it.

```
Passenger 1:
  Wait 1.22 min
  Arrive in 4.31 min
  Price $5.425511323
Passenger 2:
  Wait 10.71 min
  Arrive in 15.58 min
  Price $6.702809045
Passenger 3:
  Wait 14.05 min
  Arrive in 17.97 min
  Price $6.021103487
Passenger 4:
  Wait 4.26 min
  Arrive in 5.97 min
  Price $4.439083957
Passenger 5:
  Wait 5.93 min
  Arrive in 8.94 min
  Price $5.371031523
Passenger 6:
  Wait 8.88 min
  Arrive in 11.58 min
  Price $5.144190755

Total time to arrival = 64.37080038 min
Average time = 10.72846673 min
Total distance driven = 27.96554091 km
Total deadheading distance = 11.86818034 km
```

Figure 7 Program results display example with 6 users

Coordinates are random and different every time the program is run. For that reason, the formulation also displays the coordinates used in the run so they can be used as data to run the program again. This is useful to run the program using the same data but with different minimization approaches in order to compare. Also, to see how changes in the data affect the design of the routes, like for example changing users from shared to non-shared requests, or reducing the number of cars available.

!Write down the coordinates used so they can be used again

```
writeln("")
writeln("COORDINATES:[")
forall(i in TOTAL)
  writeln(COORD(i,1), " ", COORD(i,2))
writeln("]")
writeln("SHARE:[")
forall(i in CUSTOMERS)
  writeln(SHARE(i))
writeln("]")
```

Here is an example of how it looks.

```
COORDINATES: [  
47.86420844      49.51615713  
73.12374621      79.84625184  
77.32200594      50.95376011  
20.9704002       53.75123935  
41.97780511      14.95700968  
18.1202761       97.07967634  
82.59048494      49.44828476  
90.80260349      97.60886077  
0.6898233205     24.12673357  
98.01113368      4.624326576  
33.51723251      52.27160573  
37.45583773      34.20314581  
30.65278564      28.87744458  
17.57492433      22.27052358  
0.9672076912     21.05676011  
]  
SHARE: [  
0  
1  
0  
0  
0  
0  
0  
1  
]
```

Figure 8 Example of the display of data by the program

Finally, the last part of the code consists on the display of a map that shows the results. Using the coordinates given in the data and the results obtained from the minimization function the program draws a map that shows the routes taken by the cars and the passengers inside the car at every arc.

!Display visual result

svgsetgraphviewbox(-10,-10,120,120) ! Graph size

svgsave("Carsmap.svg") ! Save graphic to file

svgaddfile("PASSENGER2.png", "CUSTOMER") !Adds the image used to indicate pickup point

svgaddfile("DESTINATION.png", "DESTINATION") !Image used to indicate drop-off points

svgaddfile("CAR5.png", "CAR") !Image used to indicate car initial position

!Draws positions of cars and customers

svgaddgroup("CIRCLES","Coordinates", SVG_BLACK)

forall(i in TOTAL) do

 svgaddcircle(COORD(i,1),COORD(i,2), 0.1) !Creates a circle wherever there is a coordinate

 svgsetstyle(svggetlastobj, SVG_STROKEWIDTH, 0.2) ! Wider border

 if(i<=N_CUSTOMERS) then ! For customer locations:

 svgsetstyle(svggetlastobj, SVG_FILL, SVG_RED) ! Fills circle red

 svgaddimage("CUSTOMER", COORD(i,1)-2.5,COORD(i,2), 5, 5)

 svgaddtext(COORD(i,1)+1, COORD(i,2)+2, "" + i + ".P" + i + " " + SHARE(i)) ! Adds
 a label to each customer that shows its number and whether he shares or not

 end-if

 if (i>N_CUSTOMERS and i<=2*N_CUSTOMERS) then ! For customer destinations:

 svgsetstyle(svggetlastobj, SVG_FILL, SVG_GREEN) ! Fills circle green

 svgaddimage("DESTINATION", COORD(i,1)-2.5,COORD(i,2), 5, 5)

 svgaddtext(COORD(i,1)+1, COORD(i,2)+1, "" + i + ".F" + (i-N_CUSTOMERS)) ! Adds
 a label to each destination

 end-if

```

if (i>2*N_CUSTOMERS) then                                     !For each Car:
    svgsetstyle(svggetlastobj, SVG_FILL, SVG_BLUE) !Fills circle blue
    svgaddimage("CAR", COORD(i,1)-2.5,COORD(i,2)-2.5, 5, 5)
    svgaddtext(COORD(i,1)+1, COORD(i,2)+2, "" + i + ".C" + (i-2*N_CUSTOMERS))
    !Adds a label to each car
end-if
svgsetstyle(svggetlastobj, SVG_FONTSIZE, "1pt") !Size of the labels names
svgsetstyle(svggetlastobj, SVG_FONTWEIGHT, "bold") !Makes the labels bold
end-do

forall(k in CARS) do
    svgaddgroup("CAR_ROUTE"+k, "Car route "+ k, SVG_BLACK)
    forall(i in TOTAL, j in TOTAL| getsol(X(i,j,k))>0.1) do !For every trip existing between i and j
        svgaddarrow(COORD(i,1),COORD(i,2),COORD(j,1),COORD(j,2)) !Draws the line between the
        coordinates of i and j
        svgsetstyle(SVG_STROKEWIDTH, 0.2) !Wider line
        svgaddtext((COORD(i,1)+COORD(j,1))/2, (COORD(i,2)+COORD(j,2))/2, ""+
        getsol(PASSENGERS(i,j,k)))
        svgsetstyle(svggetlastobj, SVG_FONTSIZE, "2pt") !Size of the labels names
        svgsetstyle(svggetlastobj, SVG_FONTWEIGHT, "bold") !Makes the labels bold
    end-do
end-do

svgrefresh ! Display graphic
svgwaitclose ! Wait until window is closed

end-model

```

Every time the program is run, once the minimization is complete the program displays a map like the one shown in Figure 8 below.

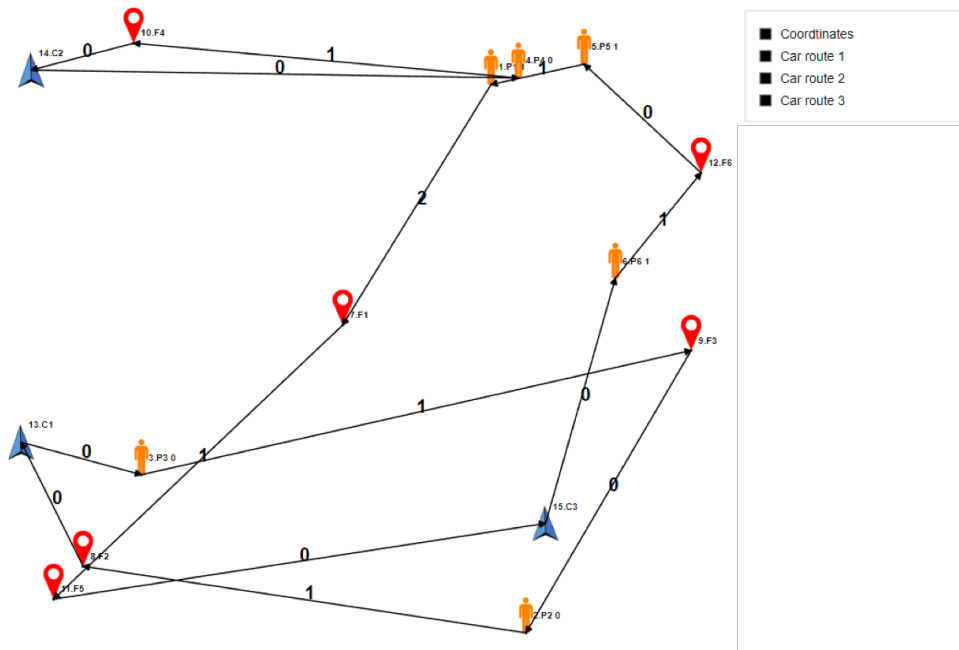


Figure 9 Example of the graphic display of the program run with 6 users and 3 cars

In the map, car initial positions are represented by blue triangles, pickup points are represented by orange figures and drop-off points are represented by red pins. Next to each location is showed a text that indicates the position in the data followed by a letter that indicates what kind of node it is: Car(C), P(Passenger) or Final destination(F); and a number that indicates in which position it is inside its set. Therefore, the drop-off point that corresponds to Passenger 1(P1) is the Final destination 1(F1), P2 corresponds to D2 and so on. Additionally, next to each pickup-node there is a number that indicates the value of the SHARE variable for that customer: “1” if the user requested a shared trip, or “0” if not. The designed routes are indicated with arrows that connect the nodes. Next to each arrow there is a number that indicates the number of passengers being carried by the car on each arc. The objective function used in figure 9 was the minimization of total arrival time. In this example, there are 3 cars and 6 user requests. The number 1 shown next to passengers 5 and 1 indicates that they requested a shared. In this case, the program decided that the shortest route was to make use of the sharing option and pick both of them consecutively. That is why in the arc(1,7) there are 2 passengers inside the car.

In addition, the display of the map has been programmed so that it gives the option to select which routes the user wants to be shown, making it possible to see each of the routes independently. The figure 10 below shows the same display as figure 9 but only route 1 has been selected, making it easier to visualize.

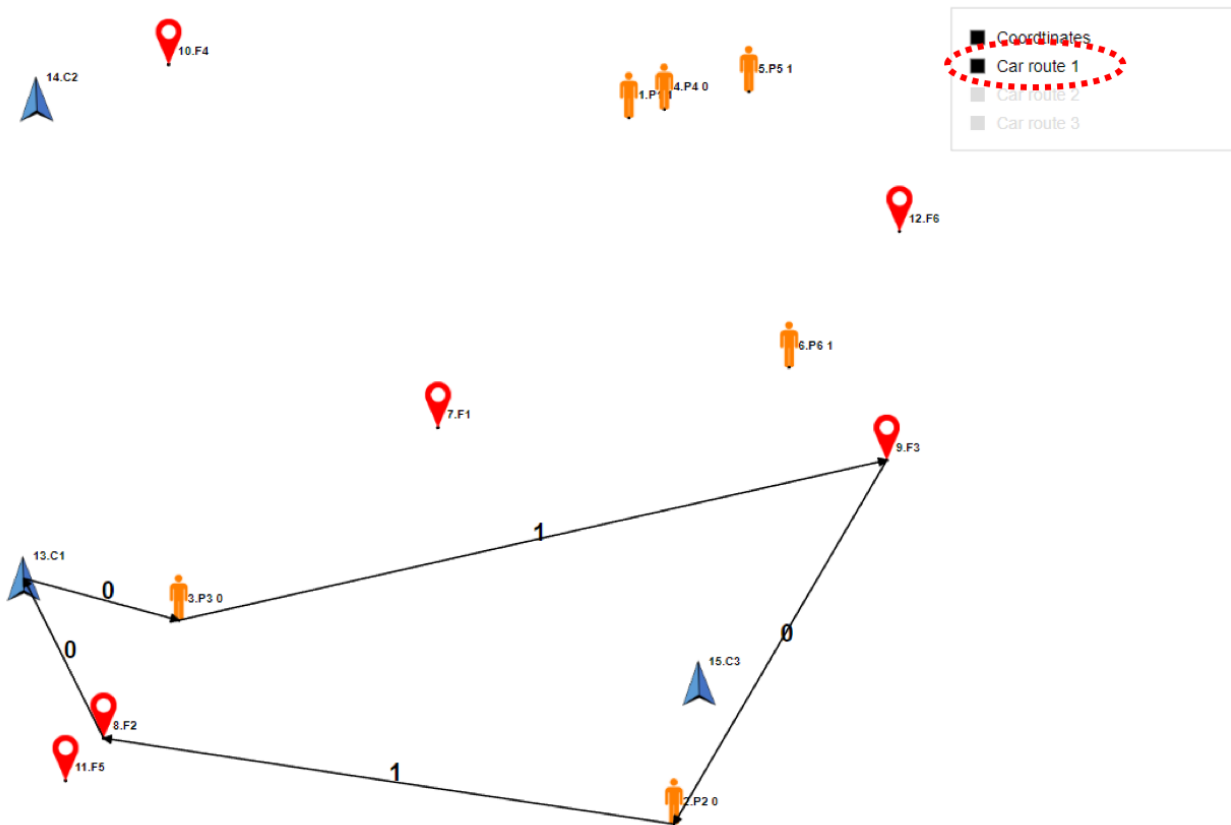


Figure 10 Example of program display with only route 1 selected

2.6 Observations

We used the program to generate a set of random data sets in order to observe the results given by the program to different conditions and using different approaches.

One of the first objectives was to test the performance of the program. The map display allows the user to quickly check if the routes created had any major problems, which could be due to either lack of constraints or, on the contrary, too much constraint restriction. After that, we tested the performance of the simplification constraints. To do that, we run the program using the same data with and without the simplification constraints, and observed that in any of the cases the routes designed were changed, which is something positive. However, after including the simplification constraints, the time required by the program to get to the optimal solution was always reduced. We also noticed a big difference in the running time after selecting a good value for the W in constraint 15. Both things together significantly reduced the processing time.

2.6.1 Comparing minimization approaches

Minimization of cost

In reference to the Minimization of cost approach, the tests performed on random data showed that, in most of the cases, not all the cars were used, and in many of them, the shortest route was performed by only one car which provided the entire service.

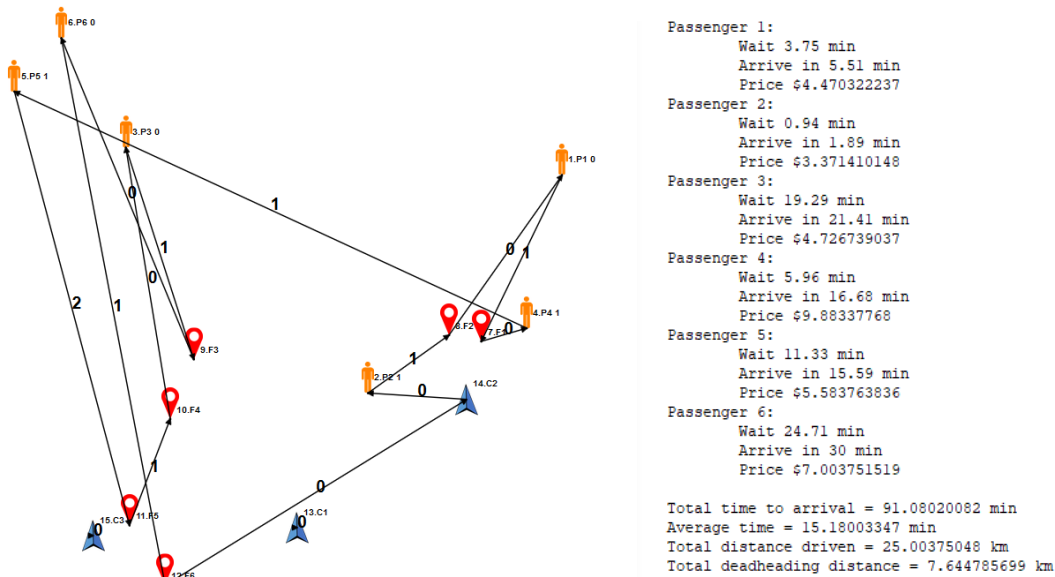


Figure 11 Results from test 10 using a minimization of distance

In the picture above it can be appreciated how, although there are three cars available, the route designed by the program only includes one of the cars, because that is the shortest route. Also, in one of the arcs there are two users in the car, which means that making use of the sharing possibility allowed the driver to take a shorter route. However, due to the lack of any time constraints, this approach often causes for some users to experience very long waits. Since there

aren't any time limitation constrains, there is no limit to the time a customer can wait. In this example, passenger 6 takes 30 minutes to get to its drop-off point, even though there are two more unassigned drivers available. In fact, from the ten different sets of data used with three available cars, in four of them only one of the cars was used, and in the other six, only two cars where used. This indicates that only focusing on minimizing the distance driven is not the best approach for this problem, since cars are not used in an efficient way and customers experience long waits.

Minimization of user discomfort

The second approach studied is the **Minimization of user discomfort**. We run the program using the same initial sets of data as with the previous approach but with a different minimization function. The results showed that, in all the cases, all the cars were used and the routes where optimized so that the users could get to their destination in the minimum time.

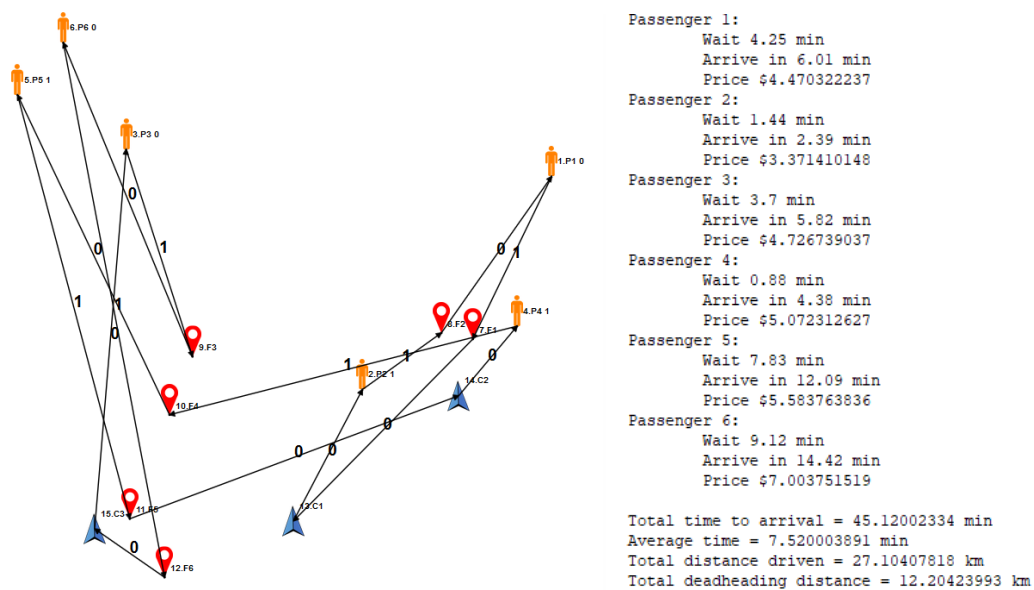


Figure 12 Results from test 10 using a minimization of total time

The figure shows the same example presented in the previous figure but, this time, it has been solved using the first of the discomfort minimization approaches presented: the **minimization of total time (B1)**. The map shows that, in contrast with the previous case, the three cars have been assigned a route. In this case, no users had to share car with other individuals. Comparing the results shown in both cases, the average time waited has decreased from 15 to 7 minutes, which is a significant difference. The biggest improve is received by the passenger 6, whose travel time was reduced from 30 minutes only 14, and at the same price. Price hasn't varied because price doesn't depend on the time waited but on the time and distance inside the car. Also, although total time decreased significantly, the total distance driven only increased by 2km. The result is a better route, since the user discomfort is much lower and cost only a little bit higher. In addition, it is better for the company if more cars are used, instead of one of them doing the entire job. Having too many cars without assigned routes might upset drivers and cause them to stop providing the service for the company.

We generated 20 different data sets to compare the results obtained with the minimization of cost and with the minimization discomfort approaches. The solutions given by the program showed that, on average, the routes obtained when minimizing total time were 36.3% faster, but the distance driven was 8% longer. Deadheading distance also increased in the second case by almost 60%.

The other two variations of the discomfort minimization were also studied: the **minimization of weighted sum of times (B2)** and the **minimization of difference between actual and shortest time (B3)**.

The **Minimization of weighted sum of times** approach provided the expected results. The solutions were similar to the ones obtained with the minimization of total time, but with variations depending on the value of user preference constant (alpha) selected. Big values of alpha prioritize the minimization of waiting time while small values prioritize the minimization of riding time.

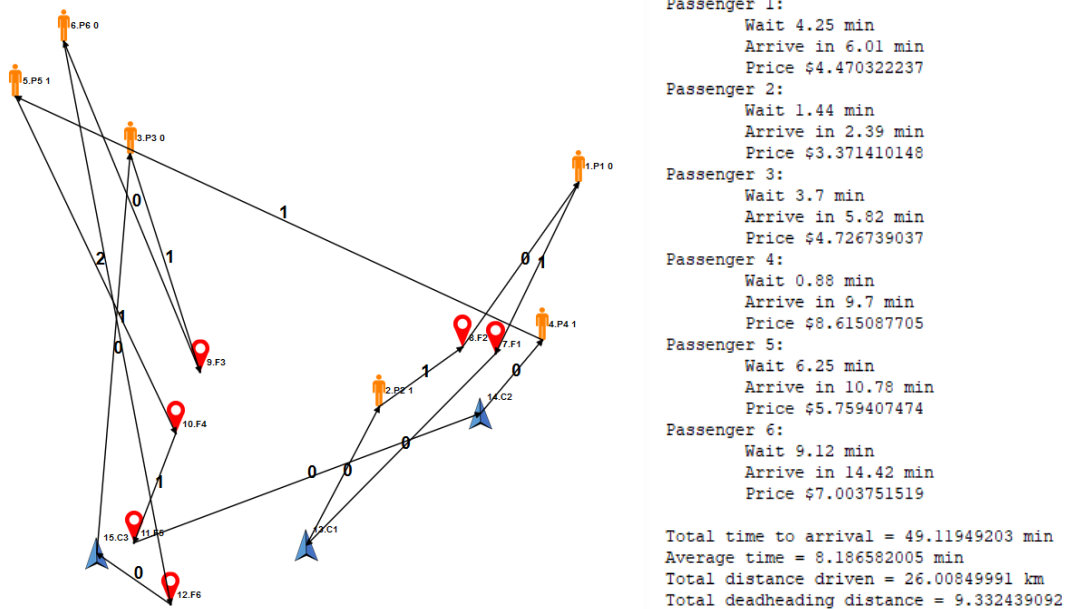


Figure 13 Results from test 10 using a weighted minimization of time

The image shows the same example presented in the two previous cases but using the minimization of weighted times. The alpha selected was 0.9, which gives a high priority to minimizing the waiting time. It can be appreciated that the waiting time was reduced by looking at the deadheading distance, which measures the distance the car travels without passengers. Minimizing waiting time induces the program to pick up customers as soon as possible, reducing the deadheading time. Comparing this image with the previous one shows that deadheading distance was reduced by from 12.2km to 9.3km, and total distance driven was also reduced from 27.1km to 26km. However, minimizing waiting time also caused an increase in the average total time for arrival, from to 7.52 to 8.19 minutes.

On the other side, the **minimization of difference between actual and shortest time** didn't provide new solutions. All the tests provided the exact same solution as the minimization of total

time approach. This led us to realize that both minimizations were actually the same. One of them is the minimization of total time and the other one the minimization of the difference between total time and a constant. These two cases, from a minimization point of view, are the same. However, the idea that actual routes shouldn't be much longer than the shortest path possible is a logical statement. Users whose TNC route takes much longer than the shortest route are likely to select another option. One way to keep this gap small could be to square the differences between actual and shortest time. By doing this, we would ensure that there aren't big differences between those two values. Nevertheless, that would make the solution to stop being linear, so it wouldn't be solved with this program.

Minimization of weighted sum of cost and discomfort

Finally, the last minimization approach presented was the combination of the **minimization of cost and discomfort (C)** through a weighted sum of both. The results obtained testing this approach were, in many cases, the same results obtained using the minimization of time approach. However, in other cases, the results showed routes which had a noticeable smaller distance driven with a negligible increase in the time waited. This indicates that this could be more reliable than the previous ones. It often provides shortest routes that have a smaller cost and a lower impact on the environment while barely increasing user discomfort. And in the worst scenario, where there is no possible improvement, the route selected is simply the same that would be obtained with the minimization of time. The level at which the company is willing to provide a slower service in exchange of a shortest route can be determined with the parameters w_1 and w_2 presented in the code before. The values of w_1 and w_2 selected to perform the tests for this approach were 1 and $1/\text{Speed}$, respectively, which results in a relation of 1km to 1 minute. Meaning that routes that save more km than minutes will be chosen by the program. Also, this usually brings a reduction of the deadheading distance.

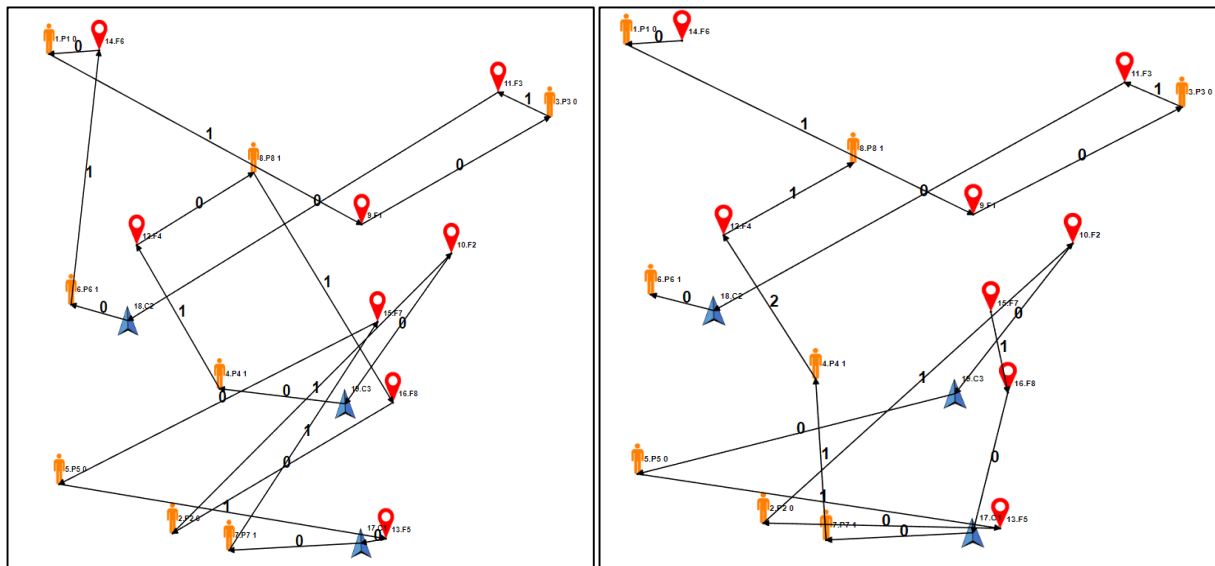


Figure 14 Comparison of the results of test 19 using minimization of Time (on the left) and minimization of both distance and time (on the right)

The figures shows one of the tests carried where the routes designed for the each of the two approaches were different. In this example, the total time increased by 2.7%, but the distance driven decreased by 9.6% and the deadhead distance by 27%.

From 10 tests performed with this approach, six of them resulted in the same route selected by the minimization of total time approach. In the other four, there was a reduction of the number of kilometers driven with a smaller increase in time waited. In those four tests, on average, the total distance driven was reduced by a 6.8% while the time only increased by a 2.14%. Also, the deadheading distance was reduced by an 18%.

2.6.2 Modifying number of cars available

Using the Minimization of car and discomfort, we observed the results obtained after eliminating one of the cars from the data set. This, in practice, could happen if one of the drivers available rejects the request for service. Comparing the results of ten random data sets, we observed that, on average, after removing one of the cars, the arrival time of the users increased by a 30%, however, the total distance increased only by less than 1% and the deadheading time stayed the same. This indicates that the number of cars is not a big determining factor of the total distance driven. Reducing the number of cars only has an impact on the time waited, but the total distance traveled reminds relatively steady. This also matches the results obtained with in the minimization of cost approach, where the shortest assignation of routes included usually only one car. Increasing the number doesn't necessary increase or decrease the total distance neither. Adding more cars can result in either shorter distance, if the new car is placed in a favorable position, or longer distances, if the program assigns a route that is longer but saves time. Furthermore, once the number of cars has surpassed the number of users, the total distance remains very steady since adding more cars only causes for more cars to not get assigned a route.

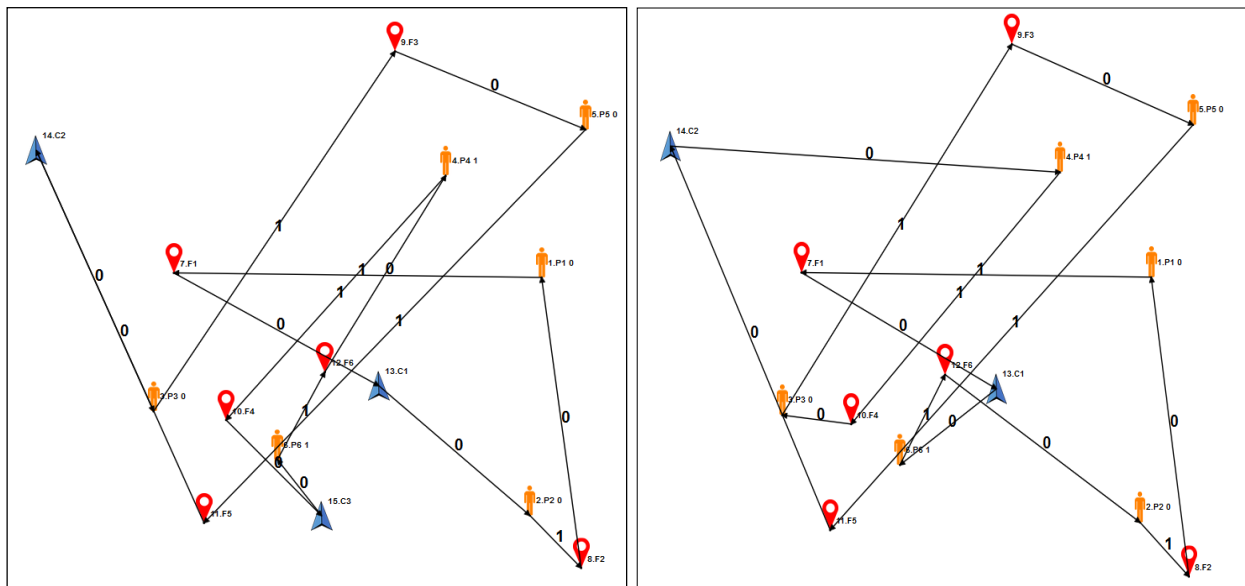


Figure 15 Comparison of the results from test 10 after eliminating one of the cars

2.6.3 Comparing effect of shared trips

Lastly, we used the program to compare the effect that shared trips have on the design of the routes. We compared the routes designed by the program for five different sets of coordinates in two different scenarios: first, with all the requests being for non-shared trips, and then, all of them being for shared trips. The comparison of the two different scenarios showed that, on average, the total arrival time was 7% lower when all users had requested shared trips. Also, the total distance was 20% lower and the deadheading distance was reduced by 41%. This indicates that shared trips are much more efficient than non-shared trips, as they allow for many more possible route combinations, allowing for routes to be shorter and faster at the same time. However, users are not always willing to share the trip with other people. For shared routes to be significantly more efficient, the number of shared requests must be high, since two users won't share if one of them doesn't want to. Something else to mention is that the time needed for the program to reach the optimal solution was much lower when there were no shared requests. That is because shared requests substantially increase the number of possible routes. Here is an example of one of the data run by the program.

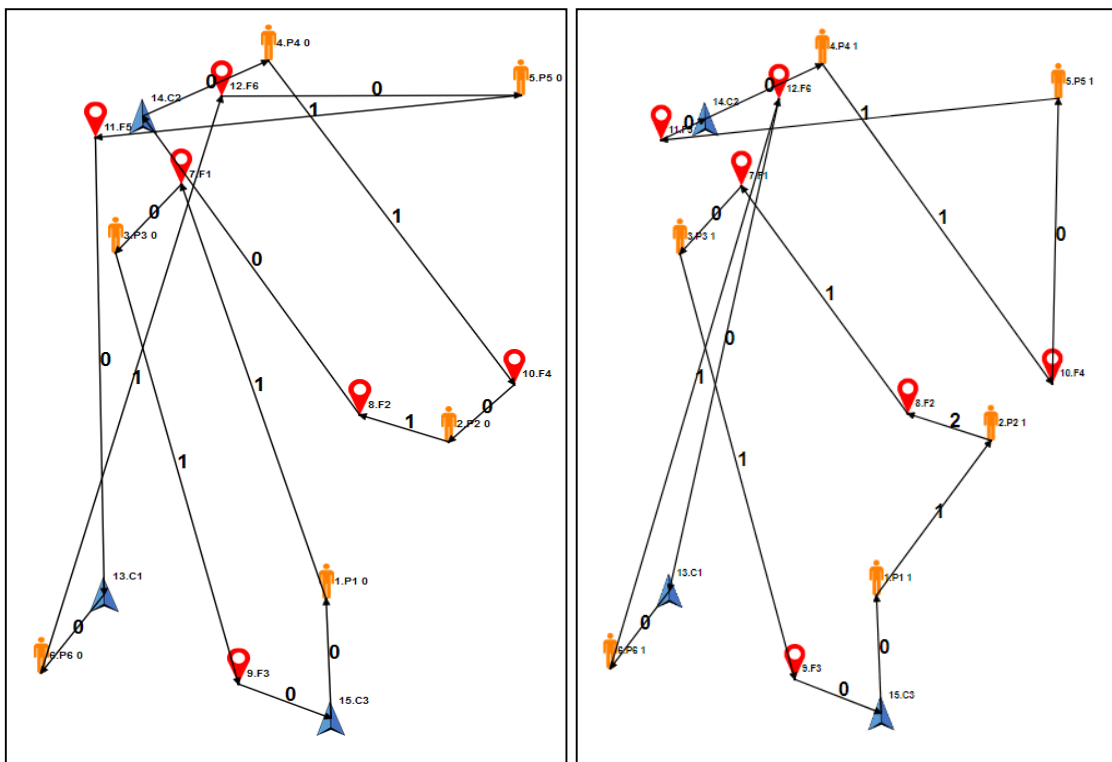


Figure 16 Comparison of the results from test 2 setting all requests to non-shared (on the left), and shared (on the right)

The result from these tests suggest that the sharing system could be a potential way to decrease overall deadheading travelling and carbon emissions, while also reducing total time.

CONCLUSION

TNCs meet the demand for flexible, fast and cheap mobility. The wide range of possibilities it can offer has allowed it to have a big impact on many different transportation systems. In particular, the taxi industry is the most affected sector, and continues to decrease its market share as TNCs gain popularity. This is due to the lower prices, better service, and more effective design of routes of ride-hailing services, thanks to the use of optimization programs, neural networks and new features, like shared rides.

There is inconclusive data about whether TNCs increase the number of cars on the road or, by the contrary, induce people to reduce their car usage and ownership. There is still room for the improvement of TNCs service through the advances in technology. Advances in neural network training can lead to more accurate predictive models. Optimization systems can also be improved to decrease carbon emissions and deadheading travelling. The results obtained from the tests have shown that the most effective way to do that without affecting waiting times is the use of shared trips. If companies promote the use of the shared requests between users they could increase the effectivity of their routes and reduce their impact of congestion and on the environment. On top of that, the minimization approach that provided the best results was the minimization of the sum of costs and user discomfort. By including both terms in the minimization it's possible to obtain routes that keep both length and waiting times low. The results also showed that the number of drivers is not a big determining factor on the total distance needed to provide the service, but it affects the waiting times of the users.

Lastly, some TNCs like Uber and Lyft are already working on the autonomous mobility-on-demand. This new service could bring all the advantages of traditional TNCs service but without the drawbacks. Their introduction is expected to reduce carbon emissions, congestion, accidents and maybe even revolutionize the private car industry. If this service is reliable and fast-responsive, it could decrease the dependence of owning a private car, leading to a decrease of car production, transportation prices, carbon emissions and an increase in parking spaces. For all this to happen, autonomous cars need on-road experience to train their neural networks.

Appendix

Full code

```
!@encoding CP1252
model TFG_Transportation_Network_Companies
uses "mmsxprs"; !gain access to the Xpress-Optimizer solver
uses "mmsvg" !gain access to the graphs plotter

parameters
    !Customizable variables
    N_CARS = 3          !Number of cars
    N_CUSTOMERS = 6    !Number of passengers

    SIZE = 25         !Size of the map in km^2
    CAR_SPEED = 50    !Average car speed in km/h
    SHARE_PROB = 1    !Probability to share ride
    BASE_FARE = 3.21  !Base fare for normal car
    COST_KM = 0.5     !Cost per KM
    COST_MINUTE = 0.3 !Cost per minute
    BASE_FARE_SHARE = 2.74 !Base fare for shared cars
    COST_KM_SHARE = 0.5 !Cost per KM on shared cars
    COST_MINUTE_SHARE = 0.25 !Cost per minute with shared car
end-parameters

declarations

    CUSTOMERS = 1..N_CUSTOMERS
    DESTINATIONS = ((N_CUSTOMERS+1)..(2*N_CUSTOMERS))
    CAR_POSITIONS = ((2*N_CUSTOMERS+1)..(2*N_CUSTOMERS+N_CARS))
    TOTAL_NO_CAR = 1..(2*N_CUSTOMERS)
    TOTAL = 1..(N_CARS+2*N_CUSTOMERS)

    CARS = 1..N_CARS

    COORD: array(TOTAL, 1..2) of real !coordinates x,y of each driver, customer and final destination
    DIST: array(TOTAL, TOTAL) of real !driving distance from i to j
    SHARE: array(CUSTOMERS) of real !Whether a customer wants to share ride or not

    FLOW: array(TOTAL, TOTAL, CARS) of mpvar !Variable that controls: 1.That routes only start at cars
and end up at cars(no impossible loops). 2.Time passed since the start of the model to when the car arrives at a
certain point
    X: array(TOTAL, TOTAL, CARS) of mpvar !Weather the car 'k' goes from 'i' to 'j' or not
    PASSENGERS: array(TOTAL, TOTAL, CARS) of mpvar !Number of passengers car k has when
travelling from i to j

end-declarations

SPEED := CAR_SPEED/60 !Speed in kilometers per minute

forall (i in TOTAL, j in 1..2) !Randomly assigns coordinates to customers, cars and destinations
    COORD(i,j) := 100*random

forall (i in CUSTOMERS) do !Randomly assigns whether customers want to share or not
```



```

    SHARE(i) := random
    if(SHARE(i) <= (1-SHARE_PROB)) then
        SHARE(i) := 0           !If share is 0 then the customer doesn't want to share
    else
        SHARE(i) := 1           !If share is 1 then the customer wants to share
    end-if
end-do

forall (i in TOTAL, j in TOTAL)
    DIST(i,j) := sqrt((COORD(j,1)-COORD(i,1))^2+(COORD(j,2)-COORD(i,2))^2)
    !calculates all distances between points and adjusts it to the size of the map if the parameters

forall (i in TOTAL, j in TOTAL, k in CARS)
    X(i,j,k) is_binary           !makes X binary

!Constrains

forall (i in CUSTOMERS+DESTINATIONS) do
    sum(j in TOTAL, k in CARS)(X(i,j,k)) = 1   !a car must arrive to every customer
    forall(k in CARS)
        sum(j in TOTAL)(X(i,j,k)) = sum(j in TOTAL)(X(j,i,k))   !if a car goes to a customer it must
leave from that customer
    end-do

forall (i in CAR_POSITIONS)
    sum(j in TOTAL, k in CARS)(X(i,j,k)) <= 1 !From every car location only one car can depart. No cars will
depart if that car is not needed

forall(k in CARS)
    sum(j in TOTAL)(X(k+2*N_CUSTOMERS,j,k)) = 1   !each car has to leave from its initial position

forall(k in CARS)
    sum(j in TOTAL)(X(j, k+2*N_CUSTOMERS,k)) = 1   !each car has to end at its initial position

forall(j in DESTINATIONS, k in CARS) do
    sum(i in TOTAL)(X(i, j, k)) = sum(i in TOTAL)(X(i, j-N_CUSTOMERS, k))   !if a car picks up a
customer it must deliver that customer
    sum(i in TOTAL)(FLOW(i,j,k)) - sum(i in TOTAL)(FLOW(i,j-N_CUSTOMERS,k)) >= DIST(j-
N_CUSTOMERS,j)*sum(i in TOTAL)(X(i,j,k)) ! we make sure that a car doesn't deliver a customer before picking
it up
end-do

forall(i in CAR_POSITIONS, j in CAR_POSITIONS, k in CARS| i<>j)
    X(i,j,k)=0
    !Cars should never drive to another car's position. This constrain is not necessary but
helps finding a solution faster

W := sum(i in TOTAL)(max(j in TOTAL_NO_CAR)(DIST(i,j)))
forall(i in TOTAL, j in TOTAL, k in CARS)
    FLOW(i,j,k) <= X(i,j,k)*W           !the flow of a route that is not taken by the car is 0. Flow will never be
higher than W.

forall(k in CARS)
    sum(j in TOTAL)(FLOW(k+2*N_CUSTOMERS,j,k)) = sum(j in
TOTAL)(X(k+2*N_CUSTOMERS,j,k)*DIST(k+2*N_CUSTOMERS,j)) !When cars depart their initial time is 0
so the time when they arrive to their first customer has to be that distance divided by its velocity

```

```

forall(k in CARS, j in TOTAL_NO_CAR) do
    sum(i in TOTAL)(FLOW(j,i,k)) - sum(i in TOTAL)(FLOW(i,j,k)) = sum(i in
TOTAL)(X(j,i,k)*DIST(j,i))      !We use the variable FLOW not only to make sure no loops are created
but also to account for the time it has taken the car to arrive at one location.
end-do

```

```

forall(j in TOTAL_NO_CAR, k in CARS)
    X(j,j,k)=0      !Forbids travels from one point to itself. Without this constrain loops are formed between
points and themselves as, because distance from i to i is 0, the previous condition doesn't restrain it

```

```

forall(k in CARS, j in TOTAL)
    PASSENGERS(k+2*N_CUSTOMERS, j, k) = 0 !The number of passengers a car has when it starts is 0

```

```

forall( j in CUSTOMERS, k in CARS)
    sum(i in TOTAL)(PASSENGERS(j, i, k)) - sum(i in TOTAL)(PASSENGERS(i, j, k)) = sum(i in
TOTAL)(X(i,j,k)) !When a car picks up a customer the number of passengers adds 1

```

```

forall( j in DESTINATIONS, k in CARS)
    sum(i in TOTAL)(PASSENGERS(i, j, k)) - sum(i in TOTAL)(PASSENGERS(j, i, k)) = sum(i in
TOTAL)(X(i,j,k)) !When a car drops a customer the number of passengers subtracts 1

```

```

forall(i in TOTAL, j in TOTAL, k in CARS)
    PASSENGERS(i,j,k) <= 4*X(i,j,k)      !There are passengers only in the routes that
the car drives. Also number of passengers must always be 4 or less.

```

```

forall(k in CARS, j in CUSTOMERS | SHARE(j)=0) do      !For customers that don't want to share:
    sum(i in TOTAL)(PASSENGERS(i,j,k)) = 0      !The car must arrive to them empty
    X(j,j+N_CUSTOMERS,k)= sum(i in TOTAL)(X(i,j,k))      !It must take them directly to their
destination
end-do

```

```

forall(i in CUSTOMERS| SHARE(i)=0) do      !For customers that don't want to share:
    sum(k in CARS)(X(i,i+N_CUSTOMERS,k))=1 ! There is always going to be one car that goes from the
customer to it's destination
    forall(j in TOTAL, k in CARS| j <>(i+N_CUSTOMERS))
        X(i,j,k) = 0      !Arcs that go from the customer to any other point that is not his destination will not be
traversed
end-do

```

```

forall(i in CUSTOMERS, j in CAR_POSITIONS, k in CARS)
    X(i,j,k)=0 ! There is no reason why a car would travel from a pick-up point to a car initial position

```

```

forall(i in CAR_POSITIONS, j in DESTINATIONS, k in CARS)
    X(i,j,k)=0 ! There is no reason why a car would travel the initial position to a drop-off point

```

```

T := sqrt(SIZE)/(SPEED*100)*sum(i in TOTAL, j in DESTINATIONS, k in CARS)(FLOW(i,j,k)) !Time=
Distance/Speed and then multiplied by sqrt(SIZE)/100 to scale the coordinates to the size of the map selected
D := sqrt(SIZE)/100*sum(i in TOTAL, j in TOTAL_NO_CAR, k in CARS)(X(i,j,k)*DIST(i,j)) !Total distance
driven by cars to customers and destinations and scaled to the size of the map
minimize(T) !You can either minimize(T) to minimize time or minimize(T+D) to get a small time but decreasing
deadheading too or even minimize(D) to minimize distance.

```

```

!Writes results
forall(j in CUSTOMERS) do
    writeln("Passenger ", j, ":")
    KM_PICKUP := sum(k in CARS, i in TOTAL)(getsol(FLOW(i,j,k)))*sqrt(SIZE)/(100) !Calculate the
distance from car's initial position to pick up and scale it to KM. Coordinates go from 1 to 100 so we divide by 100
to get a percentage of the map. Then we multiply by map size to make distances proportional to the map size.
    KM_ARRIVAL := sum(k in CARS, i in TOTAL)(getsol(FLOW(i,j+N_CUSTOMERS,k)))
*sqrt(SIZE)/(100) !Calculate distance driven from car's initial position to arrival
    WAIT := KM_PICKUP/SPEED !Time passed since the car is ordered to when it picks up the customer
    ARRIVAL := KM_ARRIVAL/SPEED !Time passed since the car is ordered to when it drops the
customer off
    PRICE := (1-SHARE(j))*(BASE_FARE + COST_KM*(KM_ARRIVAL-KM_PICKUP)+
COST_MINUTE*(ARRIVAL-WAIT)) + (SHARE(j))*(BASE_FARE_SHARE +
COST_KM_SHARE*(KM_ARRIVAL-KM_PICKUP)+COST_MINUTE_SHARE*(ARRIVAL-WAIT))
    writeln("Wait " , round(WAIT*100)/100, " min")
    writeln("Arrive in " , round(ARRIVAL*100)/100, " min")
    writeln("Price $" , PRICE)
end-do

writeln("")
writeln("Total time to arrival = ", getsol(T), " min")
writeln("Average time = ", (getsol(T)/N_CUSTOMERS), " min")
writeln("Total distance driven = ", sqrt(SIZE)/100*sum(i in TOTAL, j in TOTAL_NO_CAR, k in
CARS)(getsol(X(i,j,k))*DIST(i,j)), " km")
writeln("Total deadheading distance = ", sqrt(SIZE)/100*sum(i in TOTAL, j in TOTAL_NO_CAR, k in CARS|
getsol(PASSENGERS(i,j,k)=0)(getsol(X(i,j,k))*DIST(i,j)), " km")

writeln("")
writeln("COORDINATES:[")
forall(i in TOTAL)
    writeln(COORD(i,1), "    ", COORD(i,2))
writeln("]")
writeln("SHARE:[")
forall(i in CUSTOMERS)
    writeln(SHARE(i))
writeln("]")
!Display visual result
svgsetgraphviewbox(-10,-10,120,120) ! Graph size
svgsave("Carsmap.svg") ! Save graphic to file
svgaddfile("PASSENGER2.png", "CUSTOMER")
svgaddfile("DESTINATION.png", "DESTINATION")
svgaddfile("CAR5.png", "CAR")

svgaddtext(20,90, "") ! Tittle
t:=svggetlastobj
svgsetstyle(t, SVG_FONTSIZE, "6pt") ! Size
svgsetstyle(t, SVG_FONTSTYLE, "oblique") ! Font
svgsetstyle(t, SVG_FONTWEIGHT, "bold") ! Makes it bold

!Draws positions of cars and customers

svgaddgroup("CIRCLES","Coordinates ", SVG_BLACK) !Maybe make that each customer's start and finish is the
same color?

```

```

forall(i in TOTAL) do
  svgaddcircle(COORD(i,1),COORD(i,2), 0.1) !Creates a circle wherever there is a coordinate
  svgsetstyle(svggetlastobj, SVG_STROKEWIDTH, 0.2) ! Wider border
  if(i<=N_CUSTOMERS) then
    ! For
customer locations:
    svgsetstyle(svggetlastobj, SVG_FILL, SVG_RED) ! Fills circle red
    svgaddimage("CUSTOMER", COORD(i,1)-2.5,COORD(i,2), 5, 5)
    svgaddtext(COORD(i,1)+1, COORD(i,2)+2, "" + i + ".P" + i + " " + SHARE(i)) ! Adds a label to
each customer and whether it shares or not
    end-if
    if (i>N_CUSTOMERS and i<=2*N_CUSTOMERS) then ! For customer destinations:
      svgsetstyle(svggetlastobj, SVG_FILL, SVG_GREEN) ! Fills circle green
      svgaddimage("DESTINATION", COORD(i,1)-2.5,COORD(i,2), 5, 5)
      svgaddtext(COORD(i,1)+1, COORD(i,2)+1, "" + i + ".F" + (i-N_CUSTOMERS)) ! Adds a label
to each destination
    end-if
    if (i>2*N_CUSTOMERS) then !For each Car
      svgsetstyle(svggetlastobj, SVG_FILL, SVG_BLUE) !Fills circle blue
      svgaddimage("CAR", COORD(i,1)-2.5,COORD(i,2)-2.5, 5, 5)
      svgaddtext(COORD(i,1)+1, COORD(i,2)+2, "" + i + ".C" + (i-2*N_CUSTOMERS)) ! Adds a
label to each car
    end-if
    svgsetstyle(svggetlastobj, SVG_FONTSIZE, "1pt") !Size of the labels names
    svgsetstyle(svggetlastobj, SVG_FONTWEIGHT, "bold") !Makes the labels bold
  end-do

forall(k in CARS) do
  svgaddgroup("CAR_ROUTE"+k, "Car route "+ k, SVG_BLACK)
  forall(i in TOTAL, j in TOTAL| getsol(X(i,j,k))>0.1) do !For every trip existing between i and j
    svgaddarrow(COORD(i,1),COORD(i,2),COORD(j,1),COORD(j,2)) !Draws the line
between the coordinates of i and j
    svgsetstyle(SVG_STROKEWIDTH, 0.2) !Wider line
    svgaddtext((COORD(i,1)+COORD(j,1))/2, (COORD(i,2)+COORD(j,2))/2, ""+
getsol(PASSENGERS(i,j,k)))
    svgsetstyle(svggetlastobj, SVG_FONTSIZE, "2pt") !Size of the labels names
    svgsetstyle(svggetlastobj, SVG_FONTWEIGHT, "bold") !Makes the labels bold
  end-do
end-do

svgrefresh ! Display graphic
svgwaitclose ! Wait until window is closed

end-model

```

Test 11	t.time	39.7341	-0.35779092	25.5176	0	25.5176
n=8	av.time	4.96677	-0.35779188	3.1897	0	3.1897
m=4	T.Dist	223.801	0.042162457	233.237	0	233.237
Share = 0.5	DH.Dist	71.0028	0.132883492	80.4379	0	80.4379
Test 12	t.time	73.7606	-0.21070599	58.2188	0	58.2188
n=8	av.time	9.22008	-0.21070641	7.27735	0	7.27735
m=4	T.Dist	541.064	0.010209513	546.588	0	546.588
Share = 0.5	DH.Dist	150.349	0.036741182	155.873	0	155.873
Test 13	t.time	125.541	-0.44836269	69.2531	0	69.2531
n=8	av.time	15.6927	-0.4483658	8.65663	0	8.65663
m=4	T.Dist	649.932	0.115750571	725.162	0	725.162
Share = 0.5	DH.Dist	196.285	0.383274321	271.516	0	271.516
Test 14	t.time	108.019	-0.49009063	55.0799	0.0081409	55.5283
n=8	av.time	13.5024	-0.49009139	6.88499	0.0081409	6.94104
m=4	T.Dist	516.208	0.122946564	579.674	-0.0508199	550.215
Share = 0.5	DH.Dist	162.742	0.389979231	226.208	-0.1302341	196.748
Test 15	t.time	154.479	-0.47847021	80.5654	0	80.5654
n=8	av.time	19.3099	-0.47846959	10.0707	0	10.0707
m=4	T.Dist	700.611	0.089349154	763.21	0	763.21
Share = 0.5	DH.Dist	221.401	0.28273585	283.999	0	283.999
Test 16	t.time	90.3499	-0.4190796	52.4861	0	52.4861
n=8	av.time	11.2937	-0.41907789	6.56076	0	6.56076
m=4	T.Dist	518.437	0.040188104	539.272	0	539.272
Share = 0.5	DH.Dist	133.012	0.156640002	153.847	0	153.847
Test 17	t.time	68.14	-0.30871001	47.1045	0	47.1045
n=8	av.time	8.5175	-0.3087103	5.88806	0	5.88806
m=4	T.Dist	401.493	0.10253977	442.662	0	442.662
Share = 0.5	DH.Dist	63.0533	0.72114703	108.524	0	108.524
Test 18	t.time	120.501	-0.40839495	71.289	0	71.289
n=8	av.time	15.0626	-0.4083943	8.91112	0	8.91112
m=4	T.Dist	571.377	0.120622636	640.298	0	640.298
Share = 0.5	DH.Dist	99.905	0.501366298	149.994	0	149.994
Test 19	t.time	66.8065	-0.09925082	60.1759	0.0273897	61.8241
n=8	av.time	8.35081	-0.09925025	7.52199	0.02738903	7.72801
m=4	T.Dist	523.969	0.126106697	590.045	-0.0963198	533.212
Share = 0.5	DH.Dist	160.573	0.448506287	232.591	-0.2698944	169.816
Test 20	t.time	63.1092	-0.16227903	52.8679	0.02526675	54.2037
n=8	av.time	7.88865	-0.16227998	6.60848	0.02526905	6.77547
m=4	T.Dist	508.049	0.09103059	554.297	-0.0491975	527.027
Share = 0.5	DH.Dist	114.38	0.404336422	160.628	-0.1697711	133.358
Test 21	t.time	85.4191	-0.31388998	58.6069	0	58.6069
n=8	av.time	10.6774	-0.31389102	7.32586	0	7.32586
m=4	T.Dist	570.51	0.006643179	574.3	0	574.3
Share = 0.5	DH.Dist	174.736	0.021689864	178.526	0	178.526
Test 22	t.time	75.6867	-0.13347259	65.5846	0.02504094	67.2269
n=8	av.time	9.46083	-0.13347138	8.19808	0.02504001	8.40336
m=4	T.Dist	591.643	0.120939485	663.196	-0.0780433	611.438
Share = 0.5	DH.Dist	169.655	0.305608441	221.503	-0.1625847	185.49
Average test 11-22			-0.31920812		0.00715319	
			-0.31920835		0.00715325	
			0.08237406		-0.022865	
			0.315409035		-0.0610404	
TOTAL AVERAGE			-0.36324231			
			0.102007535			
			0.598103687			

Bibliography

- Anderson, D. N. (2014). 'Not just a taxi'? For-profit ridesharing, driver strategies, and VMT. *Transportation*, 41, 1099-1117.
- Barter, P. (2013, February). "Cars are parked 95% of the time". Let's check! *Reinventing parking*.
- Bertoncello, M., & Wee, D. (2015, June). Ten ways autonomous driving could redefine the automotive world. *McKenssey & Company*.
- Bodin, L. D., & Sexton, T. (1986). The multi-vehicle subscriber dial-a-ride problem. . *TIMS Studies in Management Science*, 2, 73-86.
- Borndörfer, R., Klostermeier, F., Grötschel, M., & Küttner, C. (1997). Telebus Berlin: Vehicle Scheduling in a Dial-a-Ride System. *Konrad-Zuse-Zentrum für Informationstechnik Berlin, Technical report SC 97-23*.
- Brodeur, A., & Nield, K. (2016). Has Uber made it easier to get a ride in the rain? *Dept. Econ., IZA Inst. Lab. Econ., Bonn, Germany, IZA, 9986*.
- Certify. (2015). *Sharing Economy Q2 Report. Room for More: Business Travelers Embrace the Sharing Economy*.
- Cordeau, J.-F., & Laporte, G. (2003a). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research B*, 37, 579–594.
- Cordeau, J.-F., & Laporte, G. (2007, May 5). The dial-a-ride problem: models and algorithms. *Springer Science+Business Media, LLC, 153*, 34.
- Cramer, J., & Krueger, A. B. (2016). Disruptive change in the taxi business: The case of Uber. *Amer. Econ. Rev*, 106 no.5, 177-182.
- Desrosiers, J., Dumas, Y., & Soumis, F. (1986). A dynamic programming solution of the large-scale single- vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6, 301–325.
- Desrosiers, J., Dumas, Y., Soumis, F., Taillefer, S., & Villeneuve, D. (1991). An algorithm for mini-clustering in handicapped transport. *Les Cahiers du GERAD, G-91-02*.
- Diana, M., & Dessouky, M. M. (2004). new regret insertion heuristic for solving large-scale dial-a-ride problems with time windows. *Transportation Research Part B*, 38, 539–557.
- Dumas, Y., Desrosiers, J., & Soumis, F. (1989a). Large scale multi-vehicle dial-a-ride problems. *Les Cahiers du GERAD, G-89-30*.
- Erhardt, G. D., Roy, S., Cooper, D., Sana, B., Chen, M., & Castiglione, J. (2019, May). Do transportation network companies decrease or increase congestion? *Science Advances* , 5 no.5.
- Geier, B. (2015, June). 10% of All Uber Rides Happen in China. *Time*. Retrieved from <http://time.com/3914378/uber-china>.
- Greenwood, B., & Wattal, S. (2015). Show me the way to go home: An empirical investigation of ride sharing and alcohol related motor vehicle homicide. *Fox School of Business Research Paper, 15-054*. Retrieved from http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2557612
- Ioachim, I., Desrosiers, J., Dumas, Y., & Solomon, M. M. (1995). A request clustering algorithm for door-to- door handicapped transportation. *Transportation service*, 29, 63-78.
- KPGM. (2013). Self-Driving Cars: Are We Ready? .

- KPMG. (2014). Me, my car, my life. *KPMG LLP*.
- Lyft. (2019). Retrieved from <https://www.lyftimpact.com/analysis/casestudy/0>
- Lyft. (2019). *Lyft*. Retrieved from <http://level52019.wpengine.com/partners/>
- Melachrinoudis, E. I. (2007). A dial-a-ride problem for client transportation in a health-care organization. *Computers & Operations Research*, *34*, 742–759.
- National Academies of Sciences, Engineering, and Medicine. (2016). Between Public and Private Mobility: Examining the Rise of Technology-Enabled Transportation Services. . *The National Academies Press*.
- Ngo, V. (2015, October). Transportation Network Companies and the ridesourcing industry. A Review of Impacts and Emerging Regulatory Frameworks for Uber. *The University of British Columbia School of Community and Regional Planning*.
- Orman, A. J., & Williams, H. P. (2004). A survey of different integer programming formulations of the travelling salesman problem. *LSEOR*, *04.67*.
- Psaraftis, H. N. (1980). A dynamic programming approach to the single-vehicle, many-to-many immediate request dial-a-ride problem. *Transportation Science*, *14*, 130-154.
- Psaraftis, H. N. (1983). An exact algorithm for the single-vehicle many-to-many dial-a-ride problem with time windows. *Transportation Science*, *17*, 351–357.
- Rayle, L., Shaheen, S., Chan, N., Dai, D., & Cervero, R. (2014). App-based, on-demand ride services: Comparing taxi and ridesourcing trips and user characteristics in San Francisco. *Berkeley: University of California Transportation Center*.
- Rekiek, B., Delchambre, A., & Saleh, H. A. (2006). Handicapped person transportation: an application of the grouping genetic algorithm. *Engineering Applications of Artificial Intelligence*, *19*, 511-520.
- Sexton, T. (1979). The single vehicle many-to-many routing and scheduling problem. *SUNY at Stony Brook*.
- Sexton, T., & Bodin, L. D. (1985b). Optimizing single vehicle many-to-many operations with desired delivery times: II. Routing. *Transportation Science*, *19*, 411-435.
- Shaheen, S., Chan, N., & Rayle, L. (2017, Spring). Ridesourcing's Impact and Role in Urban Transportation. *Acces magazine*.
- Taylor, A. (2014). How the Anti-Uber Backlash is Spreading Around the World. *The Washington Post*.
- Uber Technologies Inc. . (2019). *Uber*. Retrieved from Uber Newsroom: <https://www.uber.com/en-PK/newsroom/company-info/>