Department of Mechanical Engineering
School of Engineering
University of Thessaly

Dimitrios Rizopoulos' diploma thesis with title

A generic approach for the synchronization of public transport services

Volos, Greece

August 2018

Supervising professor and thesis advisor: Dr. Georgios K.D. Saharidis

Members of the examination committee:

First member: Dr. Georgios K.D. Saharidis

Assistant professor at the Department of Mechanical Engineering, University of Thessaly

Second member: Dr. Georgios Liberopoulos

Professor at the Department of Mechanical Engineering, University of Thessaly

Third Member: Dr. Dimitrios Pandelis

Associate professor at the Department of Mechanical Engineering, University of Thessaly

The current thesis has been conducted in the frame of the Postgraduate Master of Science program of studies with the name "State-of-the-art Design and Analysis Methods in Industry". The approval of this thesis by the Department of Mechanical Engineering of the School of Engineering of the University of Thessaly does not suggest the acceptance of the views depicted in this thesis by the author.

Credits & acknowledgements

I would like to thank my supervising professor Dr. Georgios K.D. Saharidis for all his help and guidance that he has given me in the past few years. I would like to also express my gratitude to the members of the examination committee, Dr. Dimitrios Pandelis and Dr. Georgios Liberopoulos.

I would like to thank all of the professors at the Department of Mechanical Engineering and especially the professors at the Division of Production Management & Industrial Administration, for teaching me how to approach and work on bigger problems.

Finally, I would like to thank my family and friends who have supported throughout my pursuits.

"Some people see the glass half full. Other see it half empty. I see a glass that's twice as big as it needs to be."

- George Carlin

A generic approach for the synchronization of public transport services

Dimitrios Rizopoulos

Supervising professor: Dr. Georgios K.D. Saharidis

Volos, Greece

August 2018

Abstract

The current thesis analyzes the problem of the rescheduling of public transportation services in such a way that the several beneficiaries of system benefit from the new schedule of operation for the services. A generic approach to the problem, that is applicable to wide set of available data, was developed and is being introduced in the thesis. The approach presented is based on a Mixed Integer-Linear Programming (MILP) formulation. In respect, in the first chapters of this thesis we focus on generic notions of the problem and fundamental aspects of mathematical optimization and transportation, and progressively, in the last chapters we present our contributions to the problem. Finally, a case study is given that focuses on the analysis and improvement of Athens Metro system. With the objective of minimizing waiting times, our approaches achieve reduction of up to 10% in total waiting time, according to the degree up to which we decide to intervene with the current schedule of operation.

Contents

Table of figures

1 Introduction

In the last few decades public transportation systems have improved in more ways than imaginable. Not only they have become faster and cheaper, but they have also become safer and better organized. Mobility services offered nowadays have made the world more interconnected than ever before. Although such systems have facilitated the everyday life of more and more people, negative effects, such as the pollution of the environment and the dysfunctionality of their use, have emerged. Those effects not only have disrupted the proper use, function and stability of those transportation systems, but have also affected in a negative manner other system related to them.

For the improvement of transportation systems, the design and strategic planning of their networks has become an immense need. However, identifying the priorities for each beneficiary of the system, that will in turn lead to their improved welfare, is a trivial question and has become of great importance for societies worldwide. The network's infrastructure and the rules according to which that infrastructure is used, is a topic of heated debate.

Under the umbrella of that debate, falls the problem that is discussed in this paper, the problem of synchronization of public transport services. Although mobility services have connected more and more people each year, until this day, services that directly connect every possible pair of origin and destination are not yet available. Even in cases where a transportation system may be really well-developed and modern, passengers' travel patterns greatly vary and their needs are hard to be met. Thus, *co-modality*, the use of multiple modes of transport through a combined use of services is necessary for the attainment of the transition from one place to another. While more types of services (i.e. air, bus, rail, maritime transport) are becoming available and more operators (i.e. state, private, publicly owned companies) are providing their services, the overall performance of the networks does not necessarily improve, due to lack of proper planning and co-ordination between the several providers. As a result, the level of advancement in the quality of the services may sometimes not match the level of satisfaction of the passengers by the services offered. As for example, in the case of some European public transport networks, and according to a European Commission study [1] in 2014, more than 3 out of 10 respondents in 7-member states of the European Union (EU) never use public transport services in their lifetime. That fact can be partially attributed to non-existent connections between desired places of origin and destination due to lack of synchronization.

Thus, there is the need for centralized design of the network and re-arrangement in such a way that the varied services offered comply with each other. The establishment of a synchronization mechanism among all those services is necessary for the better operation of those networks and in order to meet the needs of citizens. Very important for that coordination mechanism will be the exploitation of scientific literature, associated modelling and solution approaches for the problem of *synchronization* or *timetable design* of public transport services. In our case of study, we will focus on previous theoretical representations (i.e. models) of such problems and methodologies that have been adopted in order to apply those representations to

real-world cases. We will then introduce some instances of the problem, and corresponding approaches, which have important similarities but also differences with works that exist in the literature. In respect, in *Section 2* we refer to the transportation problems found in the literature and associated ways of modelling of such problems. In *Section 3,* we present the problem of rescheduling, its definition and important terms as well as a literature review. In *Section 4*, basic instances of the problem and the main approach developed for this thesis are introduced. In *Section 5* the results concerning the several case studies are given. Finally, in *Section 6* a Conclusion is given where we discuss our findings and possible future research.

2 Transportation problems and their associated ways of modelling

2.1 Transportation problems addressed in the scientific literature

As it is apparent, optimization of transportation systems is essential for businesses and organizations worldwide. Developing models of transportation problem and associated algorithmic techniques for their solution is of great importance nowadays. Although this thesis deals with the problem of rescheduling, the authors deem that references to a wider set of transportation problems is essential for the better understanding of the topic.

To begin with, there is the wide category of problems called *routing problems*. The basis for routing problems is the *Shortest Path Problem* (SPP), where, given two points of a network, the optimal path between those two points needs to be calculated as a solution to the problem. The SPP is a problem that also can be found in many fields other than transportation, such as manufacturing, logistics, biology, astronomy and computer network problems. In many cases the term *shortest* may not refer strictly to kilometric distance, but can be defined according to several criteria. Example given, distance may be computed according time difference in arrival at the destination node. In this case we may refer to the problem as a time-dependent SPP.

Similarly to the SPP, there are two other groups of problems that can be thought of as direct derivatives to the SPP and have been addressed widely in the literature. They are often referred to as *node covering problems*. The first one is the *Travelling Salesman Problem* (TSP). Its objective is to calculate a path through a given network that navigates a salesman (i.e. vehicle, passenger, object to be routed) throughout the network, while he or she visits a set of nodes of the network. After the visiting the last node in the set, the salesman returns to his initial stop. The other fundamental problem is the *Vehicle Routing Problem* (VRP), which is also considered to be the basis for many transportation problems that have to do with the transportation of commodities. It is considered as a direct derivative of the SPP and the TSP problem. For its solution this problem requires the calculation of optimal paths throughout a given network in order for a fleet of identical vehicles, with specific capacity, to achieve a distribution of commodities (i.e. products) to several customers at different locations in that network. Very important to note are the extensions of those problems that have to do with the service of nodes. Many times, the nodes refer to customers, or other beneficiaries of the network, that have specific schedules. In respect the routing that occurs may be subject to Time-Windows (TW) which impose specific constraints of operation for the objects to routed. In regard, the *TSP-TW* and *VRP-TW* problem emerge.

Another set of problems that belong to the category of routing problems are *Edge Covering Problems* (ECP) (i.e. route inspection problem, Chinese postman problem) whose solution is a path that covers a set of edges (i.e. connections between nodes) of the network. That set of nodes, in many cases, like in the Chinese postman problem, includes every edge of the network and the path which constitutes the solution needs to be Eulerian, meaning that it

traverses each edge of the network only once. Such kinds of problems are often being used in urban environments in order to route snow plowing and garbage trucks.

Now, for both node and edge covering groups of problems certain extensions exist that have to do with the types of fleets that are needed to be routed in specific applications. According to the attributes of the vehicles the problems may be extended to be heterogenous fleet problems. Also, according to how many depots we do have for the vehicles, the TSP, VRP and ECP, can be extended to multi-deport instances of problems where further constraints have to be applied to the modelling.

A final category of problems that falls under the umbrella of routing problems are *Journey Planning Problems* (JPP) that focus on navigating passengers throughout a network. While simple instances of the journey planning problem focus on unimodal networks, other modern approaches focus on Multi-modal networks that include services from several providers for land, air and sea transport. Also, the *activity chain problems* have lately emerged in the literature, where special focus is being given to the criteria of the passenger and the activities that he or she needs to do. The routing occurs mostly based on the priorities set by the passenger rather than the attributes of network (i.e. distance between points of interest, traffic, modes of transport available).

Moving on to other types of problems, the *Maximum-flow* and *Minimum-flow problems* are two very important problems. Although they are mostly used to address airline scheduling and distribution of goods problems, the incorporation of flow into several other cases of transportation problems has enabled the modelling of demand. Also, flow variables have been used to model interaction between travelers and *traffic*. Furthermore, the *facility location problem* is another problem that has been extensively studied in the literature. In this case, it is considered that for the solution of the problem, the optimal location for a depot, or any other kind of facility, needs to be determined in order for operations run in a more optimal way.

Finally, we will make a reference to the *rescheduling* or *timetable design* problems, which are time-dependent problems and have to do with the calculation of an optimal schedule that will serve the beneficiaries of the system in a better way. In some cases, that rescheduling happens in a way that the operators of the network benefit or, on the contrary, the re-design of the timetable will provide better operation for the passengers and users of the network. Let's not forget to mention that the problem of *rescheduling* is the main topic of this thesis.

2.2 Modelling techniques for transportation problems

So far, we have described the basic instances of transportation problems. Those kinds of problems lay at the foundation of every navigation system that is used in the real world, ranging from software that aids drivers to cut through traffic in a big city or professionals and companies who need to make the transportation of goods faster and cheaper. Also, other transportation problems also yield solutions to infrastructure and strategic problems that have to do with the placement of facilities and the distribution of commodities. Throughout the

literature, several attempts have been made in order to model the several instances of transportation problems with graph and mathematical modelling techniques being the ones that have been used the most. In the next few paragraphs we will first discuss graph and the associated techniques and then we will discuss mathematical modelling.

Graphs themselves are mathematical structures that are used to represent the relations between a set of objects. Their applications range from mathematics to academic disciplines such as psychology and sociology. The components of graphs are sets of Vertices and sets of Edges, with former being represented by *V* and the latter represented by *E.* In the context of the current thesis, we represent a graph structure with *G (V, E)*. When graph *G* is used for the route planning case, its vertices represent the locations or facilities that the object to be navigated will possibly visit or use and the edges *E* represent the connection from one vertex to another. Usually, according to some criteria those connections are related to a cost.

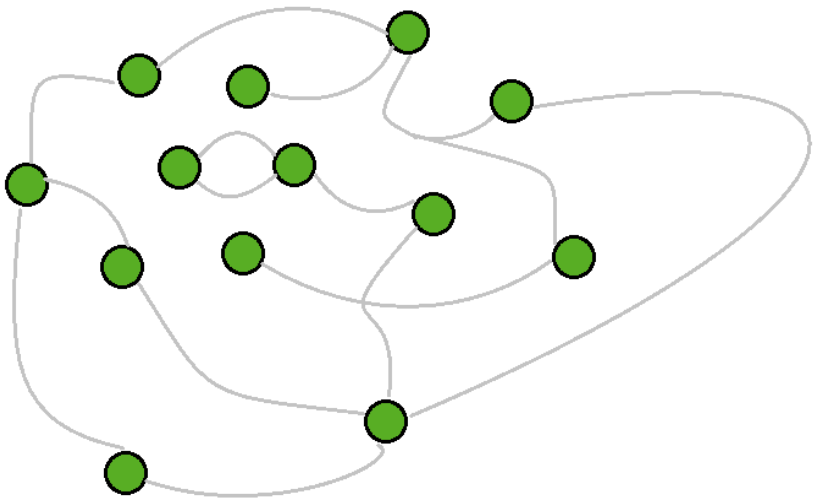

Figure 1: A graph G (V, E) that can be used to represent a transportation network. In this case the green dots or vertices can represent different cities and the grey lines or edges can represent the relationship between the vertices, which is the distance in most cases. Another example, which is totally different from route planning, could be a network of people, with each individual being represented by the vertices and the relationship between them (friends, co-workers, family, etc.) being represented by the edges.

The reason for using graphs in route planning is because of their robustness and flexibility. They have been proved to work well for a large number of applications and different kinds of datasets. Graphs can be used for small applications up to huge navigation systems with intercontinental coverage. As mentioned in the work of Bast et al. [2], the researchers have tested them against enormous datasets that produced graphs with 18 million vertices and 42.5 million directed arcs (edges). Of course, variations exist in their performance. There are many different ways to implement them and many ways to search for optimal solutions in them. *Graphs algorithms* are the *search tools* used on the graphs in order to get to the best combination of segments that lead from the starting vertex to the ending one. Before continuing on the

explanation of other techniques, we would like to remind to the reader that graphs are very abstract notions. As mentioned before, they are used in many fields that are not related to computer science or route planning. However, in the context of this thesis, when we refer to an approach as a *graph approach*, we refer to approaches that are based on *graph data structures,* meaning that the implementers of the approach, create structures in the memory of the computer that work like graphs and allow faster searches due to specific attributes.

Next, let's make a reference to Linear Programming (LP) and Mixed-Integer Linear Programming (MILP). Those two methods of applied mathematics are used in a big variety of applications. MILP is mainly used in route planning because it is considered as the go-to approach for VRP problems. The idea of a MILP approach towards a VRP was first introduced more than 50 years ago [3]. MILP modelling techniques allowed researchers and industry experts to take better management decisions when it comes to vehicle routing. In its essence, MILP is more flexible by allowing the modeler to represent decisions as mathematical entities and express the constraints and relations between those decisions by mathematical equations, equalities and inequalities. Below we provide the standard format for a LP model:

Maximize		$c \cdot x$
(1)		
Subject to		$A \cdot x \leq b$
(2)		
	$x \geq 0$	
(3)		

Moving to other types of problems, although recently there has been conducted some relative research by Saharidis et al. [4], there has been no extensive research of possible outcomes of MILP formulations of route planning on public transit networks. That is not the case though with applications of MILP in TSP problems. While there are many variations and many ways to model the TSP, the most common approach to such problems is MILP. As referred to in the work of Saharidis et al. [5], the TSP is an assignment problem, where each connection is represented by a decision variable, while we make sure that we introduce to the model the *sub-tour elimina*tion constraints. By the way, sub-tour elimination constraints are also present in the work that we will present later on, and are a common underlying trait of the TSP, VRP and MMRP, when modelled with LP or MILP.

Considering the taxonomy of the approaches, all of the techniques used in the field of Optimization, and those that will be developed and referred to in the thesis, fall under one of three categories of approaches. Those categories are exact approaches, heuristic approaches or hybrid ones. The main criterion that distinguishes one approach from another is whether that approach searches extensively all candidate solutions to a problem or not. On the one hand, we will build *heuristic approaches* when we want approximate solutions or calculate approximations of solutions. What that means is, that the answer that we calculate for the problem that is under research is not the absolutely optimal one, but it is close to it. In other words, we use heuristic methods because the application of the solution in the real world, is not required to be globally optimal, meaning that a gap between the heuristic solution and the globally optimal is not a problem for the real-world application in that case. Also, let's point out that when the solution

is an *approximation of the solution*, we know it is not the optimal one and we know the gap we need to cover to get to the optimal, whereas when calculating an *approximate solution*, the gap is unknown. On the other hand, there are the *exact approaches*, where the gap between the solution found by the exact approach and the optimal solution of the problem is zero. It is important that this statement can be proved by the way that the approach is built. Although, exact approaches tend to provide us with better solutions in terms of reduction of cost, generally speaking, they cannot yield solutions in the amount of time that heuristics can, and usually take much more time to complete the calculations. In the MMRP, graph and MILP are heuristic and exact methods correspondingly. That means that later on in the current thesis we will explore several advantages and deficiencies of the methods and see hands-on examples on how exact and heuristic differ from each other. Finally, we will present *hybrid approaches* which are a combination of the above perspectives. We will try to incorporate advantages of both methods into one.

But, before getting into the actual presentation of the modelling approaches, let's make a reference to *precomputation techniques* used in most heuristic methods. Like all of the aforementioned terms, introducing the reader to precomputation techniques, will aid us to better understand the next chapter of the thesis. With no further ado, precomputations are techniques that are applied in an "offline" stage of the process of calculating results, and enable us to speed up the same process later on when we need to. This is especially useful when we need a navigation system to be the back-end of an online web-platform. In such cases, usually there is a graph builder module that does all the precomputations and data structure building when the system is in "preparation" mode or offline. Then another part of the application handles the online phase of the system and answers to queries of the user's browser. That is the case with OpenTripPlanner (OTP) [6], an open source library used to build such systems. OTP uses some of the methods we will describe in the following paragraphs in order to turn vast graphs into smaller ones that can be managed by the computer. The gains from using precomputations techniques, is not only computer memory (RAM), hard disk space and CPU usage but also speed in the final stages of computations. Nonetheless, we should mention that most of the graph approaches are, in one way or another, connected to Dijkstra's algorithm, an algorithm first introduced in 1956 by computer scientist Edsger W. Dijkstra. Although Dijkstra in his initial proposition, described an algorithm that was not associated with precomputations, through the years many variations came up and, finally, the ones that become most well-known are the ones that use heap space to order the nodes by some criteria and guide the search. We mention that because we want the reader to get to understand that graph algorithms, and generally approaches with graphs, have always been connected with some type of precomputation even on a theoretical level. In the following chapters of this thesis, we provide more insights to the way that Dijkstra's algorithm works. OTP is also used by the *GreenYourMove* project [7] which builds a pan-European journey planner and helps consolidate modern policies related to transportation all over the EU.

3 The problem of rescheduling of public transport services

3.1 Problem description

The problem of *synchronization* of public transport services, also referred to as the timetabling or timetable generation problem, is the *optimization* problem where the departures of itineraries of public transport vehicles are scheduled in *specific way* in order to achieve a *specific outcome*. The solution to the problem, which is a new timetable of operation for the services, improves the welfare of the stakeholders of the system. Although the criteria according to which optimization occurs can vary, and not all stakeholders receive equal benefit at all times, the most prominent instance of the problem is the case when the resulting timetables have been modified in such a way that *waiting time* of passengers using the network is minimized. That waiting time is calculated at some nodes (i.e. stops) of the network which are served by more than a single service in a specified time horizon.

Several instances of the problem can be found in the literature, whose definition can greatly differ from ours. Before proceeding to the following sections, and the explanation of several definitions that will be essential for our discussion, let's remark the key distinction that is being noticed in several works in the literature. This distinction has to do with the concept of analyzing a *single* route of the network or *holistically* considering every route. Consecutively, two are main sub-categories of problems of synchronization of public transit services:

- Holistic synchronization of a network according to other services; and
- Synchronization of a single route according to other services.

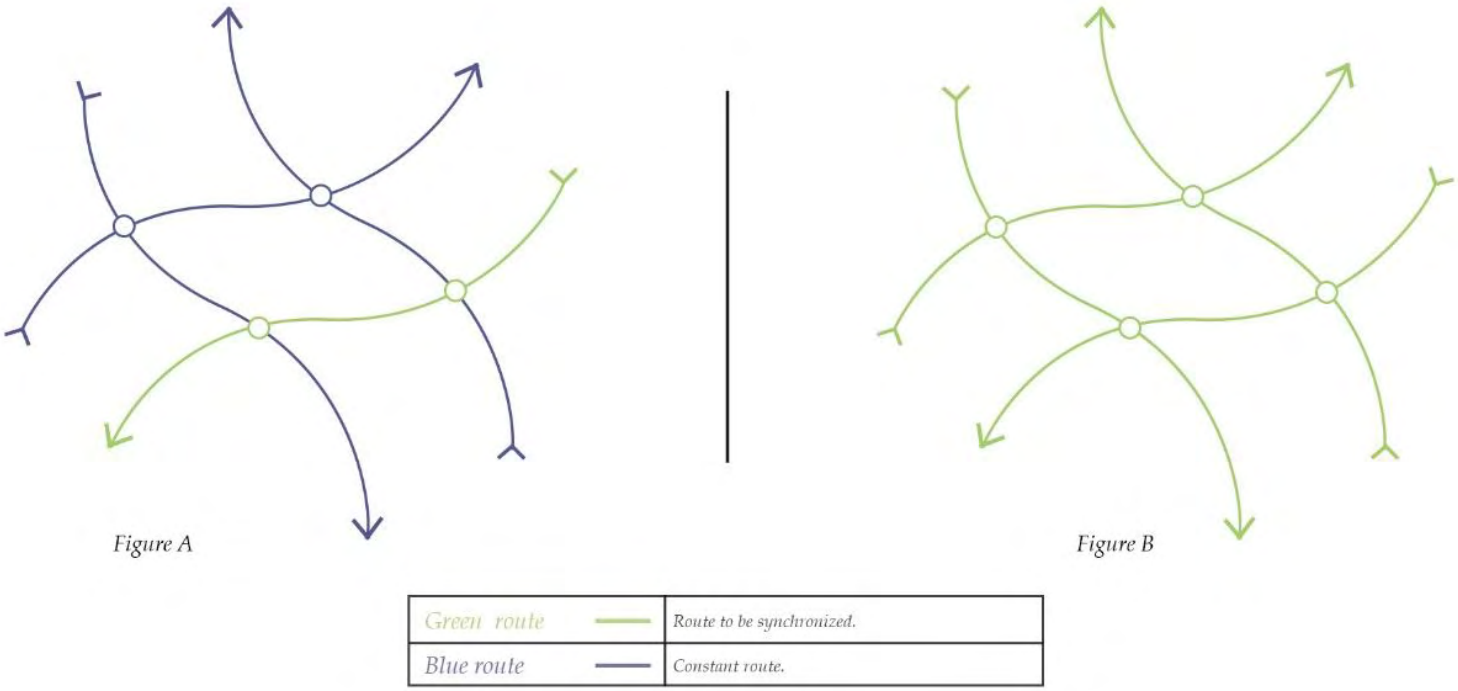

Figure 2: Graphical representation of the single route synchronization on the left side (Figure A) of the figure. Representation of the case of holistic synchronization of the network on the right side (Figure B) of the figure.

3.2 Important terms

3.2.1 Waiting time for passengers and operators

Waiting time is defined as the time that a passenger spends at a stop (i.e. node) of the network, from the moment in time when he or she has got off the vehicle, that the transport operator is utilizing to provide the service, until he or she embarks onto the next vehicle that will lead him to another stop or the final destination. The vehicles that the passenger uses may belong to one unimodal network or may be part of more complicated co-modal network with services from several providers.

However, an important distinction should be made at this point, concerning the difference in the definition of *waiting time* from the perspectives of the passenger and that of the operator of the services. In more detail, the passenger may not reach his final destination after a single interchange between routes but can change several vehicles until he or she completes the trip. Hence, from the perspective of the passenger the *overall waiting time* occurs cumulatively

from his potential waiting at several stops in a single journey. For the operator, though, overall *waiting time* can be measured at individual (i.e. single) stops independently of the potential journey to be followed by any passenger. In our study, we will adopt the definition for *waiting time* as if we were the system operators and try to minimize it at individual stops. Of course, adopting one of the two perspectives does not mean that the other stakeholder will not benefit from the solution of the problem. In order for those definitions to coincide, a modeler should include the demand for the itineraries of the network.

3.2.2 Feasible transitions in the network

Another very important aspect of the problem are the *feasible transitions* of the network. A *feasible transition* is defined as the ability to make a transition from one service to another, given that, they occur the one after another within a specified time horizon (i.e. time window). A transition can happen at the same node of the network or at nodes that are relatively close to each other. In other words, for two different services that are part of a passenger's journey, a *feasible transition* between those services is considered valid when the arrival to the node utilized by one the services happens before the departure of the other service from the same or other node within a specific time frame. This is a very important notion of the problem because, when not modeled properly, an approach may lead to a *trade-off* between the number of *feasible transitions* and *overall waiting time* in the network. According to objective, an optimization process may lead to less *overall waiting time* by reducing the number of *feasible transitions*. It is always preferable to have the same or more *feasible transitions* with less *overall waiting time*.

Also, according to the modeler and the type of the approach (i.e. exact, heuristic, metaheuristic, hybrid approaches) *feasible transitions* may be considered constant throughout the solution of the problem or may fluctuate according to desired objective.

3.2.3 Transitional nodes of the network

A transitional node of the network is a node (i.e. stop) which is served by multiple routes of the network and feasible transitions can occur on that node. Also, transitional nodes are the nodes on which we measure waiting time. In some cases, such as the first instance of the problem that we will later present, transitions of the passenger from one transitional node to another are not considered. In other cases, such as the *second* and *third* instance of the problem that we will later present, those transitions from one transitional node to another are included. If a node is not *transitional*, then feasible transitions and waiting times are not considered on that node.

Those nodes are the basis of the analysis in this paper and in most of the surveys that can be found in the literature. They function as the *Points of Interest* (POIs) for our network. The

choice of the modeler about which of them will be included in the model is crucial for the final outcome.

3.2.4 The General Transit Feed Specification (GTFS)

Every public transport operator has a schedule or timetable, based on which the services are executed and the whole organization operates. With the growing need to make that schedule publicly available to the potential passengers through Intelligent Transportation Systems (ITS), it is crucial to properly represent those timetables. Especially, when the data are subject to be used by third party organizations and software developers, the content needs to be polished and clear. For that reason, the GTFS was introduced by Google and TriMet in 2006, in order to serve as a de facto standard for how public transport companies should share their timetables with other organizations in order to achieve their incorporation in ITSs. The GTFS adopts a very simple layout for the its main data *feed*, which is the main file according to the GTFS. The number of the files to be included by the organization is not specific, but varies, because some of the files are optional to include.

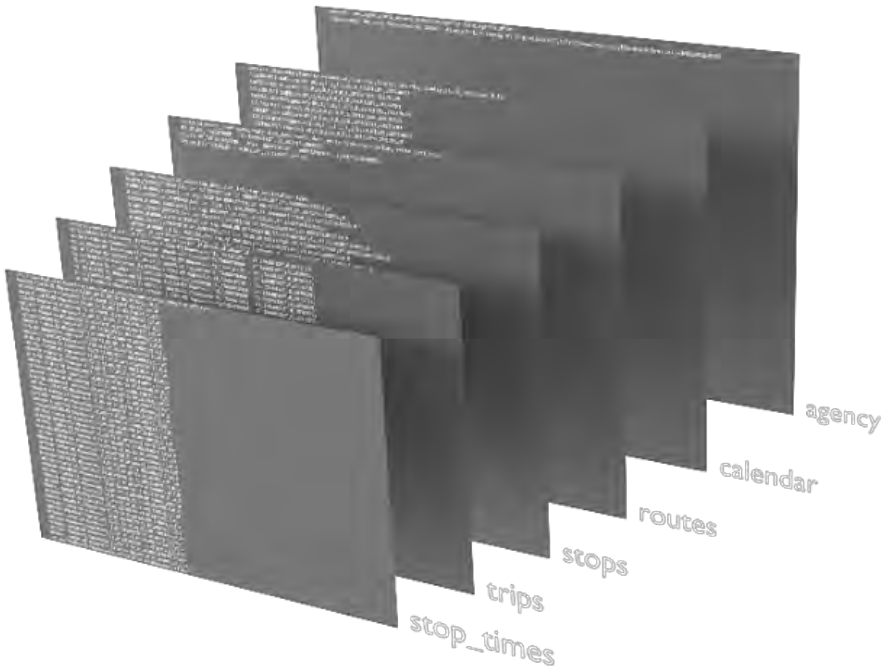

Figure 3: Graphical representation of the files that are necessarily included in a GTFS feed

3.3 Literature review

As mentioned in previous sections, the synchronization of public transport services is a problem that has been addressed in several studies in the last decades. The fact that the solutions to the several instances of the problem can be applied to already existing infrastructure has been an extra motive for researchers. In our research and review of the literature, special emphasis has been put on studies that, not only came up with models and analogous approaches, but also introduced improvements on real world systems. Another important criterion for the studies included below was the mode of transport that the approaches introduced were designed for. Although a great variety of mass transportation systems, such as airline scheduling or port scheduling, have been analyzed in several studies, in this review we will focus cases of studies that focused public transport services that operate on land, such as long-haul rail transportations systems, rapid transit rail systems and bus transport services. A final common attribute between the articles presented is the fact that they all adopt a *macroscopic* approach towards scheduling, that puts emphasis on the practicality of the results.

Without further due, first, we will make two references to two surveys that provided useful insights about the definition of waiting time and most modern literature has been based upon. We refer to the works of Bussieck et al. [8] and Goverde et al. [9]. For the former work, it provides detailed reviewing of the planning processes in public rail transport and how they can be improved based on analytics models. While the authors share the same definition of waiting time with us and solve a periodic event scheduling problem, the capacity of the computer systems at that time restricts the application of the approach to a small network. As for the work of Goverde et al. [9], here, the authors introduce a mathematical model that enables the calculation of hold times of trains at several nodes of the network in order to make sure that transitions between services remains secure (i.e. feasible), even if delays show up in the real time operation of the network. Also, waiting time is defined in a different way, with authors introducing primary and secondary waiting time based on passenger flows, passenger types and stations of the network that waiting time may occur.

An important study has been conducted by Ceder et al. [10] in 2001. The authors model and solve the problem of *creating* timetables with maximal synchronization by modelling it as assignment problem with a final Mixed Integer-Linear Programming (MILP) formulation, whose solution is attempted with a heuristic algorithm. By the term maximal synchronization, here, the researchers refer to the objective of maximizing simultaneous arrivals at the transfer (i.e. transitional) nodes of the network. Meaningful results are yielded in a real-world example from an Israeli bus network.

Another well-known paper is the work of Erranki et al. [11]. One could claim that it serves as an extension to the work of Ceder et al. [10], using the same framework for the solution of the instances of the synchronization problem. However, although the problem is formulated as MILP and solved by a heuristic algorithm for instances of the problem with greater size, similarly to Ceder et al. [10], in this survey, the simultaneous arrival of two services at a transfer point is considered when they occur within a time-window or time gap.

Similar to the aforementioned studies is the work of Ibarra-Rojas et al. [12]. While the authors focus on the specific case of Services in Monterrey, Mexico, they also share informative insights about the problem. Their work can be perceived as an extension of the studies of Ceder et al. [10] and Erranki et al. [10] and they were some of the first researchers to introduce local search for the problem of timetabling (i.e. synchronization), as referred to in their studies.

In a study conducted by Saharidis et al. [13], the authors introduce a MILP model to holistic model the bus network in the island of Crete, Greece. Some of the constraints used in this paper are common with constraints used in Ceder et al [10], Erranki et al. [11] and Ibarra-Rojas [12]. Also, this formulation has served as a basis for the first instance of the problem and its associated modelling that we will present in later section of this paper.

Two other important surveys were published in 2014 by Barrena et al. [14] [15]. For the latter, the authors present alternative formulations to the ones that have been described in paragraphs above and that are very efficient. Variables are included that represent the flow of passengers in the network, and thus the demand. Also, binary variables are used in order to describe whether a station of the network is served by a specific train. By applying their approaches to the metro lines of Madrid Metropolitan Railway, researchers were able to yield, on average, a reduction of 30% in the average waiting time of passengers. In the other paper [14] of the authors, alternative formulations are provided but, most importantly, a fast-adaptive large neighborhood search (ANLS) metaheuristic is presented. It allowed to solve bigger instances of the problem compared other solution approaches. The significance of neighborhood search is also referenced in the work of Hassanayebi et al. [16], who introduce a MILP and Non-linear programming (NLP) model, that they then solved with an adaptive variable neighborhood search algorithm for larger instances of the problem. Applications of their modelling and solutions approach were on data from Tehran intercity underground rail lines in Iran.

Furthermore, there are other two papers that do not solve the same problem as in our case but do consider waiting times and meaningful insights can be extracted. The first is the research conducted by Hall et all. [17]. In this study, special focus is given to control policies to minimize overall transfer time of passengers. By adjusting departure times in a bus network and considering waiting times at single nodes, researchers exemplify the effectiveness of the approach by applying it to data collected from Los Angeles County's Metropolitan Transportation Agency. The second mention is about the work of Reinhardt et al. [18], which addresses the problem of assisting passengers in need of special assistance in airports. The scheduling of such processes takes place in dynamic environments and the time that those passengers are left unsupervised (i.e. waiting time) should be minimized. Although the problem deals with a dial-a-ride problem with a heterogenous fleet, it could be claimed that its network is similar to a multi-modal network. The authors modelled this problem as an assignment problem and gave a solution to it with a simulated annealing-based heuristic. The model developed shares common constraints with our models.

Finally, references will be made to two other papers that are related to the two first instances of the problem that we will present. First, we will discuss the work of Sun et al. [19] , who, with their study of the case study of the Singapore Metro system, tested three models. The first model achieves optimal design of the railway line by ignoring capacity constraints and the two other models, that include limited capacity of the trains, achieve optimal operation of the metro line. In this study, the focus is being put on the improvement of the timetable of single track (i.e. route) of the network. Another survey that shares common elements with some of our approaches is the work of Wong et al. [20], in which, just like in the first two instances of the problem that we will present, counts only the first feasible transition after the arrival of service at a node of the network. Their work is especially focused in rail system and they are able to include other parameters of the system such as the regularity of dwell times at the several stops of the network, regularity of headways between trains and collision avoidance during startup period.

4 Approaches for the rescheduling problem

Three instances of the problem of rescheduling were modelled and solved in this thesis. The first two are two theoretical and simplistic approaches, and the third is an approach that is compromised out of three stages and can be applied to real world systems. With no further ado, let's move on to discuss the contributions.

4.1 Assumptions for all instances of the problem and the models developed

Before the particulars for each case of models are presented, the general *assumptions* that apply to all cases are mentioned. First, travelling time of vehicles between nodes (i.e. stops) of the networks are considered constant and not subject to dynamic factors influencing the traveling time (e.g. traffic, weather conditions). Travelling times are included as they emerged from the processing of the data provided by the public transit agencies. Also, for the cases when we needed to consider the walking speed of the passengers, when they needed to move in-between stops or stations of the network, constant walking speed was assumed for all passengers. Finally, in order to reduce the complexity of the models, the bus-drivers' work schedule was not included in the analysis.

4.2 Fundamental instances and modellings of the problem of rescheduling

In this section we will present two very basic applications of MILP modelling techniques on the problem of rescheduling. The instances presented are the *theoretical one-node* problem and the *theoretical two-sequential-nodes* problem. As it is easily understood, the first instance includes the analysis of one node and the itineraries that serve it. The second case, is a bit more complicated, and requires the analysis of two sequential stops. This section is essential for the better understanding of the literature review by the reader. The aim with the presentation of those cases is to provide the reader with the simplest of examples for the problem of rescheduling. The logic according to which the analysis happens will be presented here but the actual results will be included in *Section 5*.

In respect, we initially present and discuss the case where we inspect *one stop* and the corresponding routes and trips that serve that stop.

4.2.1 The theoretical one-node problem

The theoretical one node problem is considered to be the simplest instance of the problem on which we can apply a MILP model and acquire a solution. We consider a single

node with several arrival and departure events. The stops, routes and their corresponding service times are *manually* analyzed in order to find out which combinations of services create waiting time. As it is apparent, after the manual analysis, the data were hard-coded into a C++ program. Below a graphical representation of the case is depicted:

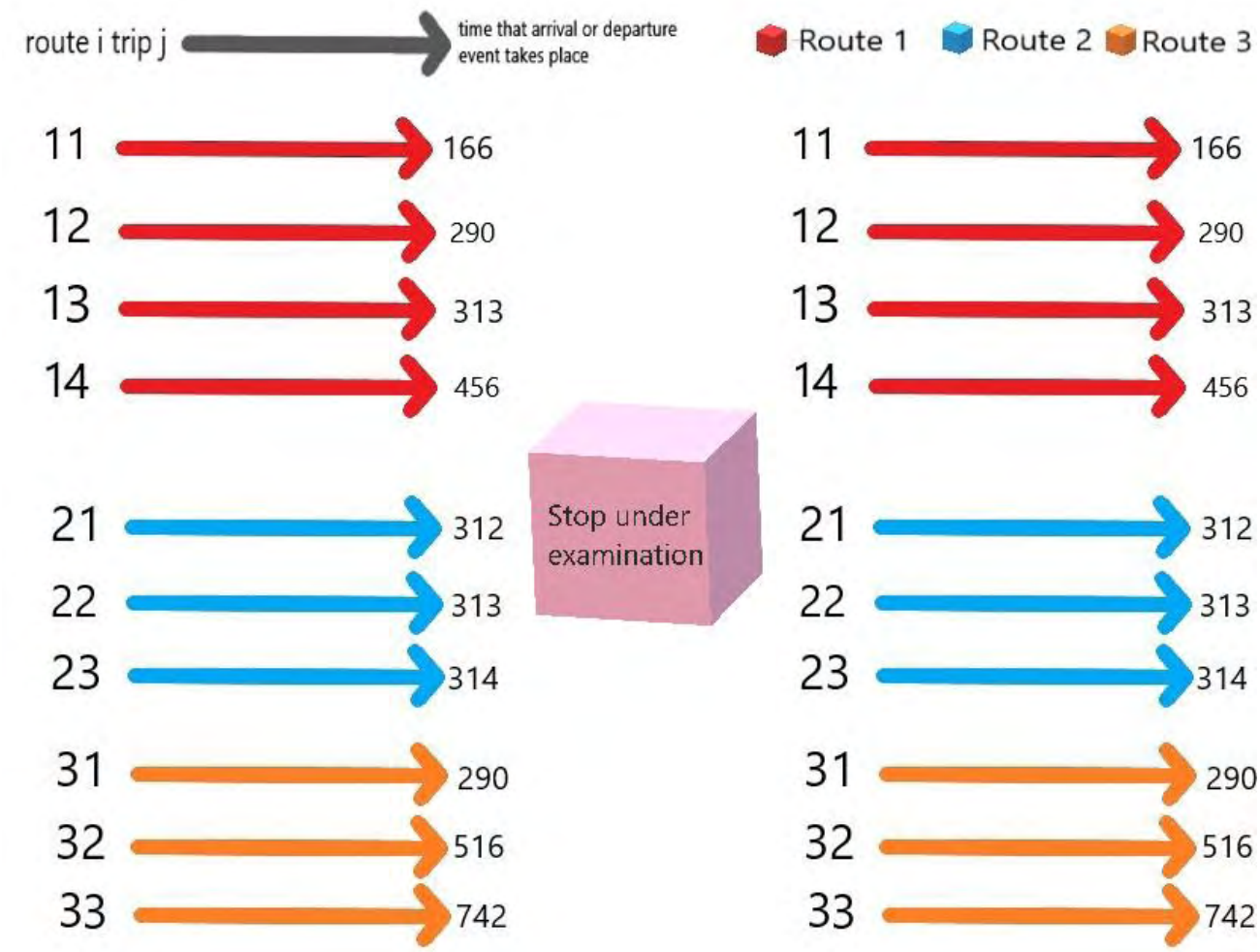

Figure 4: The theoretical case of rescheduling the itineraries that serve one stop

In this example we have 3 routes, with the first route having 4 trips, and the second and third routes having 3 trips each. With index *i* we refer to the route and with index *j* we refer to the trip. Specific services are referred to as *ij*. By manually analyzing the graph we can identify the following waiting time events between nodes. Waiting are created by the following services:

1. **Transition from service 11 to 21**
2. Transition from service 11 to 31
3. **Transition from service 12 to 21**

4. Transition from service 12 to 32
5. **Transition from service 13 to 3**
6. Transition from service 13 to 32
7. **Transition from service 14 to 33**
8. **Transition from service 21 to 13**
9. Transition from service 21 to 32
10. **Transition from service 22 to 14**
11. Transition from service 22 to 32
12. **Transition from service 23 to 14**
13. Transition from service 23 to 32
14. **Transition from service 31 to 13**
15. Transition from service 31 to 21

The mathematical model that can be applied to this case is:

Nomenclature:

Indices	
r	Indices to indicate *route.*
t	Indices to indicate *trip.*

Parameters	
$\overline{NSDT_{rt}}$	Non-synchronized departure times for route *r* and trip *t* (current).
$\overline{NSAT_{rt}}$	Non-synchronized arrival times for route *r* and trip *t* (current).
$\overline{AP_r}$	*Allowed Percentage* of change in departure or arrival time for route *r* (determined in pre-optimization).
$\overline{RF_r}$	*Route Frequency* of the execution of the trips for a route *r* (calculated in pre-optimization).

Decision Variables	
TOS_{rt}	Time when route r with trip t serves the stop.

The formulation:

Constraints

$$\overline{NSDT_{rt}} - (\overline{AP_r} * \overline{RF_r}) \leq TOS_{rt} \leq \overline{NSDT_{rt}} + (\overline{AP_r} * \overline{RF_r}) \ \forall \ rt \qquad \text{Constraint (1)}$$

$$TOSrt \ \in \ Z^+$$

$$(*) \ for \ every \ TOSrt \ that \ is \ declared \ according \ to \ the \ pre-optimization \ phase$$

Objective function

$$MINIMIZE \sum (TOS_{r1\,t1} - TOS_{r2\,t2})$$

$$for \ selected \ cobinations \ of \ r1, t1, r2, t2 \ according \ to \ pre-optimization$$

4.2.2 The theoretical two-sequential-nodes problem

Moving on, we investigate the case of *two stops* and the corresponding binding of services that serve both stops. Just like in *Section 4.1.1*, we will manually analyze waiting times and present a mathematical model that will calculate the desired timetables for that *simplistic* case of modelling. Below we present the visual representation of the case.

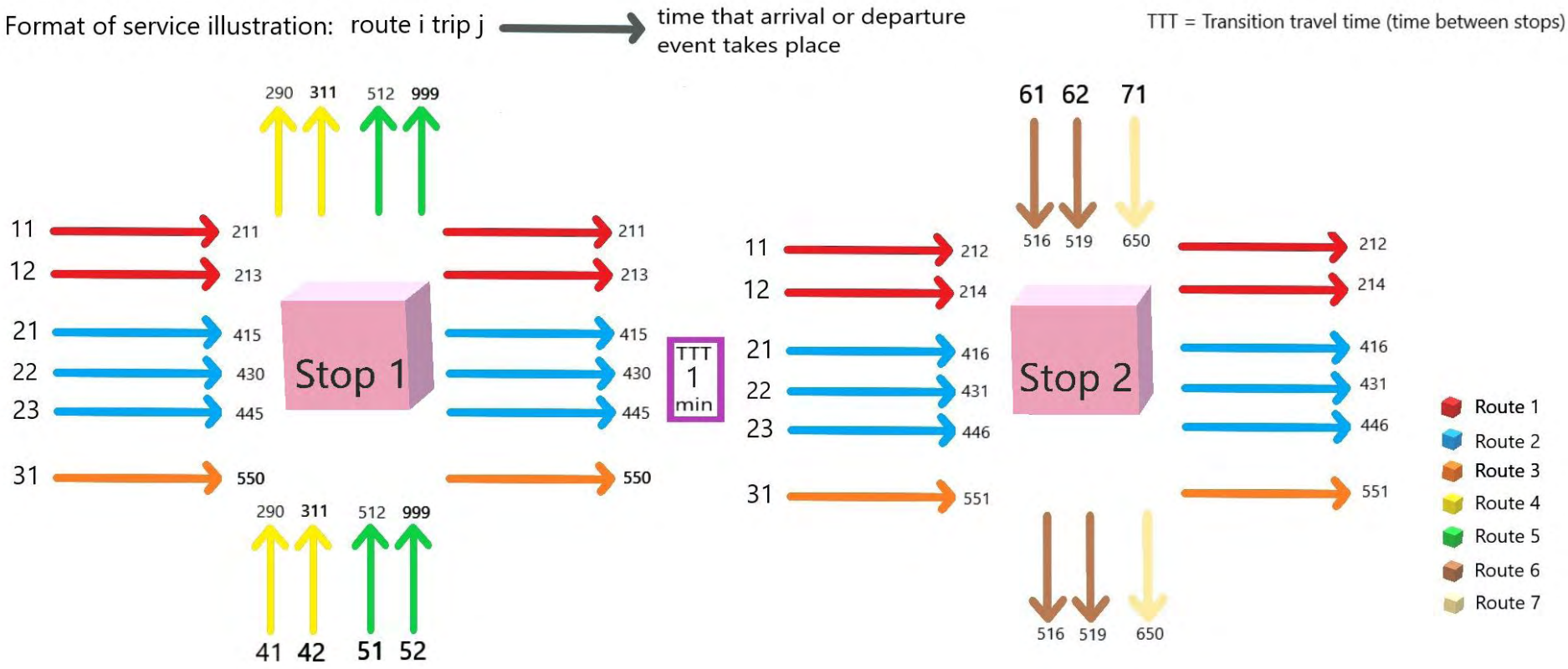

Figure 5: Graphical illustration the two sequential-nodes case.

In this example, we have 7 routes overall with each of the routes having several trips. Just like in the first case, with index *i* we refer to the route and with index *j* we refer to the trip. Specific services are referred to as *ij*. Important to note is that in this example of a network, routes 1, 2 and 3 serve both nodes (i.e. stops), whereas routes 4,5,6 & 7 serve individual nodes. By manually analyzing the graph we can identify the following waiting time events between nodes. Waiting are created by the following services:

For Stop 1:

1. **Transition from service 11 to 21.**
2. Transition from service 11 to 31.
3. Transition from service 11 to 41.
4. Transition from service 11 to 51.
5. **Transition from service 12 to 21.**
6. Transition from service 12 to 31.
7. Transition from service 12 to 41.
8. Transition from service 12 to 51.
9. **Transition from service 21 to 31.**
10. Transition from service 21 to 51.
11. **Transition from service 22 to 31.**
12. Transition from service 22 to 51.
13. **Transition from service 23 to 31.**
14. Transition from service 23 to 51.
15. **Transition from service 31 to 52.**
16. **Transition from service 41 to 21.**
17. Transition from service 41 to 31.
18. Transition from service 41 to 51.
19. **Transition from service 42 to 21.**
20. Transition from service 42 to 31.
21. Transition from service 42 to 51.
22. **Transition from service 51 to 31.**

And no waiting times for transitions from 52. Below we present the manual analysis for the second stop.

For Stop 2:

1. **Transition from service 11 to 21**.
2. Transition from service 11 to 31.
3. Transition from service 11 to 61.
4. Transition from service 11 to 71.
5. **Transition from service 12 to 21**.
6. Transition from service 12 to 31.
7. Transition from service 12 to 61.

8. Transition from service 12 to 71.
9. **Transition from service 21 to 31**.
10. Transition from service 21 to 61.
11. Transition from service 21 to 71.
12. **Transition from service 22 to 31**.
13. Transition from service 22 to 61.
14. Transition from service 22 to 71.
15. **Transition from service 23 to 31**.
16. Transition from service 23 to 61.
17. Transition from service 23 to 71.
18. **Transition from service 31 to 71**.
19. **Transition from service 61 to 71**.
20. **Transition from service 62 to 71**.

And no waiting times for transitions from 71.

The mathematical model that can be applied to this case for the analysis of the two sequential nodes is:

Nomenclature:

Indices	
r	Index to indicate route
t	Index to indicate trip

Parameters	
$NSDTA_{rt}$	Non-synchronized departure times for route *r* and trip *t* (current) at stop A
$NSDTB_{rt}$	Non-synchronized departure times for route *r* and trip *t* (current) at stop B
$NSATA_{rt}$	Non-synchronized arrival times for route *r* and trip *t* (current) at stop A
$NSATB_{rt}$	Non-synchronized arrival times for route *r* and trip *t* (current) at stop B

APr	*Allowed Percentage* of change in departure or arrival time for route *r* (determined in pre-optimization)
RFr	*Route Frequency* of the execution of the trips for a route *r* (calculated in pre-optimization)
TT_{rt}	*Travel Time* between stop A and B for route *r* and trip *t*

Decision Variables

$TOSA_{rt}$	Time when Network Event i should occur at stop A
$TOSB_{rt}$	Time when Network Event i should occur at stop B

The formulation:

Constraints

$$NSDTA_{rt} - (AP_r * RF_r) \leq TOSA_{rt} \leq NSDTA_{rt} + (AP_r * RF_r), \forall\, rt \qquad \text{Constraint (2)}$$

$$NSDTB_{rt} - (AP_r * RF_r) \leq TOSB_{rt} \leq NSDTB_{rt} + (AP_r * RF_r), \forall\, rt \qquad \text{Constraint (3)}$$

$$TOSA_{rt} + TT_{rt} = TOSB_{rt}\,, \forall\, r, t \qquad \text{Constraint (4)}$$

$$TOSA_{rt}, TOSB_{rt} \in Z^{+}$$

$$(*)\ Declared\ for\ combinations\ of\ TOSA_{rt}\ and\ TOSB_{rt}\ that\ have\ been\ decided\ to\ be\ used\ according\ to\ pre-optimization\ phase$$

Objective function

$$MINIMIZE \sum (TOSA_{r1\,t1} - TOSA_{r2\,t2}) + \sum (TOSB_{r3\,t3} - TOSB_{r4\,t4})$$

$$for\ selected\ cobinations\ of\ r1, t1, r2, t2, r3, t3, r4, t4$$

For both cases, the results will be presented and discussed in Results section which *Section 6*.

4.3 Indicative formulations for the problem of rescheduling

For this section of the thesis, we identified and homogenized several indicative works in the literature that are simple and have been used to derive solutions to the problem. Reference will be made to *Formulation 1* and *Formulation 2* which are MILP formulations used for two different instances, *Instance 1* and *Instance 2* of the problem.

4.3.1 Instance 1 of the synchronization problem: The 1-node problem

4.3.1.1 Description of Instance 1 of the problem

As a first instance of the problem, we present the case where each of the routes of a public transit services operators are synchronized separately and for every route we calculate the waiting times, and minimize them, according to *one* node of the route. The rest of the nodes of the route are not considered and waiting time at those nodes is neglected. Specifically, for every route that we would like to improve and achieve less waiting time for its passengers, there are three points of interest, the starting, (i.e. departure) node, the ending, (i.e. arrival) node and a third one, the intermediate node. That intermediate node, is considered as the transitional node, meaning that the potential passenger could hop-on or hop-off the service at that specific node only. That intermediate node could be a station, a city or a town that involves two physical nodes of the network, whose distance in time units is described by the $\overline{T2K}_k$ parameter. The objective of the problem, is to minimize the time spent by the passengers arriving or leaving the transitional intermediate node. However, for each route there are several nodes that could be assumed as the intermediate nodes of the analysis. We select as the intermediate node one of the most popular highly visited nodes of the route under study, based on the assumption that nodes of the network with higher usage will have a greater effect on the passengers and the system after our improved results are applied. For example, if a small-town of 2000 inhabitants is considered as the intermediate node, then, after the synchronization of the routes and trips, a smaller group of passengers would benefit, rather than in the case where a city of 200,000 inhabitants is considered as the intermediate node.

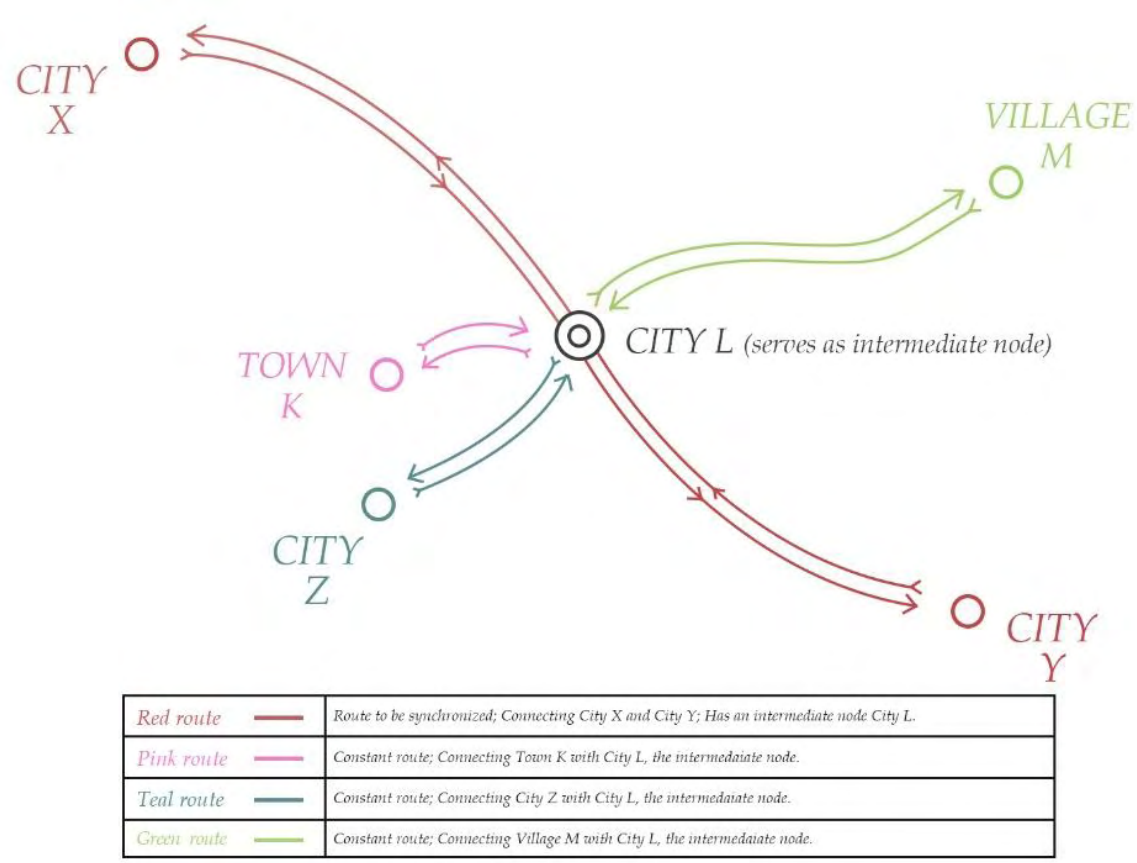

Figure 7: Graphical display of how the first instance of synchronization problems has been addressed

4.3.1.2 Formulation 1 first instance of the synchronization problem

4.3.1.2.1 Indices, Sets, Parameters and Decision variables

Indices	
i, j	Indices used to indicate the nodes of the network;
k	Index used to indicate the intermediate nodes;
n, m	Indices used to indicate routes of the network;
t	Index used to indicate if a connection exists, equals to 0 between nodes 0 and 1, equals to 1 between nodes 1 and 2, etc.

Sets	
I	Set of total number of nodes;

K	Set of total number of intermediate nodes;
$Route_{i,k}$	Set of total number of routes directly connecting nodes i and k;
$Route_{k,j}$	Set of total number of routes directly connecting k and j;
$Route_{i,j}$	Set of total number of routes directly connecting i and j;
C	Set of total number of lines connecting sequential nodes of route.

Parameters

$\overline{T2K}_k$	Necessary traveling time between 2 nodes of the network at the intermediate point k;
$\overline{K}_{i,j,m}$	Travel time from i to j on route m;
$\overline{ST}_{i,j,m}$	The lower bound for a departure time;
$\overline{ET}_{i,j,m}$	The upper bound for a departure time;
$\overline{Route}_{i,j}$	Total number of routes from i to j for a period of one week;
$\bar{X}_{i,j}$	Equals to 1 if a connection exists between nodes i and j, equals to 0 if not;
$\bar{L}_{i,j}$	Equals to 1 if i and j are both nodes of a route to be synchronized *or* if i and j both belong to a route that is not being synchronized. Otherwise, equals to 0 if i or j are nodes/stations that belong to either the route to be synchronized or the routes whose service times will remain the same. This parameter is critical in order to define if the transition from i to j should be taken into consideration and be optimized;
$\bar{G}_{i,j}$	Equals to 1 if i and j are sequential nodes of a route, or 0 in other cases. This parameter is used to define the time difference between the several trips of a single route that serve the same node;
$\overline{GAP}$	The time difference between sequential departures of the same node and line (for the trips that are subject to change only);
$\bar{O}_{i,j,t}$	Equals to 1 if nodes i and j with connection indicator t, are sequential nodes of a route; equals to 0 if not. The parameter is used since the departure time of each intermediate node of a route is directly related to all previous departures. For the second analysis this parameter is not used as the network used does not include more than 2 sequential nodes but only starting and ending point nodes;

$\overline{BigM}$	A very big number;
$\bar{m}$	A very small number.

Decision variables

$DT_{i,j,m}$	Continuous variable, expresses the departure time from i to j on route m of intercity bus or train schedule;
$Y_{i,k,j,m,n}$	Binary variable, equals to 1 if the optimized departure times allow transition from k to j on route n, under the condition that route m was decided to be used for the transition from i to k. Equals to 0, if not;
$WT_{i,k,j,m,n}$	Continuous variable; expresses the waiting time on transition node k, visited by a passenger that arrives from node i of route m and travels to node j as final destination, by taking route n.

4.3.1.2.2 Constraints and Objective function of the mathematical model for the first instance

Constraints	
$\overline{ST}_{i,j,m} \le DT_{i,j,m} \le \overline{ET}_{i,j,m} \quad \forall\, i,j \in I \text{ and } \forall\, m \in \overline{Route}_{i,j}$	Constraint (5)
$\bar{m} \cdot \left(DT_{k,j,n} - DT_{i,k,m} - \bar{K}_{i,k,m} - \overline{T2K}_k\right) - \overline{BigM} \cdot \left(2 - \bar{X}_{i,k} - \bar{X}_{k,j} + \bar{X}_{i,j}\right) \le Y_{i,k,j,m,n}$ $\forall\, i,k,j \in I, \forall\, m \in \overline{Route}_{i,k} \text{ and } \forall\, n \in \overline{Route}_{k,j}$	Constraint (6)
$DT_{k,j,0} - DT_{i,k,m} - \bar{K}_{i,k,m} - \overline{T2K}_k - WT_{i,k,j,m,0} - \overline{BigM} \cdot \left(2 - \bar{X}_{i,k} - \bar{X}_{k,j} + \bar{X}_{i,j}\right) \le 0$ $\forall\, i,k,j \in I \text{ and } \forall\, m \in \overline{Route}_{i,k}$	Constraint (7)
$DT_{k,j,n} - DT_{i,k,m} - \bar{K}_{i,k,m} - \overline{T2K}_k - WT_{i,k,j,m,n} - \overline{BigM} \cdot Y_{i,k,j,m,n-1} - \overline{BigM} \cdot \left(2 - \bar{X}_{i,k} - \bar{X}_{k,j} + \bar{X}_{i,j}\right) \le 0$ $\forall\, i,k,j \in I \text{ and } \forall\, m \in \overline{Route}_{i,k}, n > 0 \text{ and } n \in \overline{Route}_{k,j}$	Constraint (8)
$\left(DT_{k,j,m} - DT_{i,k,m} - \bar{K}_{i,k,m}\right) \cdot \bar{O}_{i,k,t-1} \cdot \bar{O}_{k,j,t} = 0$ $\forall\, i,k,j \in I \text{ and } \forall\, m \in \overline{Route}_{i,k}, t > 0, t \in C$	Constraint (9)

$\overline{GAP} \leq (DT_{i,j,m} - DT_{i,j,m-1}) \cdot \bar{G}_{i,j} \; \forall \; i,k,j \; \in I, m > 0, m \in \; \overline{Route}_{i,j}$	Constraint (10)

Objective function

$$\min \sum_{i=0}^{I} \sum_{k=0}^{K} \sum_{j=0}^{J} \sum_{m=0}^{Route_{i,k}} \sum_{n=0}^{Route_{k,j}} WT_{i,k,j,m,n} \cdot X_{i,k} \cdot X_{k,j} \cdot \left(1 - X_{i,j}\right) \cdot \left(1 - L_{i,j}\right), \forall i \neq k, k \neq j, i \neq j$$

4.3.1.3 Explanation of the formulation and details

In this formulation, the Objective function minimizes the overall waiting time for transitions that are considered as feasible transitions for the problem in the intermediate node. Constraint (1) makes sure that the departure times will remain between desired bounds and Constraint (2) assigns the right value, 0 or 1, to variable $Y_{i,k,j,m,n}$ according to whether the transition between nodes *i*, *k* and *j* with route *m* and *n*, correspondingly, is feasible or not. Moving on to constraints (3), (4) and (5), they connect variable $WT_{i,k,j,m,n}$ to the rest of the model, while, in combination with constraint (5) they guarantee the continuity of time in our model. Constraint (6) spaces out the trips of a route in order for the schedule to keep its initial structure as far as when each of trips start in correlation to other trip services of the same route.

4.3.2 Instance 2 of the synchronization problem: The 4-nodes problem

4.3.2.1 Description of Instance 2 of the problem

As a second instance of the problem, we introduce the case where a 4-nodes schema is adopted for the minimization of waiting times. This problem was based on the initial concept that the synchronization of a route is efficient enough when it is based on the first (i.e. starting, departing), and the last (i.e. ending, destination), node of the route. Those two nodes had then to be paired with two other nodes, which are used for the calculation of waiting times. This schema is especially efficient when the first node of the route is paired with another node of the same first city, and the last node is also paired with another node in the same second city. As a result of this process and for this second problem, waiting times are measured between the first pair of nodes and then for the second pair of nodes. The first of those two nodes of the first pair is called ID_{AB} and the second ID_{DT}. The first node of the second pair of nodes is called ID_{AT} and the second ID_{DB}. Routes and trips that serve nodes ID_{AB} and ID_{DB} we're *not* synchronized and are considered *constant*. We also refer to the routes that are not synchronized and the associated stops of the network, as the *constant network*.

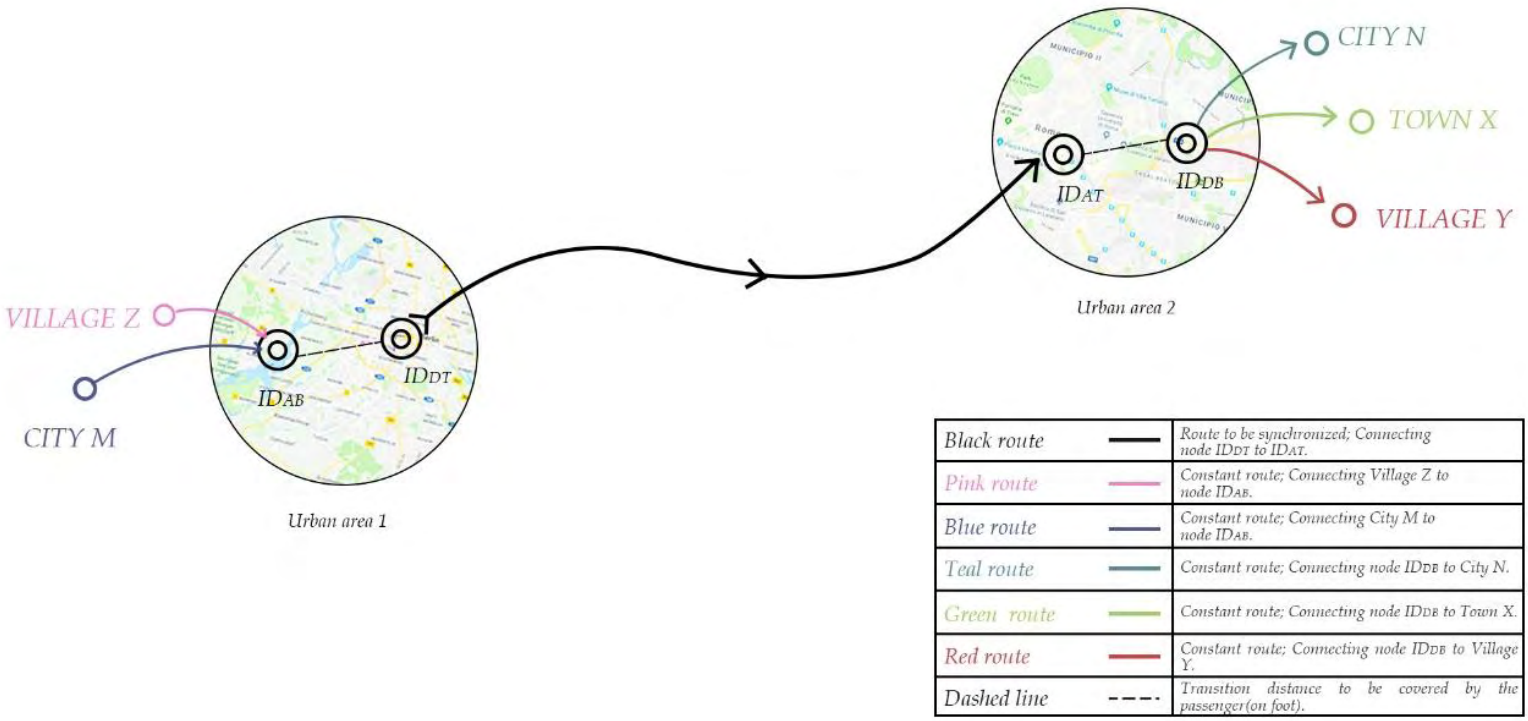

Figure 8: Graphical display of how the second instance of synchronization problems has been addressed

For the calculation of overall waiting time, we proceeded with two parts. The first part of the overall waiting time, was calculated between all trips of all *constant* routes that served node ID_{AB} and the trips of the synchronizing route (one at a time) that served ID_{DT} and departed from it in the first city (or any part of the network considered). Then a second part of the overall waiting time was calculated, between the constant trips that served ID_{DB} and departed from it, and the trips of the synchronizing route that arrived at ID_{AT}. A visual representation of the analysis can be found below.

In comparison to the first instance of the problem, in this case, we focus our waiting time minimization efforts on the first and the last node served by each route, or in other words, in the starting location of the route and the destination location of the route. No emphasis is being put on intermediate nodes of the route and how they interact with other local services. An important aspect is that, by solving this second problem, we do not have to include in our analysis the importance of the intermediate node, and, in simpler terms, the parameters that will define the results are less dependent on the choice of the officer using the model. A greater percentage of the passengers will potentially benefit, as most of them trip use the service in order to move from the starting node to the final node of the route. On the other hand, apart from their differences, the first and second instances of the synchronization problem, both share the same assumptions and analyze the waiting times of networks by examining *every* single route.

4.3.2.2 Formulation 2 for the second case of synchronization problems

4.3.2.2.1 Indices, Sets, Parameters and Decision variables

Indices

i Index used to indicate the route and trip; A unique Identity number (ID) was used to distinguish each trip of each route; Used for trips between nodes ID_{AB} and ID_{DT};

j Index used to indicate the route and trip; Used for trips that connect ID_{DT} and ID_{AT};

k Index used to indicate the route and trip; Used for trips that connect ID_{AT} and ID_{DB};

Sets

I_{max} Set of total number of trips of routes arriving at node ID_{AB};

J_{max} Set of total number of trips of routes connecting node ID_{DT} with node ID_{AT};

K_{max} Set of total number of trips of routes departing from node ID_{DB}.

Parameters

BAT_i Arrival time of trip with ID i at node ID_{AB};

TDT_j Departure time of route with ID j from node ID_{DT};

TAT_j Arrival time of route with ID j at node ID_{AT};

TTT_j Travel time from node ID_{DT} to node ID_{AT} with trip with ID j;

BDT_k Departure time of a trip with ID k from node ID_{DB};

IT Average time needed in order for a passenger to move between a pair of selected nodes;

R	A parameter used to define the upper and lower bounds of possible change of existing departure time of route j connecting ID_{DT} with ID_{AT}(route to be synchronized). For instance, if $R = 5$ then the new departure time has -5 minutes or +5 minutes difference compared from the current one;
M	A very big number;
m	A very small number.

Decision variables

$DTTV_j$	Continuous variable, expressing the departure time for trip with ID j, connecting ID_{DT} and ID_{AT};
$WT1_{i,j}$	Continuous variable, expressing the waiting time between trips with ID i and j, that serve ID_{AB} and ID_{DT} in correspondence;
$Y1_{i,j}$	Binary variable, equal to 1 if the departure times of trips with ID i and j allow transition with j from ID_{DT} to ID_{AT}. Equals to 0, otherwise;
$WT2_{j,k}$	Continuous variable, expressing the waiting time between trips with ID j and k, that serve ID_{AT} and ID_{DB} in correspondence;
$Y2_{j,k}$	Binary variable, equal to 1 if the departure times of trips with ID j and k allow transition with k from ID_{DB} to the final node of route k. Equals to 0, otherwise;

4.3.2.2.2 Constraints and Objective function of the mathematical model for the second case

Constraints	
$TDT_j - R \leq DTTV_j \quad \forall j \in J_{max}$	Constraint (11)
$DTTV_j \leq TDT_j + R \; \forall j \in J_{max}$	Constraint (12)
$m \cdot (DTTV_j - BAT_i - IT) - Y1_{i,j} \leq 0$ $\forall i, j \in I_{max}, J_{max}$	Constraint (13)
$0 \leq 1 + m \cdot (DTTV_j - BAT_i - IT) - Y1_{i,j}$	Constraint (14)

$\forall i, j \in I_{max}, J_{max}$	
$m \cdot \left(BDT_k - DTTV_j - TTT_j - IT\right) - Y2_{j,k} \leq 0$ $\forall j, k \in J_{max}, K_{max}$	Constraint (15)
$0 \leq 1 + m \cdot \left(BDT_k - DTTV_j - TTT_j - IT\right) - Y2_{j,k}$ $\forall j, k \in J_{max}, K_{max}$	Constraint (16)
$DTTV_{[0]} - BAT_i - IT - WT1_{i,[0]} \leq 0$ $\forall i \in I_{max}$	Constraint (17)
$DTTV_i - BAT_i - IT - M \cdot Y1_{i-1,j} - WT1_{i,j} \leq 0$ $\forall i \in I_{max}, j \in J_{max}, j > 0$	Constraint (18)
$BDT_k - DTTV_i - TTT_j - IT - WT2_{j,k} - M \cdot \left(1 + Y2_{j+1,k} - Y2_{j,k}\right) \leq 0$ $\forall j \in J_{max}, j < (j_{max} - 1)\ \forall k \in K_{max}$	Constraint (19)
$BDT_k - DTTV_{j_{max}-1} - TTT_{j_{max}-1} - IT - WT2_{j_{max}-1,k} - M \cdot \left(1 - Y2_{j_{max}-1,k}\right) \leq 0$ $\forall k \in K_{max}$	Constraint (20)
$BDT_k - DTTV_{j_{max}-1} - TTT_{j_{max}-1} - IT - WT2_{j_{max}-1,k} - M \cdot \left(1 - Y2_{j_{max}-1,k}\right) \leq 0$ $\forall k \in K_{max}$	Constraint (21)

Objective function

$$\min \sum_{i=0}^{I_{max}} \sum_{j=0}^{J_{max}} WT1_{i,j} + \sum_{j=0}^{J_{max}} \sum_{k=0}^{K_{max}} WT2_{j,k} \qquad \forall i \in I_{max}, j \in J_{max}, k \in K_{max}$$

4.3.2.3 Explanation of the formulation and details

The objective function of the problem aims to minimize the sum of the two decision variables $WT1_{i,j}$ and $WT2_{j,k}$, which correspond to the waiting time at the first node served by the route under examination and the last node of the route. Constraints (7) and (8) guarantee that departure and arrival times throughout the network will remain between bounds. Constraints (9), (10), (11) and (12) are utilized in order for variables $Y1_{i,j}$ and $Y2_{j,k}$ to take the appropriate values, 0 or 1, according to if transitions between the routes that they refer to are feasible or not. Next, Constraints (13) to (17) are used in order to $WT1_{i,j}$ and $WT2_{j,k}$ to take the appropriate values and ensure time continuity in our model.

4.4 A generic approach to the problem of rescheduling

In the scope of this thesis, a solution approach is introduced for the problem of the synchronization of public transport services. On the one hand, a framework is proposed on how the data can be handled and then get loaded on to a MILP model that after its execution produces a new schedule for the transit agency whose data are under research.

On the one hand then, a three-stage framework proposed. Just like numerous times in the literature [21], a successful attempt was made to separate the optimization process into several steps and separate the computational results from the final decisions made by the transport agencies. The basic layout of the approach is as follows:

- First step: Pre-optimization phase - Analysis and reduction of the size of the data
- Second step: Main optimization phase – Execution of MILP model and solution by full enumeration for each day of the week
- Third step: Post-optimization phase – Analysis of results and synthesis into a meaningful schedule that can be utilized by the agency

Whereas in other approaches to the problem of rescheduling, the modelling of the problem was directly interconnected with the dataset available, in our approach we make sure that the data, in our case the GTFS data, are analyzed and downsized to their essential parts that need to be included in order for the problem to be solved efficiently. We refer to this initial stage of the approach as the Pre-optimization stage. Next, the data are given as an input to the mathematical model that will associate them with the right mathematical structures. After applying a solver software, we acquire the solutions to the mathematical model that minimize overall waiting time. Finally, the post-optimization analysis takes place where computational results are brought together into a schedule that can be applied. Although our case study occurred prior to contacting the services operators, in the post-optimization phase the opinion of the managers of the services can be considered. In the next three subsections 4.1, 4.2 and 4.3, we will present the details of each stage of the approach and then in 4.4, we will discuss the mathematical formulations of the problem.

4.4.1 Pre-optimization stage

The pre-optimization stage is the most definitive stage for our approach. The proper analysis and storage of data is often neglected and leads to models of great sizes that take more time than necessary to solve. An important role in the ability of the modeler to make use of the aforementioned concepts is the data integration models that are assumed and implemented. An important factor for our analysis is that is it based on the GTFS and its way of associating the different entities of the problem.

Moving on to the details of the first stage, its function is based upon three parameters, which are presented in the list below:

- Time parameter – Used to define the maximum waiting time of a passenger that arrives at a station and potentially makes the transition from one route of the network to another. This time parameter creates the time window in which feasible transitions are considered valid and transitions that could occur outside that time window are not incorporated into the main optimization phase.
    - Example given(e.g.): If time parameter is set equal to 20, then that means that during the pre-optimization phase and after analyzing the data, transitions between routes that serve the same stop but with a time difference of less than 20 minutes are stored in the appropriate data structures and given as input to the model.
- Distance parameter – Used to filter events in the GTFS by the distance at which they occur. A feasible transition from one stop to another happens only if the second stop to which we want to go to on foot, falls inside a circle defined with its center being the first stop and its radius being equal to the value of the distance parameter in meters.
    - Example given(e.g.): If the distance parameter is set at 20 meters then we consider that the average passenger can walk at average 20 meters from a stop to another and he would not walk 21.

  Based on the network that is under consideration this parameter should change. When analyzing a metro network, the passenger usually makes the transition between routes that serve one metro station, but when he or she makes a trip in an urban environment from the house to the workplace, one could possibly walk 300 meters in order to change routes, an action that could have significant financial benefits for that person over time.

- GTFS dataset – Any set of data that describe a public transport network and the services of an organization.

When those parameters are set to specific values the preprocessing takes place and through several routines produces the downsize dataset. Reduction in size can be seen in later Section where the case is presented. The downsized data are filtered and the parameters for the mathematical model and the main optimization phase are calculate by an interface program, that serves as link between stage one and stage two of the approaches.

Before, presenting the mathematical formulations a reference should be made to the parameter that functions *a priori* to the execution of the solving of the model each time and defines that same execution. And that is *day* that we want the model to run for, meaning that results are produced for each day of the week separately and then combined into one weekly schedule in the *post-optimization analysis*. Given that *day* is selected and the right set of parameters for the model are chosen then, the following mathematical formulations need to be solved in order to acquire solutions to the problem. The algorithm according to which the pre-optimization phase works is based, is presented below.

Algorithm: Pre-optimization stage of the approach	
Input:	GTFS file (i.e. agency.txt, trips.txt, stops.txt, stop_times.txt, calendar.txt, routes.txt)
Output:	Parts of the network to consider for the optimization of Waiting times according to time and distance parameters
Steps:	
1:	Read the files, identify departure and arrival events at *all* stops of the network;
2:	Filter network events according to *time* and *distance* parameters;
3:	Group filtered events into events for each day;
4:	For each day of the week:
4.1:	Define which decision variables need to be created to facilitate the model;
4.2:	Define which constraints and what objective function should be incorporated in the model;
4.3:	Calculate transition times between schronized stops included in the model according to steps 4.1 & 4.2;
4.4:	Calculate route frequencies for each route and trip;
5:	Write all calculated data to files to be read by an interface to the model.

Figure 9: Algorithm used in the pre-optimization stage of the approach

Also, given the fact that all codes were included in *Section 8* of this thesis in the Appendix, below we will provide a list of correspondence between the Steps of the algorithm and the section of the thesis that includes the code.

Step of the algorithm	Thesis section with corresponding code
1 & 2	Section 8.2.2
3 & 4	Sections 8.2.3, 8.2.4, 8.2.5, 8.2.6, 8.2.7
5	Occurs for each file of code and corresponding Section

Table 1: Includes the correspondence between Steps of the algorithm of the pre-optimization phase and the Section of the Appendix that include the code in the end of the thesis

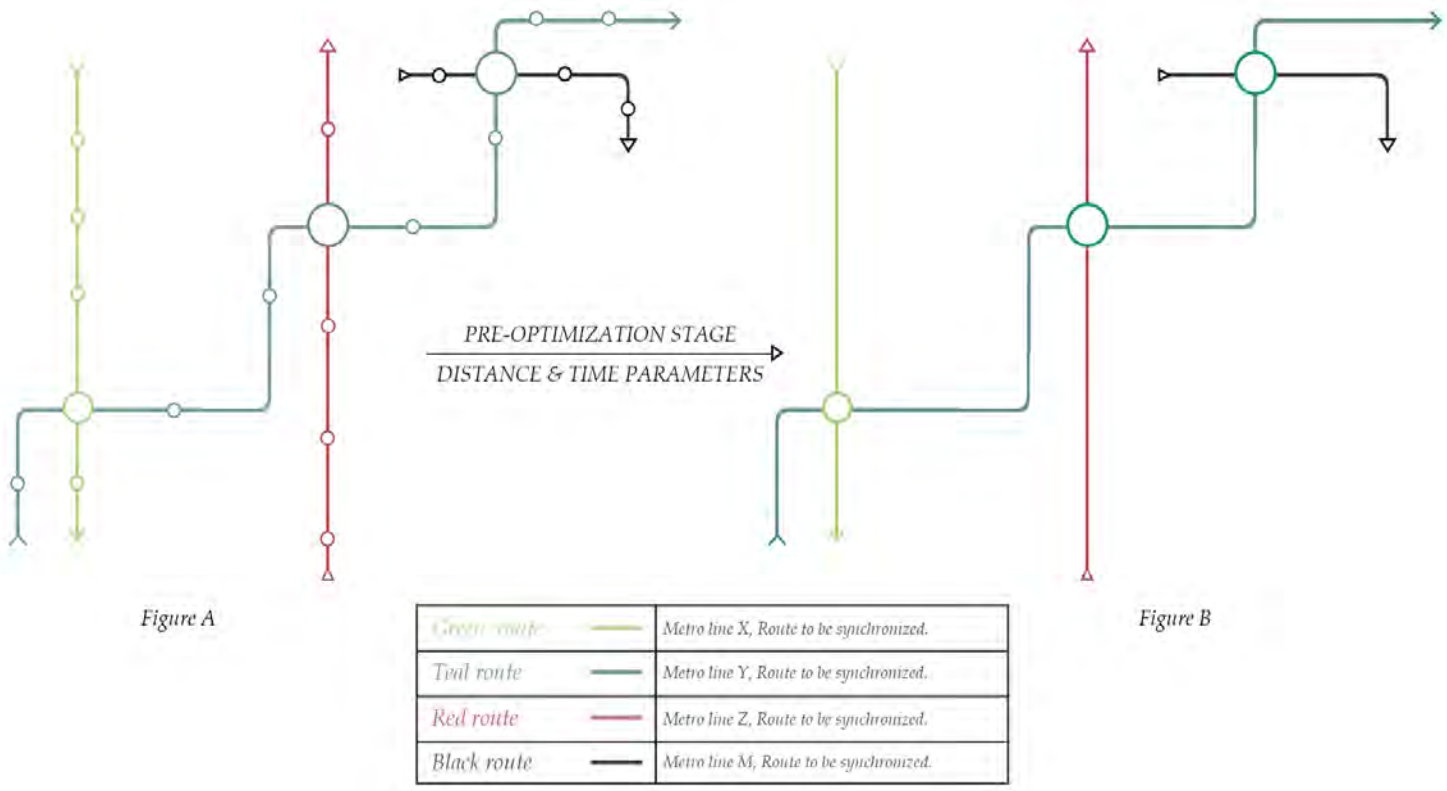

Figure 10: Graphical example of the effect of precomputation phase on a given network

4.4.2 The mathematical model

During this stage the most important part of the analysis takes place. Based on the existing schedule the relations between the services are being built up and expressed in a MILP model. The model is then solved with the CPLEX optimization studio solver and the best solution is derived. As we will show next, this solution of the mathematical model indicates how th service times of the routes should change in order to minimize waiting time at nodes of the network that serve multiple routes.

4.4.2.1 Indices, Sets, Parameters and Decision variables

Indices	
r	Index used to indicate a route of the network;
t	Index used to indicate a trip of the network;
s	Index used to indicate a stop in the network.

Sets	
R	Set of total number of routes existing in the network;
T	Set of total number of trips existing in the network;
S	Set of total number of stops existing in the network.

Parameters	
$\overline{NSDT_{srt}}$	Non-synchronized departure times for route *r* and trip *t* at stop *s*;
$\overline{NSAT_{srt}}$	Non-synchronized arrival times for route *r* and trip *t* at stop *s*;
$\overline{AP_{rt}}$	*Allowed Percentage* of change in departure or arrival time for route *r* (determined in pre-optimization);
$\overline{RF_{rt}}$	*Route Frequency* of the execution of the trips for a route *r* (calculated in pre-optimization);
$\overline{TT_{s_1 s_2 rt}}$	*Transportation time between stop s1, stop s2, route r and trip t.*

Decision Variables	
TOS_{srt}	Time when route *r* and trip *t* of that route serves the stop *s* of the network.

4.4.2.2 Constraints and Objective function of the mathematical model for the third instance

Constraints	
$\overline{NSDT_{srt}} - (\overline{AP_r} * \overline{RF_r}) \leq TOS_{srt} \leq \overline{NSDT_{srt}} + (\overline{AP_r} * \overline{RF_r})$ $\forall\ s, r, t$	Constraint (22)
$TOS_{s_1rt} + \overline{TT_{s_1s_2rt}} = TOS_{s_2rt}$ $\forall\ s, r, t$	Constraint (23)
$TOS_{s_1r_1t_1} - TOS_{s_2r_2t_2} \geq TR_{s_1,r_1,t_1,s_2,r_2,t_2}$	Constraint (24)
$TOS_{srt}\ \in\ Z^+$	

Objective function
$MINIMIZE \sum (TOS_{s_1r_1t_1} - TOS_{s_2r_2t_2})$

All of the constraints and the Objective function of the model are introduced according to selected combinations of $s_1, s_2, r_1, r_2, t_1, t_2$ that are decided upon the pre-optimization stage/algorithm of the algorithm.

4.4.3 Post-optimization stage

After the numerical results are extracted from the solutions, it is needed that they are combined into a schedule that can be applied to public transportation system under real world circumstances. Since the analysis that we did was focused on the welfare of the passengers, the numerical results can greatly vary from the ones that are actually applicable. This is the goal of

this final step, to actually turn numerical quantities calculated into meaningful real-world services.

Of course, as mentioned before, the participation of agencies is essential for the final outcome. The results of the numerical tests can be applied according to a wide variety of ways, and a great variety of solutions is acquired. All of the solutions refer to each day of the week but differ in terms of how much the trips are allowed to change. From the perspective of the operators, for a particular day there may exist the constraints on the allowed percentage of change in the execution times but in other days the demand may be different and, as a result, greater changes are allowed in the existing timetables. It is crucial then for the efficiency of our approach to prioritize in this final step the needs of the operators of the services. After providing the operator with the results table, it was decided that they could provide us with information on what are the most suitable changes for them. This could be signified simply by indicating the desired level of change for each day. That input can be further interpreted in the following ways:

1. On a first level, the desired changes indicated by the operators can be straightly applied and returned to them as a schedule. The final results will be as desired and certain level of waiting time reduction can be achieved.
2. On the contrary, the input can be then used again in the models and new constraints can be introduced in order for the values of the trips to change based on the values given by the operators. The final results will change again, but because of the intermediate optimization problem the waiting time will be slightly improved.
3. As an intermediate solution, some of the trips could be kept as is and some could be inserted in a MILP model for further optimization. This third way to interpret the input of the operators is considered as the sweet-spot solution.

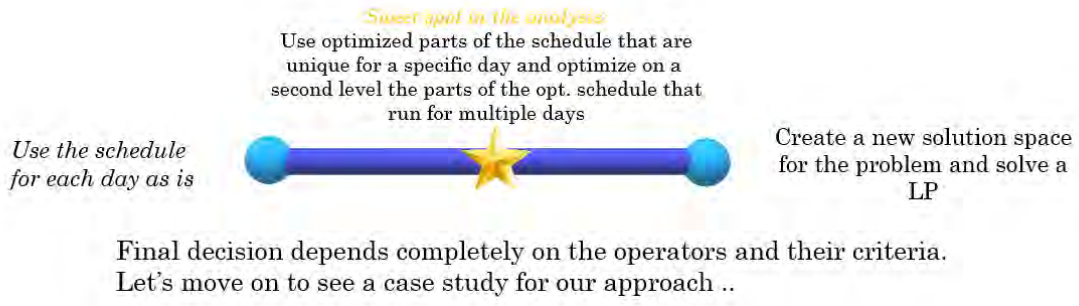

Figure 11: Graphical representation of how the analysis occurs in the post-optimization phase

In any case, after the third step a new schedule has emerged after following our solution approach. Although the decision of the schedule depends exclusively on the operators, this final step ensures that their criteria are incorporated into our result. Ensuring the better welfare of all beneficiaries in such systems is key to a successful approach and compromising the satisfaction of one part of the system for improving the other is many times unavoidable.

5 Results

In this section, the results that were acquired from the analysis of several case studies are presented. Across all of the cases, the results indicate the practicality of the approaches. For the implementation of our approaches, C++, IBM's C++ Concert API and ILOG CPLEX solver (version 12.7) are used. For a better comparison of the CPU times needed for the solution of the models, the computer system used was an Intel Core i7-6700HQ Processor with 8 Gigabytes of RAM.

Regarding the format of the *results tables* that are provided below for each case, a general outline is adopted. It is based on the fact the results vary for the several cases of experiments that were run. In respect, the rows of the tables below, refer to resulting quantities that occurred according to the parameters that are related to the upper and lower bounds that were set for the departure time of the trips. Parameters varied from case to case and some of the matrices may contain more rows than others. On the contrary, there was a standard set of columns for every result table. In each table, the columns refer to the *Total waiting time* in the network and the *Average waiting time* per passenger. Also, the corresponding percentages of change of those quantities are given, in comparison with the initial values of those statistics of the system before its optimization. This initial setting of the services, before adjustments are applied, is also referred to as the *Current* situation.

5.1 The theoretical 1-node case study

After analyzing the 1-node case results were acquired. Although the results are not associated with any physical meaning they provide insights to the expected behavior of a real-world system when subjected to optimization according to the techniques developed in this thesis. Parameters that were chosen for the experiments were set according to the matrix below.

$\overline{RFr}$	
First route	306.25
Second route	1
Third route	226

And:

$\overline{APr}$	
First route	0.1
Second route	0.1
Third route	0.1

And the final results are provided below:

	Total waiting time	%	Av. Waiting time	%
Current	1948 min	-	129.86 m	-
$\overline{AP_{rt}}, \overline{RF_{rt}}$	1644 min	15.7 %	109.6 m	15.69 %

5.2 The theoretical 2-sequential-nodes case study

Similarly, to the theoretical 1-node case study the parameters for this experiment were set indicated in the matrix below:

$\overline{RFr}$	
First route	2
Second route	15
Third route	300
Forth route	21
Fifth route	487
Sixth route	2
Seventh route	15
Eighth route	300
Ninth route	3
Tenth route	300

And:

$\overline{APr}$	
First route	0.5
Second route	0.334
Third route	1
Forth route	0.55
Fifth route	0.334
Sixth route	0.5
Seventh route	0.334
Eighth route	1
Ninth route	0.334
Tenth route	1

And the final results are provided below:

	Total waiting time	%	Av. Waiting time	%
Current	8280 min	-	197.14 m	-
$\overline{AP_{rt}}, \overline{RF_{rt}}$	1326 min	83.98%	74.42 m	62.25 %

5.3 Athens metro case study

For the last instance of the problem and the corresponding modelling that was introduced, we applied it to the *Athens Metro system*. With some of its sections operating since 1869, Athens Metro is a system a rapid-transit system that serves the areas of Athens and the wider area of Attiki in Greece. While it is composed of 3 major lines, in many stops of its network there is a connection with other modes of transport and especially with trains of TRAINOSE. Out of the 61 stations that belong to network 57 are served by the Metro services alone, TRAINOSE S.A. serves the remainder of the stations. Below we provide an image of the initial network.

Figure 12: Athens metro network; Initial network. Image initially from Attiko Metro website [22].

Now given the fact that this network was analyzed and optimized according to the third approach, the initial network has been simplified in order for it to be considered *holistically*. In other words, while this third approach considers transitions in every stop (i.e. node) of the network, they it does not include *all* transitions at every stop. Based on the values of the *time* and *distance parameters*, a simplified network is produced. As discussed, they are utilized in a pre-optimization phase, before the solution to the mathematical model is attempted. During the pre-optimization stage, the algorithm is run that produces a simplified network for the initial data according to the values assigned to those two parameters. For networks such as bus, train or multi-modal networks, the choice of the values for parameters matters a lot in the final network that is produced. For the different values of the parameters that are decided by the officer that runs the experiment, a greater or lesser number of stops and feasible transitions may be included in the model. Transitional stops and feasible transitions are determined in pre-optimization and are constant in the main optimization phase in this modelling of the problem. Only departure times change for the network that is analyzed. The figure below depicts both the initial and simplified network for *Athens Metro system*.

Figure 13: The simplified network for the case of Athens Metro; Result of pre-optimization stage. Image initially from Attico Metro Website [22].

As for our case of *Athens Metro system*, in the case of metro the *distance parameter* has a different meaning since transitions from one route to another can happen only in metro stations and, usually, they are separated from one another for what could be considered long enough distances. As for the *time parameter*, for the experiments that were run below, transitions that happen within *20 minutes* at each single station were considered.

	Total waiting time	%	Av. Waiting time	%
Current	290520 min	-	11m	-
$\overline{AP_{rt}}, \overline{RF_{rt}}$	285566 min	1.7 %	10.5m	1.69 %
15-min relaxation	280122 min	3.57 %	10.23m	3.58 %
30-min relaxation	271122 min	6.67 %	9.9m	6.69 %
45-min relaxation	264236 min	9.04 %	9.65m	9.04 %

Table 2: Results table for the case of synchronization of the network of Athens Metro System

6 Conclusion

This study addressed the problem of minimizing waiting times for the passengers of public transport networks. Three instances of the problem were discussed, analyzed in depth and corresponding MILP models were introduced for the solution of the problems. The most substantial part of our work is the third approach which is applied to a real-world system. Also, the pre-optimization and post-optimization parts of the approach are also crucial for the successful extraction of meaningful results. While the modelling approaches had different advantages and disadvantages compared to each other, each one of them is able to introduce small or greater changes to the network according to the values of the parameters that were chosen.

There are several ways in which our approaches could be improved. The most important factor would be the inclusion of more parameters into the model, in such a way that more of the constraints of the physical system are expressed. Such a parameter would be the dwell time of vehicles at some stops of the network. By introducing them into the modelling, the initial departure times at the first stops could be kept the same and only the arrival and departure times of the service in the rest of stops of the route could be altered. In that manner, more feasible transitions could be enabled and fewer waiting time achieved with fewer changes in the initial structure of the timetable. Another important parameter to include would be the demand of the passengers and, finally, the schedule of the drivers, which is very crucial for the way that the system operates.

7 Bibliography

[1] TNS Political & Social, "Europeans' satisfaction with urban transport," European Commission, 2014.

[2] Hannah Bast, Daniel Delling, Andrew V. Golderg, Matthias Muller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, Renato F. Werneck, "Route Planning in Transportation Networks," 2014.

[3] G. B. Dantzig, J. H. Ramser, "The Truck Dispatching Problem," *Management Science,* vol. Vol. 6, pp. 80-91, 1959.

[4] Georgios K.D. Saharidis, Dimitrios Rizopoulos, Antonios Fragogios, Chrysostomos Chatzigeorgiou, "A hybrid approach to the problem of journey planning with the use of mathematical programming and modern techniques," *Transportation Research Procedia,* vol. 24, pp. 401 - 409, 2017.

[5] Georgios K.D. Saharidis, George Kolomvos, George Liberopoulos, "Modelling and solution approach for the environmental travelling salesaman problem," *Transport Research Arena,* 2014.

[6] C. LLC, "OpenTripPlanner," [Online]. Available: https://github.com/opentripplanner.

[7] "GreenYourMove project," [Online]. Available: https://www.greenyourmove.org/.

[8] M. R. Bussieck, T. Winter, U.T. Zimmerman, "Discrete optimization in public rail transport," *Article of Mathematical Programming,* vol. 79, no. 1, pp. 415-444, 1997.

[9] R. Goverde, "Synchronization Control of Scheduled Train Services to minimize Passenger Waiting Time," *Proceedings of the 4th TRAIL Congress-Transport Infrastructure and Logistics,* 1998.

[10] Ceder A., Golany B., Tal O., "Creating bus timetables with maximal synchronization," *Transp Res,* vol. A 35, pp. 913-928, 2001.

[11] A. Erranki, A model to create bus timetables to attain maximum synchronization considering waiting times at transfer stops, University of Florida Scholar Commons, 2004.

[12] Omar J.Ibarra-Rojas, Yasmin A.Rios-Solis, "Synchronization of bus timetabling," *Transportation Research Part B: Methodological,* vol. 46, no. 5, pp. 599-614, 2012.

[13] Georgios Saharidis, Charalampos Dimitropoulos, Erotokritos Skordilis, "MInimizing waiting time at transitional nodes for public bus transportation in Greece," *Operational research,* vol. 14, no. 3, p. 341–359, 2013.

[14] Eva Barrena, David Canca, Leandro C. Coelho, Gilbert Laporte, "Single-line rail rapid transit timetabling under dynamic passenger demand," *Transportation Research Part B,* vol. 70, pp. 134-150, 2014.

[15] Eva Barrena, David Canca, Leandro C. Coelho, Gilbert Laporte, "Exact formulations and algorithm for the train timetabling problem with dynamic demand," *Computers & Operations Research,* vol. 44, pp. 66-74, 2014.

[16] S. H. Z. Erfan Hassannayebi, "Variable and adaptive neighbourhood search algorithms for rail rapid transit timetabling problem," *Computers & Operations Research,* vol. 78, pp. 439-453, 2017.

[17] M. D. Q. L. Randolph Hall, "Optimal holding times at transfer stations," *Computers & Industrial engineering,* vol. 40, no. 4, pp. 379-397, 2001.

[18] T. C. D. P. Line Blander Reinhardt, "Synchronized dial-a-ride transportation of disabled passengers at airports," *European Journal of Operational Research,* vol. 225, no. 1, pp. 106-117, 2013.

[19] J. G. J. D.-H. L. K. W. A. A. E. Lijun Sun, "Demand-driven timetable design for metro services," *Transportation Research Part C: Emerging Technologies,* vol. 46, pp. 284-299, 2014.

[20] Rachel C. W. Wong, Tony W. Y. Yuen, Kwok Wah Fung, Janny M. Y. Leung, "Optimizing timetable synchronization for rail mass transit," vol. 42, no. 1, 2008.

[21] A.M. Geoffrion; G.W. Graves, "Multicommodity System Design by Benders Decomposition," *Management Science,* vol. 20, no. 5, pp. 822-844, 1974.

8Appendix

In this appendix, all the codes that have been developed for this thesis are attached. Starting with the most important files that include the implementation of the mathematical models and later on in this section we include code developed for several functions that served as utilities to the scientific approaches.

For the better understanding of those codes by the readers, let's remark that the codes are part of a Visual studio 2013 Project file that should be used for the reproduction of the approaches. It can be requested from the authors of this thesis.

8.1 main.cpp – Main file, used for the execution of the functions and routines

```
1.  #include "stdafx.h"
2.  #include <time.h>
3.  #include <iostream>
4.  #include "basicModel.h"
5.  #include "preOptimization.h"
6.  #include "interfaceToModel_dailyList.h"
7.  #include "interfaceToModel_decisionVars.h"
8.  #include "interfaceToModel_parameters.h"
9.  #include "interfaceToModel_transitionTimes.h"
10. #include "interfaceToModel_objFun.h"
11. #include "oneStationProblem.h"
12. #include "twoStationProblem.h"
13. #include "athensMetroCaseStudyTwoTripsPerRoute.h"
14. #include "athensMetroCaseStudyFourTripsPerRoute.h"
15. #include "athensMetroCaseStudyAllTripsPerRoute.h"
16.
17. using namespace std;
18.
19. int main(int argc, char **argv)
20. {
21.
22.     clock_t tStart = clock();
23.
24.     int a = preOptimization();
25.
26.     int b = interfaceToModel_dailyList();
27.     int c = interfaceToModel_decisionVars();
28.     int d = interfaceToModel_parameters();
29.     int e = interfaceToModel_transitionTimes();
30.     int f = interfaceToModel_objFun();
31.
32.     int g = athensMetroAllTrips_1();
33.
34.     cout << endl;
35.     printf("Time needed up to this point: %.2fs\n", (double)(clock() - tStart) / CLOCKS_PER_SEC);
36.     cout << endl;
37.
38.     return 1;
```

```
39. }
```

8.2 Pre-optimization procedures

8.2.1 identifyIndex.cpp – File to identify index of variable sued to represent an event

```
1.  #include <ilcplex/ilocplex.h>
2.  #include <time.h>
3.  #include <vector>
4.  #include <sstream>
5.  #include <fstream>
6.  #include <string>
7.  #include <iostream>
8.  #include <chrono>
9.  #include <map>
10. #include <thread>
11. #include <list>
12. #include <stdio.h>
13. #include <array>
14. #include <string>
15. #include "identifyIndex.h"
16. #include "conversionFromHHMMSStoContinuousTimeRepresentation.h"
17.
18. using namespace std;
19.
20. int findStopIndex(std::string stopID)
21. {
22.
23.     //cols of the gtfs, fixed for any gtfs file
24.     int const stops_cols = 8;
25.
26.     //rows of the gtfs, change for each gtfs file
27.     int const stops_rows = 7898;
28.
29.     string stops_data[stops_rows][stops_cols];
30.     std::ifstream file1("Data folder/athens_metro/stops.txt");
31.
32.     for (int row = 0; row < stops_rows; row++)
33.     {
34.         std::string line;
35.         std::getline(file1, line);
36.
37.         std::stringstream iss(line);
38.
39.         for (int col = 0; col < stops_cols; ++col)
40.         {
41.             std::string val;
42.             std::getline(iss, val, ',');
43.
44.             std::stringstream convertor(val);
45.             convertor >> stops_data[row][col];
46.
47.             if (!iss.good())
48.                 break;
```

```
49.         }
50.
51.         if (!file1.good())
52.         break;
53.     }
54.
55.     int keepRow;
56.     for (int row = 0; row < stops_rows; row++)
57.     {
58.         if (stops_data[row][0] == stopID)
59.         {
60.             keepRow = row;
61.         }
62.     }
63.
64.     return keepRow + 1;
65. }
66.
67. int findTripIndex(std::string tripID)
68. {
69.     int const trips_cols = 7;
70.     int const trips_rows = 3636;
71.
72.     string trips_data[trips_rows][trips_cols];
73.     std::ifstream file2("Data folder/athens_metro/trips.txt");
74.
75.     for (int row = 0; row < trips_rows; ++row)
76.     {
77.         std::string line;
78.         std::getline(file2, line);
79.
80.         std::stringstream iss(line);
81.
82.         for (int col = 0; col < trips_cols; ++col)
83.         {
84.             std::string val;
85.             std::getline(iss, val, ',');
86.             if (!iss.good())
87.                 break;
88.
89.             std::stringstream convertor(val);
90.             convertor >> trips_data[row][col];
91.         }
92.         if (!file2.good())
93.             break;
94.     }
95.
96.     int keepRow;
97.     for (int row = 0; row < trips_rows; ++row)
98.     {
99.         if (trips_data[row][2] == tripID)
100.        {
101.            keepRow = row;
102.        }
103.    }
104.
105.    return keepRow + 1;
```

```
106. }
107.
108. int findRouteIndex(std::string routeID)
109. {
110.     int const routes_cols = 9;
111.     int const routes_rows = 4;
112.
113.     string routes_data[routes_rows][routes_cols];
114.     std::ifstream file1("Data folder/athens_metro/routes.txt");
115.
116.     for (int row = 0; row < routes_rows; ++row)
117.     {
118.         std::string line;
119.         std::getline(file1, line);
120.
121.         std::stringstream iss(line);
122.
123.         for (int col = 0; col < routes_cols; ++col)
124.         {
125.             std::string val;
126.             std::getline(iss, val, ',');
127.
128.             std::stringstream convertor(val);
129.             convertor >> routes_data[row][col];
130.
131.             if (!iss.good())
132.                 break;
133.         }
134.         if (!file1.good())
135.             break;
136.     }
137.
138.     int keepRow;
139.     for (int row = 0; row < routes_rows; ++row)
140.     {
141.         if (routes_data[row][0] == routeID)
142.         {
143.             keepRow = row;
144.         }
145.     }
146.
147.     return keepRow + 1 ;
148. }
149.
150. std::vector<string> findStopByIndex(int fileRow)
151. {
152.     //cols of the gtfs, fixed for any gtfs file
153.     int const stops_cols = 8;
154.
155.     //rows of the gtfs, change for each gtfs file
156.     int const stops_rows = 7898;
157.
158.     string stops_data[stops_rows][stops_cols];
159.     std::ifstream file1("Data folder/athens_metro/stops.txt");
160.
161.     for (int row = 0; row < stops_rows; ++row)
162.     {
```

```
163.        std::string line;
164.        std::getline(file1, line);
165.
166.        std::stringstream iss(line);
167.
168.        for (int col = 0; col < stops_cols; ++col)
169.        {
170.            std::string val;
171.            std::getline(iss, val, ',');
172.
173.            std::stringstream convertor(val);
174.            convertor >> stops_data[row][col];
175.
176.            if (!iss.good())
177.                break;
178.        }
179.
180.        if (!file1.good())
181.            break;
182.    }
183.
184.    std::vector<string> stop;
185.    for (int column = 0; column < stops_cols; column++)
186.    {
187.        stop.push_back(stops_data[fileRow][column]);
188.    }
189.
190.    return stop;
191. }
192.
193. std::vector<string> findTripByIndex(int fileRow)
194. {
195.    int const trips_cols = 7;
196.    int const trips_rows = 3636;
197.
198.    string trips_data[trips_rows][trips_cols];
199.    std::ifstream file2("Data folder/athens_metro/trips.txt");
200.
201.    for (int row = 0; row < trips_rows; ++row)
202.    {
203.        std::string line;
204.        std::getline(file2, line);
205.
206.        std::stringstream iss(line);
207.
208.        for (int col = 0; col < trips_cols; ++col)
209.        {
210.            std::string val;
211.            std::getline(iss, val, ',');
212.            if (!iss.good())
213.                break;
214.
215.            std::stringstream convertor(val);
216.            convertor >> trips_data[row][col];
217.        }
218.        if (!file2.good())
219.            break;
```

```
220.     }
221.
222.     std::vector<string> trip;
223.     for (int column = 0; column < trips_cols; column++)
224.     {
225.         trip.push_back(trips_data[fileRow][column]);
226.     }
227.
228.     return trip;
229.
230.
231. }
232.
233. std::vector<string> findRouteByIndex(int fileRow)
234. {
235.     int const routes_cols = 9;
236.     int const routes_rows = 4;
237.
238.     string routes_data[routes_rows][routes_cols];
239.     std::ifstream file1("Data folder/athens_metro/routes.txt");
240.
241.     for (int row = 0; row < routes_rows; ++row)
242.     {
243.         std::string line;
244.         std::getline(file1, line);
245.
246.         std::stringstream iss(line);
247.
248.         for (int col = 0; col < routes_cols; ++col)
249.         {
250.             std::string val;
251.             std::getline(iss, val, ',');
252.
253.             std::stringstream convertor(val);
254.             convertor >> routes_data[row][col];
255.
256.             if (!iss.good())
257.                 break;
258.         }
259.         if (!file1.good())
260.             break;
261.     }
262.
263.     std::vector<string> route;
264.     for (int column = 0; column < routes_cols; column++)
265.     {
266.         route.push_back(routes_data[fileRow][column]);
267.     }
268.
269.     return route;
270. }
```

8.2.2 preOptimization.cpp – Main pre Optimization file, runs first, before interface to model files are run

```
1.  //dimrizo
2.  #include <iostream>
3.  #include <sstream>
4.  #include <fstream>
5.  #include <windows.h>
6.  #include <iostream>
7.  #include <locale.h>
8.  #include <codecvt>
9.  #include <time.h>
10. #include <typeinfo>
11. #include <list>
12. #include <vector>
13. #include "preOptimization.h"
14. #include "havershineDistance.h"
15. #include "conversionFromHHMMSStoContinuousTimeRepresentation.h"
16.
17. using namespace std;
18.
19. int preOptimization()
20. {
21.
22.     //std::locale::global(std::locale(std::locale::empty(), new std::codecvt_utf8<wchar_t>));
23.     //SetConsoleOutputCP(65001);
24.
25.     clock_t tStart = clock();
26.
27.     //cols of the gtfs, fixed for any gtfs file
28.     int const stops_cols = 8;
29.     int const routes_cols = 9;
30.     int const trips_cols = 7;
31.     int const calendar_cols = 10;
32.     int const stop_times_cols = 7;
33.
34.     //rows of the gtfs, change for each gtfs file
35.     int const stops_rows = 7898;
36.     int const routes_rows = 4;
37.     int const trips_rows = 3636;
38.     int const calendar_rows = 554;
39.     int const stop_times_rows = 69863;
40.
41.     int timeParameter = 20;
42.     int distanceParameter = 30;
43.
44.     //loading of the GTFS
45.
46.     //start of reading stops.txt
47.
48.     string stops_data[stops_rows][stops_cols];
49.     std::ifstream file("Data folder/athens_metro/stops.txt");
50.
51.     for (int row = 0; row < stops_rows; ++row)
52.     {
```

```
53.         std::string line;
54.         std::getline(file, line);
55.
56.         std::stringstream iss(line);
57.
58.         for (int col = 0; col < stops_cols; ++col)
59.         {
60.             std::string val;
61.             std::getline(iss, val, ',');
62.
63.             std::stringstream convertor(val);
64.             convertor >> stops_data[row][col];
65.
66.             if (!iss.good())
67.                 break;
68.         }
69.         if (!file.good())
70.             break;
71.     }
72.
73.     //end of reading stops.txt
74.     //start of reading routes.txt
75.
76.     /*string routes_data[routes_rows][routes_cols];
77.     std::ifstream file1("Data folder/athens_metro/routes.txt");
78.
79.     for (int row = 0; row < routes_rows; ++row)
80.     {
81.         std::string line;
82.         std::getline(file1, line);
83.
84.         std::stringstream iss(line);
85.
86.         for (int col = 0; col < routes_cols; ++col)
87.         {
88.             std::string val;
89.             std::getline(iss, val, ',');
90.
91.             std::stringstream convertor(val);
92.             convertor >> routes_data[row][col];
93.
94.             if (!iss.good())
95.                 break;
96.         }
97.         if (!file1.good())
98.             break;
99.     }*/
100.
101.    //end of reading routes.txt
102.    //start of reading trips.txt
103.
104.    string trips_data[trips_rows][trips_cols];
105.    std::ifstream file2("Data folder/athens_metro/trips.txt");
106.
107.    for (int row = 0; row < trips_rows; ++row)
108.    {
109.        std::string line;
```

```
110.        std::getline(file2, line);
111.
112.        std::stringstream iss(line);
113.
114.        for (int col = 0; col < trips_cols; ++col)
115.        {
116.            std::string val;
117.            std::getline(iss, val, ',');
118.            if (!iss.good())
119.            break;
120.
121.            std::stringstream convertor(val);
122.            convertor >> trips_data[row][col];
123.        }
124.        if (!file2.good())
125.            break;
126.    }
127.
128.    //end of reading trips.txt
129.    //start of reading calendar.txt
130.
131.    string calendar_data[calendar_rows][calendar_cols];
132.    std::ifstream file3("Data folder/athens_metro/calendar.txt");
133.
134.    for (int row = 0; row < calendar_rows; ++row)
135.    {
136.        std::string line;
137.        std::getline(file3, line);
138.
139.        std::stringstream iss(line);
140.
141.        for (int col = 0; col < calendar_cols; ++col)
142.        {
143.            std::string val;
144.            std::getline(iss, val, ',');
145.            if (!iss.good())
146.            break;
147.
148.            std::stringstream convertor(val);
149.            convertor >> calendar_data[row][col];
150.        }
151.        if (!file3.good())
152.            break;
153.    }
154.
155.    //end of reading calendar.txt file
156.    //start of reading stop_times.txt file
157.
158.    std::string stop_times_data[stop_times_rows][stop_times_cols];
159.    std::ifstream file4("Data folder/athens_metro/stop_times.txt");
160.
161.    for (int row = 0; row < stop_times_rows; row++)
162.    {
163.        std::string line;
164.        std::getline(file4, line);
165.
166.        std::stringstream iss(line);
```

```
167.
168.        for (int col = 0; col < stop_times_cols; col++)
169.        {
170.            std::string val;
171.            std::getline(iss, val, ',');
172.
173.            std::stringstream convertor(val);
174.            convertor >> stop_times_data[row][col];
175.
176.            if (!iss.good())
177.                break;
178.        }
179.
180.        if (!file4.good())
181.            break;
182.    }
183.
184.    /*for (int row = 0; row < stops_rows; row++)
185.    {
186.    for (int col = 0; col < stops_cols; col++)
187.    {
188.    cout << stops_data[row][col] << " ";
189.    }
190.    cout << endl;
191.    }*/
192.
193.    std::vector<vector<string>>        waitingTimeEvents;
194.
195.    double  latOfEvent1;
196.    double  lonOfEvent1;
197.    double  latOfEvent2;
198.    double  lonOfEvent2;
199.    int     timeOfEvent1;
200.    int     timeOfEvent2;
201.    string  stopIDOfEvent1;
202.    string  stopIDOfEvent2;
203.    double  distanceBetweenEvents;
204.    double  div_result;
205.    bool    doubleCheckEventsB = false;
206.    bool    doubleCheckEventsC = false;
207.    bool    addEvent = false;
208.
209.    bool MON_1  = false;
210.    bool TUE_1  = false;
211.    bool WED_1  = false;
212.    bool THUR_1 = false;
213.    bool FRI_1  = false;
214.    bool SAT_1  = false;
215.    bool SUN_1  = false;
216.
217.    bool MON_2  = false;
218.    bool TUE_2  = false;
219.    bool WED_2  = false;
220.    bool THUR_2 = false;
221.    bool FRI_2  = false;
222.    bool SAT_2  = false;
223.    bool SUN_2  = false;
```

```
224.
225.    int counterOfFeasibleTransitions = 0;
226.
227.    std::vector<string> previous_event_1;
228.    std::vector<string> previous_event_2;
229.
230.    string tripID_1;
231.    string tripID_2;
232.
233.    std::vector<bool> scheduleOfEvent_1;
234.    std::vector<bool> scheduleOfEvent_2;
235.
236.    string tripRowID;
237.    string currentCalendar;
238.    string calendarRowID;
239.
240.    string tripRowID_2;
241.    string currentCalendar_2;
242.    string calendarRowID_2;
243.
244.    string routeID_1;
245.    string routeID_2;
246.
247.    int keepCalendarRow;
248.    int keepCalendarRow_1;
249.
250.    for (int row_1 = 1; row_1 < stop_times_rows; row_1++)
251.    {
252.
253.        //cout << "Line : " << row_1 + 1 << endl;
254.
255.        scheduleOfEvent_1.clear();
256.        waitingTimeEvents.clear();
257.        routeID_1.clear();
258.        keepCalendarRow_1;
259.
260.        MON_1   = false;
261.        TUE_1   = false;
262.        WED_1   = false;
263.        THUR_1  = false;
264.        FRI_1   = false;
265.        SAT_1   = false;
266.        SUN_1   = false;
267.
268.        timeOfEvent1 = conversionToContinuousTime(stop_times_data[row_1][2]);
269.
270.        //detection of the X,Y of the stop of the event
271.        stopIDOfEvent1 = stop_times_data[row_1][3];
272.        for (int row = 1; row < stops_rows; ++row)
273.        {
274.            if (stopIDOfEvent1 == stops_data[row][0])
275.            {
276.                lonOfEvent1 = atof(stops_data[row][5].c_str());
277.                latOfEvent1 = atof(stops_data[row][4].c_str());
278.                break;
279.            }
280.        }
```

```
281.
282.     //get at what days it is executed
283.     tripID_1 = stop_times_data[row_1][0];
284.
285.     for (int trip_row = 1; trip_row < trips_rows; trip_row++)
286.     {
287.       if (trips_data[trip_row][2] == tripID_1)
288.       {
289.         currentCalendar = trips_data[trip_row][1];
290.         break;
291.       }
292.     }
293.
294.     //identify currentCalendar among all Calendars available at calendars.txt
295.     for (int calendar_row = 1; calendar_row < calendar_rows; calendar_row++)
296.     {
297.
298.       if (calendar_data[calendar_row][0] == currentCalendar)
299.       {
300.         // SEE AT WHAT DAYS IS THE TRIP EXECUTED
301.         keepCalendarRow_1 = calendar_row;
302.         if (calendar_data[calendar_row][1] == "1")
303.         {
304.           //timeOfMONEvent_1 = weeklyProjectionOfEvents(stop_times_data[row_1][2], "MON");
305.           MON_1 = true;
306.         }
307.         if (calendar_data[calendar_row][2] == "1")
308.         {
309.           //timeOfTUEEvent_1 = weeklyProjectionOfEvents(stop_times_data[row_1][2], "TUE");
310.           TUE_1 = true;
311.         }
312.         if (calendar_data[calendar_row][3] == "1")
313.         {
314.           //timeOfWEDEvent_1 = weeklyProjectionOfEvents(stop_times_data[row_1][2], "WED");
315.           WED_1 = true;
316.         }
317.         if (calendar_data[calendar_row][4] == "1")
318.         {
319.           //timeOfTHUREvent_1 = weeklyProjectionOfEvents(stop_times_data[row_1][2], "THUR");
320.           THUR_1 = true;
321.         }
322.         if (calendar_data[calendar_row][5] == "1")
323.         {
324.           //timeOfFRIEvent_1 = weeklyProjectionOfEvents(stop_times_data[row_1][2], "FRI");
325.           FRI_1 = true;
326.         }
327.         if (calendar_data[calendar_row][6] == "1")
328.         {
329.           //timeOfSATEvent_1 = weeklyProjectionOfEvents(stop_times_data[row_1][2], "SAT");
330.           SAT_1 = true;
331.         }
332.         if (calendar_data[calendar_row][7] == "1")
333.         {
334.           //timeOfSUNEvent_1 = weeklyProjectionOfEvents(stop_times_data[row_1][2], "SUN");
335.           SUN_1 = true;
336.         }
337.         break;
```

```
338.          }
339.      }
340.
341.      scheduleOfEvent_1.push_back(MON_1);
342.      scheduleOfEvent_1.push_back(TUE_1);
343.      scheduleOfEvent_1.push_back(WED_1);
344.      scheduleOfEvent_1.push_back(THUR_1);
345.      scheduleOfEvent_1.push_back(FRI_1);
346.      scheduleOfEvent_1.push_back(SAT_1);
347.      scheduleOfEvent_1.push_back(SUN_1);
348.
349.      int counter_2 = 0;
350.      for (int row_2 = 1; row_2 < stop_times_rows; row_2++)
351.      {
352.
353.          scheduleOfEvent_2.clear();
354.          routeID_2.clear();
355.          addEvent = false;
356.
357.          timeOfEvent2 = conversionToContinuousTime(stop_times_data[row_2][2]);
358.          tripID_2 = stop_times_data[row_2][0];
359.
360.          MON_2   = false;
361.          TUE_2   = false;
362.          WED_2   = false;
363.          THUR_2  = false;
364.          FRI_2   = false;
365.          SAT_2   = false;
366.          SUN_2   = false;
367.
368.          if ((timeOfEvent2 -
     timeOfEvent1 <= timeParameter) && (timeOfEvent2 > timeOfEvent1 + 1))          //if WT is between 2 and
     timeParameter
369.          {
370.
371.              stopIDOfEvent2 = stop_times_data[row_2][3];                          //identify stop and i
   ts lon and lat
372.              for (int row = 1; row < stops_rows; ++row)
373.              {
374.                  if (stopIDOfEvent2 == stops_data[row][0])
375.                  {
376.                      lonOfEvent2 = atof(stops_data[row][5].c_str());
377.                      latOfEvent2 = atof(stops_data[row][4].c_str());
378.                      break;
379.                  }
380.              }
381.
382.              distanceBetweenEvents = distanceEarth(latOfEvent1, lonOfEvent1, latOfEvent2, lonOfEvent2) * 100
   0;   //calculation of distance in meters
383.
384.              if (distanceBetweenEvents <= distanceParameter)                      //if distance betw
   een events is lesser than distanceParameter
385.              {
386.                  for (int trip_row_3 = 1; trip_row_3 < trips_rows; trip_row_3++)
387.                  {
388.                      if (trips_data[trip_row_3][2] == tripID_1)
389.                      {
```

```
                routeID_1 = trips_data[trip_row_3][0];
                break;
            }
        }

        for (int trip_row_3 = 1; trip_row_3 < trips_rows; trip_row_3++)
        {
            if (trips_data[trip_row_3][2] == tripID_2)
            {
                routeID_2 = trips_data[trip_row_3][0];
                break;
            }
        }

        if (routeID_1 != routeID_2)
        {
            addEvent = true;
        }

    }
}

if (addEvent == true)
{

    for (int trip_row_2 = 1; trip_row_2 < trips_rows; trip_row_2++)
    {
        if (trips_data[trip_row_2][2] == tripID_2)
        {
            currentCalendar_2 = trips_data[trip_row_2][1];
            break;
        }
    }

    //identify currentCalendar among all Calendars available at calendars.txt

    for (int calendar_row_2 = 1; calendar_row_2 < calendar_rows; calendar_row_2++)
    {
        if (calendar_data[calendar_row_2][0] == currentCalendar_2)
        {
            keepCalendarRow = calendar_row_2;
            // SEE AT WHAT DAYS IS THE TRIP EXECUTED
            if (calendar_data[calendar_row_2][1] == "1")
            {
                //timeOfMONEvent_1 = weeklyProjectionOfEvents(stop_times_data[row_1][2], "MON");
                MON_2 = true;
            }
            if (calendar_data[calendar_row_2][2] == "1")
            {
                //timeOfTUEEvent_1 = weeklyProjectionOfEvents(stop_times_data[row_1][2], "TUE");
                TUE_2 = true;
            }
            if (calendar_data[calendar_row_2][3] == "1")
            {
                //timeOfWEDEvent_1 = weeklyProjectionOfEvents(stop_times_data[row_1][2], "WED");
                WED_2 = true;
            }
```

```
447.                if (calendar_data[calendar_row_2][4] == "1")
448.                {
449.                   //timeOfTHUREvent_1 = weeklyProjectionOfEvents(stop_times_data[row_1][2], "THUR");

450.                   THUR_2 = true;
451.                }
452.                if (calendar_data[calendar_row_2][5] == "1")
453.                {
454.                   //timeOfFRIEvent_1 = weeklyProjectionOfEvents(stop_times_data[row_1][2], "FRI");
455.                   FRI_2 = true;
456.                }
457.                if (calendar_data[calendar_row_2][6] == "1")
458.                {
459.                   //timeOfSATEvent_1 = weeklyProjectionOfEvents(stop_times_data[row_1][2], "SAT");
460.                   SAT_2 = true;
461.                }
462.                if (calendar_data[calendar_row_2][7] == "1")
463.                {
464.                   //timeOfSUNEvent_1 = weeklyProjectionOfEvents(stop_times_data[row_1][2], "SUN");
465.                   SUN_2 = true;
466.                }
467.                break;
468.             }
469.          }
470.
471.          scheduleOfEvent_2.push_back(MON_2);
472.          scheduleOfEvent_2.push_back(TUE_2);
473.          scheduleOfEvent_2.push_back(WED_2);
474.          scheduleOfEvent_2.push_back(THUR_2);
475.          scheduleOfEvent_2.push_back(FRI_2);
476.          scheduleOfEvent_2.push_back(SAT_2);
477.          scheduleOfEvent_2.push_back(SUN_2);
478.
479.          counterOfFeasibleTransitions = 0;
480.          for (int counter_1 = 0; counter_1 < 7; counter_1++)
481.          {
482.             if ((scheduleOfEvent_1[counter_1] == true) && (scheduleOfEvent_2[counter_1] == true))
483.             {
484.                counterOfFeasibleTransitions += 1;
485.             }
486.          }
487.
488.          if (counterOfFeasibleTransitions > 0)
489.          {
490.             std::vector<string> event;
491.             for (int column = 0; column < stop_times_cols; column++)
492.             {
493.                event.push_back(stop_times_data[row_2][column]);
494.             }
495.             //event.push_back(to_string(counterOfFeasibleTransitions));
496.             event.push_back(calendar_data[keepCalendarRow][1]);
497.             event.push_back(calendar_data[keepCalendarRow][2]);
498.             event.push_back(calendar_data[keepCalendarRow][3]);
499.             event.push_back(calendar_data[keepCalendarRow][4]);
500.             event.push_back(calendar_data[keepCalendarRow][5]);
501.             event.push_back(calendar_data[keepCalendarRow][6]);
502.             event.push_back(calendar_data[keepCalendarRow][7]);
```

```
503.                 waitingTimeEvents.push_back(event);
504.             }
505.         }
506.     }
507.
508.     //write to file
509.     if (waitingTimeEvents.size() != 0)
510.     {
511.         stringstream ss_pathToWriteFile;
512.         ss_pathToWriteFile << "Data folder/athens_metro/toFeedToModel/eventOfLine_" << (row_1 + 1) <
    < ".txt";
513.         string pathToWriteFile = ss_pathToWriteFile.str();
514.         std::ofstream out(pathToWriteFile);
515.
516.         out << MON_1 << "," << TUE_1 << "," << WED_1 << "," << THUR_1 << "," << FRI_1 << "," << SAT_1
    << "," << SUN_1 << endl;
517.
518.         //write arriving event
519.         for (int col = 0; col < stop_times_cols; col++)
520.         {
521.             out << stop_times_data[row_1][col];
522.             if (col != stop_times_cols - 1)
523.             {
524.                 out << ",";
525.             }
526.         }
527.         out << endl;
528.
529.         //write departing events events
530.         for (int i = 0; i < waitingTimeEvents.size(); ++i)
531.         {
532.             for (int col = 0; col < stop_times_cols + 7; col++)
533.             {
534.                 out << waitingTimeEvents[i][col];
535.                 if (col != stop_times_cols + 6)
536.                 {
537.                     out << ",";
538.                 }
539.             }
540.             out << endl;
541.         }
542.         out.close();
543.     }
544.
545.     div_result = div(row_1 + 1, 5000).rem;
546.     if (div_result == 0)
547.     {
548.         cout << "Line : " << row_1 + 1 << endl;
549.         printf("Time needed up to this point: %.2fs\n", (double)(clock() - tStart) / CLOCKS_PER_SEC);
550.         cout << endl;
551.     }
552.
553.     //cout << "COUNTER " << counter_2 << endl;
554.
555. }
556.
557. printf("Time taken: %.2fs\n", (double)(clock() - tStart) / CLOCKS_PER_SEC);
```

```
558.
559.     return 1;
560. }
```

8.2.3 InterfaceToModel_dailyList.cpp – Creates the daily list of events based on the day chosen as the value of a parameter

```
#include <ilcplex/ilocplex.h>
#include <time.h>
#include <vector>
#include <sstream>
#include <fstream>
#include <string>
#include <iostream>
#include <chrono>
#include <map>
#include <thread>
#include <list>
#include <stdio.h>
#include <array>
#include "interfaceToModel_dailyList.h"
#include "identifyIndex.h"
#include "conversionFromHHMMSStoContinuousTimeRepresentation.h"

using namespace std;

int interfaceToModel_dailyList()
{
    int const   stop_times_rows = 69863;
    int const   stop_times_cols = 8;
    bool        addEvent;
    std::vector<vector<string>> readDataAll;

    //day parameter
    int dayOfTheWeekToInvestigate = 0; // 0 for monday, 1 for tuesday, 2 for wednesday..

    for (int row_3 = 2; row_3 < stop_times_rows; row_3++)
    {
        //readData.clear();
        string readData[20][stop_times_cols + 6];
        //readDataAll[stop_times_rows][stop_times_cols];

        stringstream ss_pathToReadFile;
        ss_pathToReadFile << "Data folder/athens_metro/toFeedToModel/eventOfLine_" << row_3 << ".txt";
        string pathToReadFile = ss_pathToReadFile.str();
        std::ifstream file(pathToReadFile);

        if (file)
        {

            for (int row = 0; row < 20; row++)
            {
                std::string line;
                std::getline(file, line);

                std::stringstream iss(line);
                for (int col = 0; col < stop_times_cols + 6; col++)
                {
                    std::string val;
```

```
53.                 std::getline(iss, val, ',');
54.
55.                 std::stringstream convertor(val);
56.                 convertor >> readData[row][col];
57.
58.                 if (!iss.good())
59.                     break;
60.             }
61.
62.             if (!file.good())
63.                 break;
64.         }
65.
66.         /*cout << endl;
67.         for (int row = 0; row < 20; row++)
68.         {
69.             if ((readData[row][0] == "") && (readData[row][1] == ""))
70.             {
71.                 break;
72.             }
73.
74.             for (int col = 0; col < (stop_times_cols + 6); col++)
75.             {
76.                 cout << readData[row][col] << " ";
77.             }
78.             cout << endl;
79.         }
80.         cout << endl;*/
81.
82.         //an einai to mera X tote
83.         if (readData[0][dayOfTheWeekToInvestigate] == "1")
84.         {
85.             for (int row_1 = 1; row_1 < 20; row_1++)                //gia na diavaseis to arxiaki
86.             {
87.                 addEvent = true;
88.                 for (int row_2 = 0; row_2 < readDataAll.size(); row_2++) //gia na diavaseis ti lista me ta unique
    event
89.                 {
90.                     if (readData[row_1][0] == readDataAll[row_2][0])
91.                     {
92.                         if (readData[row_1][4] == readDataAll[row_2][4])
93.                         {
94.                             addEvent = false;
95.                         }
96.                     }
97.                 }
98.
99.                 if (row_1 == 1)
100.                {
101.                    //do nothin
102.                }
103.                else
104.                {
105.                    if (readData[row_1][7 + dayOfTheWeekToInvestigate] != "1")
106.                    {
107.                        addEvent = false;
108.                    }
```

```
109.                 }
110.
111.                 if (addEvent)
112.                 {
113.                     std::vector<string> event;
114.                     for (int column = 0; column < stop_times_cols; column++)
115.                     {
116.                         event.push_back(readData[row_1][column]);
117.                     }
118.                     readDataAll.push_back(event);
119.                 }
120.             }
121.         }
122.     }
123.
124.     //cout << readDataAll.size() << endl;
125.
126.     double div_result = div(row_3, 5000).rem;
127.     if (div_result == 0)
128.     {
129.         cout << "Line : " << row_3 << endl;
130.         cout << endl;
131.     }
132.
133. }
134.
135. /*for (int row_2 = 0; row_2 < readDataAll.size(); row_2++)
136. {
137. for (int column = 0; column < stop_times_cols; column++)
138. {
139. cout << readDataAll[row_2][column] << " ";
140. }
141. cout << endl;
142. }*/
143.
144. int counter;
145. for (int row_2 = 0; row_2 < readDataAll.size(); row_2++)
146. {
147.     for (int row_3 = 0; row_3 < readDataAll.size(); row_3++)
148.     {
149.         counter = 0;
150.         if (readDataAll[row_2][0] == readDataAll[row_3][0])
151.         {
152.             if (readDataAll[row_2][4] == readDataAll[row_3][4])
153.             {
154.                 counter += 1;
155.             }
156.         }
157.     }
158.     if (counter > 1)
159.     {
160.         cout << "Just a found an element twice in your list! It's not unique, then.";
161.     }
162. }
163.
164. cout << "FINAL: " << readDataAll.size();
165.
```

```
166.    stringstream ss_pathToWriteFile;
167.    ss_pathToWriteFile << "Data folder/athens_metro/toFeedToModel/ALL_EVENTS_FOR_ONE_DAY.txt";

168.    string pathToWriteFile = ss_pathToWriteFile.str();
169.    std::ofstream out(pathToWriteFile);
170.
171.    //write arriving event
172.    for (int row_2 = 0; row_2 < readDataAll.size(); row_2++)
173.    {
174.        for (int col = 0; col < stop_times_cols; col++)
175.        {
176.            out << readDataAll[row_2][col];
177.            if (col < stop_times_cols - 2)
178.            {
179.                out << ",";
180.            }
181.        }
182.        out << endl;
183.    }
184.    out.close();
185.
186.    //find the decision variables, write to file
187.
188.    //calculation of TTs1ss2rt
189.
190.    //calculation of APt and RFt, write to file
191.
192.    //declaration of constraints, write to file
193.
194.    return 1;
195. }
```

8.2.4 InterfaceToModel_decisionVars – based on PreOptimization file, decides the decision variables to be declared

```
#include <ilcplex/ilocplex.h>
#include <time.h>
#include <vector>
#include <sstream>
#include <fstream>
#include <string>
#include <iostream>
#include <chrono>
#include <map>
#include <thread>
#include <list>
#include <stdio.h>
#include <array>
#include "interfaceToModel_decisionVars.h"
#include "identifyIndex.h"
#include "conversionFromHHMMSStoContinuousTimeRepresentation.h"
#include "findRouteIDbyTripID.h"

using namespace std;

int interfaceToModel_decisionVars()
{
    int const stop_times_rows = 69863;
    int const stop_times_cols = 8;

    string readData[stop_times_rows][stop_times_cols];

    string stopIDOfEvent;
    string routeIDOfEvent;
    string tripIDOfEvent;

    int stopIndexOfEvent;
    int routeIndexOfEvent;
    int tripIndexOfEvent;
    int timeOfEventInContTime;

    std::vector<int>        varIndices;
    std::vector<vector<int>>    vars;

    stringstream ss_pathToReadFile;
    ss_pathToReadFile << "Data folder/athens_metro/toFeedToModel/ALL_EVENTS_FOR_ONE_DAY.txt";

    string pathToReadFile = ss_pathToReadFile.str();
    std::ifstream file(pathToReadFile);

    if (file)
    {
       for (int row = 0; row < stop_times_rows; ++row)
       {
          std::string line;
          std::getline(file, line);
          if (!file.good())
```

```
            break;

        std::stringstream iss(line);
        for (int col = 0; col < stop_times_cols; ++col)
        {
            std::string val;
            std::getline(iss, val, ',');

            std::stringstream convertor(val);
            convertor >> readData[row][col];

            if (!iss.good())
                break;
        }
        if (!file.good())
            break;
    }
}

for (int row = 0; row < stop_times_rows; ++row)
{
    if ((readData[row][0] == "") && (readData[row][1] == ""))
    {
        break;
    }

    cout << "Working on event of row: " << row << endl;

    /*for (int col = 0; col < stop_times_cols; ++col)
    {

        cout << readData[row][col] << " ";

    }*/
    tripIDOfEvent = readData[row][0];
    stopIDOfEvent = readData[row][3];

    tripIndexOfEvent = findTripIndex(tripIDOfEvent);
    stopIndexOfEvent = findStopIndex(stopIDOfEvent);

    string routeID = findRouteIDByTripID(tripIDOfEvent);
    int routeIndex = findRouteIndex(routeID);

    timeOfEventInContTime = conversionToContinuousTime(readData[row][1]);

    //cout << tripIndexOfEvent << " " << stopIndexOfEvent;

    //cout << endl;

    varIndices.clear();

    varIndices.push_back(stopIndexOfEvent);
    varIndices.push_back(routeIndex);
    varIndices.push_back(tripIndexOfEvent);
    varIndices.push_back(timeOfEventInContTime);

    vars.push_back(varIndices);
```

```
109.
110.    }
111.
112.    stringstream ss_pathToWriteFile;
113.    ss_pathToWriteFile << "Data folder/athens_metro/toFeedToModel/interfaceToModel/decisionVars.txt";

114.    string pathToWriteFile = ss_pathToWriteFile.str();
115.    std::ofstream out(pathToWriteFile);
116.
117.    ////write arriving event
118.    for (int i = 0; i < vars.size(); ++i)
119.    {
120.        out << vars[i][0] << "," << vars[i][1] << "," << vars[i][2] << "," << vars[i][3] << endl;
121.    }
122.    out.close();
123.
124.    return 1;
125.
126. }
```

8.2.5 interfaceToModel_objFun.cpp – File that decides the relations between variables in the objective function

```
#include <ilcplex/ilocplex.h>
#include <time.h>
#include <vector>
#include <sstream>
#include <fstream>
#include <utility>
#include <string>
#include <iostream>
#include <chrono>
#include <map>
#include <thread>
#include <list>
#include <stdio.h>
#include <array>
#include "interfaceToModel_objFun.h"
#include "identifyIndex.h"
#include "findRouteIDbyTripID.h"
#include "conversionFromHHMMSStoContinuousTimeRepresentation.h"

using namespace std;

int interfaceToModel_objFun()
{

    int const stop_times_rows = 69863;
    int const stop_times_cols = 7;
    bool addEvent;
    std::vector<vector<int>>    objFunEntries;
    std::vector<int>        singleEntry;

    int stopIndex_1;
    int stopIndex_2;
    string routeID_1;
    string routeID_2;
    int routeIndex_1;
    int routeIndex_2;
    int tripIndex_1;
    int tripIndex_2;

    int stopIndexOfEvent;
    int routeIndexOfEvent;
    int tripIndexOfEvent;

    //day parameter
    int dayOfTheWeekToInvestigate = 0; // 0 for monday, 1 for tuesday, 2 for wednesday..

    for (int row_3 = 2; row_3 < stop_times_rows; row_3++)
    {
        cout << "We are currently at: " << row_3 << endl;

        string readData[20][stop_times_cols+7];

```

```
53.         stringstream ss_pathToReadFile;
54.         ss_pathToReadFile << "Data folder/athens_metro/toFeedToModel/eventOfLine_" << row_3 << ".txt";
55.         string pathToReadFile = ss_pathToReadFile.str();
56.         std::ifstream file(pathToReadFile);
57.
58.         if (file)
59.         {
60.             for (int row = 0; row < 20; ++row)
61.             {
62.                 std::string line;
63.                 std::getline(file, line);
64.
65.                 std::stringstream iss(line);
66.
67.                 for (int col = 0; col < (stop_times_cols + 7); col++)
68.                 {
69.                     std::string val;
70.                     std::getline(iss, val, ',');
71.
72.                     std::stringstream convertor(val);
73.                     convertor >> readData[row][col];
74.
75.                     if (!iss.good())
76.                     break;
77.                 }
78.
79.                 if (!file.good())
80.                     break;
81.             }
82.
83.             //kane tora tin analisi
84.             if (readData[0][dayOfTheWeekToInvestigate] == "1")
85.             {
86.
87.                 stopIndex_1     = findStopIndex(readData[1][3]);
88.
89.                 routeID_1       = findRouteIDByTripID(readData[1][0]);
90.
91.                 routeIndex_1    = findRouteIndex(routeID_1);
92.
93.                 tripIndex_1     = findTripIndex(readData[1][0]);
94.
95.                 for (int row_2 = 2; row_2 < 20; row_2++)
96.                 {
97.                     if (readData[row_2][7 + dayOfTheWeekToInvestigate] == "1")
98.                     {
99.
100.                        stopIndex_2     = findStopIndex(readData[row_2][3]);
101.
102.                        routeID_2       = findRouteIDByTripID(readData[row_2][0]);
103.
104.                        routeIndex_2    = findRouteIndex(routeID_2);
105.
106.                        tripIndex_2     = findTripIndex(readData[row_2][0]);
107.
108.                        singleEntry.clear();
109.
```

```
                    singleEntry.push_back(stopIndex_2);
                    singleEntry.push_back(routeIndex_2);
                    singleEntry.push_back(tripIndex_2);

                    singleEntry.push_back(stopIndex_1);
                    singleEntry.push_back(routeIndex_1);
                    singleEntry.push_back(tripIndex_1);

                    objFunEntries.push_back(singleEntry);
                }
            }
        }
    }

    stringstream ss_pathToWriteFile;
    ss_pathToWriteFile << "Data folder/athens_metro/toFeedToModel/interfaceToModel/objFun.txt";
    string pathToWriteFile = ss_pathToWriteFile.str();
    std::ofstream out(pathToWriteFile);

    for (int i = 0; i < objFunEntries.size(); i++)
    {
        for (int j = 0; j < objFunEntries[i].size(); j++)
        {
            out << objFunEntries[i][j];
            if (j != (objFunEntries[i].size() - 1))
            {
                out << ",";
            }
        }
        out << endl;
    }
    out.close();

    return 1;
}
```

8.2.6 interfaceToModel_parameters.cpp – Decides on the values of the parameters of the model based the output of DailyList code

```
#include <ilcplex/ilocplex.h>
#include <time.h>
#include <vector>
#include <sstream>
#include <fstream>
#include <utility>
#include <string>
#include <iostream>
#include <chrono>
#include <map>
#include <thread>
#include <list>
#include <stdio.h>
#include <array>
#include "interfaceToModel_parameters.h"
#include "identifyIndex.h"
#include "conversionFromHHMMSStoContinuousTimeRepresentation.h"

using namespace std;

int interfaceToModel_parameters()
{
    int const stop_times_rows = 69863;
    int const stop_times_cols = 8;
    int const routes_rows = 4;
    int const routes_cols = 9;
    int const trips_rows = 3636;
    int const trips_cols = 7;

    string readData[stop_times_rows][stop_times_cols];

    string stopIDOfEvent;
    string routeIDOfEvent;
    string tripIDOfEvent;

    int stopIndexOfEvent;
    int routeIndexOfEvent;
    int tripIndexOfEvent;

    tuple<string, string, int>                                  tripCorrespIndex;
    vector<tuple<string, string, int>>                          trips;

    vector<tuple<string, vector<string>, vector<vector<string>>>>          routes;
    vector<tuple<string, vector<string>, vector<vector<vector<string>>>>>  routesWithEvents;
    vector<tuple<string, vector<string>, vector<vector<int>>>>             routesWithContTime;
    vector<string>                                              listOfTrips;
    vector<string>                                              listOfCalendars;

    std::vector<std::tuple<std::string, vector<vector<string>>>>           eventsToFindMeanPerRoute;
    std::vector<std::vector<string>>                            eventsToFindMean;

```

```
52.  std::vector<std::tuple<std::string, vector<int>>>                eventsToFindMeanPerRouteInIntegerMins;
53.
54.  std::vector<std::tuple<std::string, vector<string>, vector<vector<int>>>> timeDifferencesBetweenTrips;
55.
56.  std::vector<int>                                        varIndices;
57.  std::vector<vector<int>>                                    vars;
58.
59.  std::vector<std::tuple<std::string, vector<string>, vector<double>>>      RFrt;
60.
61.  bool addTrip;
62.  bool addRoute;
63.  string routeID;
64.
65.  stringstream ss_pathToReadFile;
66.  ss_pathToReadFile << "Data folder/athens_metro/toFeedToModel/ALL_EVENTS_FOR_ONE_DAY.txt";
67.  string pathToReadFile = ss_pathToReadFile.str();
68.  std::ifstream file(pathToReadFile);
69.
70.  if (file)
71.  {
72.     for (int row = 0; row < stop_times_rows; ++row)
73.     {
74.        std::string line;
75.        std::getline(file, line);
76.        if (!file.good())
77.           break;
78.
79.        std::stringstream iss(line);
80.        for (int col = 0; col < stop_times_cols; ++col)
81.        {
82.           std::string val;
83.           std::getline(iss, val, ',');
84.
85.           std::stringstream convertor(val);
86.           convertor >> readData[row][col];
87.
88.           if (!iss.good())
89.              break;
90.        }
91.        if (!file.good())
92.           break;
93.     }
94.  }
95.
96.  //start of reading stop_times.txt file
97.
98.  std::string stop_times_data[stop_times_rows][stop_times_cols];
99.  std::ifstream file4("Data folder/athens_metro/stop_times.txt");
100.
101. for (int row = 0; row < stop_times_rows; row++)
102. {
103.    std::string line;
104.    std::getline(file4, line);
105.
106.    std::stringstream iss(line);
```

```
107.
108.        for (int col = 0; col < stop_times_cols; col++)
109.        {
110.            std::string val;
111.            std::getline(iss, val, ',');
112.
113.            std::stringstream convertor(val);
114.            convertor >> stop_times_data[row][col];
115.
116.            if (!iss.good())
117.                break;
118.        }
119.
120.        if (!file4.good())
121.            break;
122.    }
123.
124.    //end of reading stop_times.txt file
125.    //start of reading routes.txt file
126.
127.    string routes_data[routes_rows][routes_cols];
128.    std::ifstream file1("Data folder/athens_metro/routes.txt");
129.
130.    for (int row = 0; row < routes_rows; ++row)
131.    {
132.        std::string line;
133.        std::getline(file1, line);
134.
135.        std::stringstream iss(line);
136.
137.        for (int col = 0; col < routes_cols; ++col)
138.        {
139.            std::string val;
140.            std::getline(iss, val, ',');
141.
142.            std::stringstream convertor(val);
143.            convertor >> routes_data[row][col];
144.
145.            if (!iss.good())
146.                break;
147.        }
148.        if (!file1.good())
149.            break;
150.    }
151.
152.    //end of reading routes.txt
153.    //start of reading trips.txt
154.
155.    string trips_data[trips_rows][trips_cols];
156.    std::ifstream file2("Data folder/athens_metro/trips.txt");
157.
158.    for (int row = 0; row < trips_rows; ++row)
159.    {
160.        std::string line;
161.        std::getline(file2, line);
162.
163.        std::stringstream iss(line);
```

```
164.
165.        for (int col = 0; col < trips_cols; ++col)
166.        {
167.            std::string val;
168.            std::getline(iss, val, ',');
169.            if (!iss.good())
170.                break;
171.
172.            std::stringstream convertor(val);
173.            convertor >> trips_data[row][col];
174.        }
175.        if (!file2.good())
176.            break;
177.    }
178.
179.    //end of reading trips.txt
180.
181.    //exoume diavasei ola ta arxeia kai ksekinaei o elegxos
182.
183.    for (int row =3500; row < stop_times_rows; row++)                    //edo o elegxos pou ginetai afora kirios to
       na eimaste kalimenoi os pros ta rows para oti exoume na knaoume me to stop_times_rows
184.    {
185.        if ((readData[row][0] == "") && (readData[row][1] == ""))
186.        {
187.            break;
188.        }
189.
190.        //vres to trip tou event
191.        tripIDOfEvent = readData[row][0];
192.        /*tripIndexOfEvent = findTripIndex(tripIDOfEvent);*/
193.
194.        //vres to route tou event
195.        for (int j = 0; j < trips_rows; j++)
196.        {
197.            if (trips_data[j][2] == tripIDOfEvent)
198.            {
199.                routeID = trips_data[j][0];
200.                break;
201.            }
202.        }
203.
204.        //vres to service
205.        string calendarID;
206.        for (int j = 0; j < trips_rows; j++)
207.        {
208.            if (tripIDOfEvent == trips_data[j][2])
209.            {
210.                calendarID = trips_data[j][1];
211.                break;
212.            }
213.        }
214.
215.        //list of unique routes, frontise i lista na einai unique
216.        addRoute = true;
217.
218.        for (int j = 0; j < routes.size(); j++)
219.        {
```

```
220.         if (get<0>(routes[j]) == routeID)
221.         {
222.           addRoute = false;
223.           break;
224.         }
225.       }
226.
227.       if (addRoute) // an den iparxei valto
228.       {
229.         routes.push_back(std::make_tuple(routeID, vector<string>(), vector<vector<string>>()));
230.       }
231.
232.       //vres to index pou exeis kataxorisei telika to route
233.       int indexOfCurrentRouteinRoutes = 0;
234.       for (int j = 0; j < routes.size(); j++)
235.       {
236.         if (get<0>(routes[j]) == routeID)
237.         {
238.           indexOfCurrentRouteinRoutes = j;
239.           break;
240.         }
241.       }
242.
243.       addTrip = true;
244.       //vale unique trip
245.       for (int j = 0; j < get<1>(routes[indexOfCurrentRouteinRoutes]).size(); j++)
246.       {
247.         if (get<1>(routes[indexOfCurrentRouteinRoutes])[j] == tripIDOfEvent)
248.         {
249.           addTrip = false;
250.           break;
251.         }
252.       }
253.
254.       if (addTrip)
255.       {
256.         get<1>(routes[indexOfCurrentRouteinRoutes]).push_back(tripIDOfEvent);
257.         get<2>(routes[indexOfCurrentRouteinRoutes]).push_back(vector<string>());
258.
259.         int indexOfCurrentTripinList = get<2>(routes[indexOfCurrentRouteinRoutes]).size() - 1;
260.
261.         for (int j = 0; j < trips_rows; j++)
262.         {
263.           if ((calendarID == trips_data[j][1]) && (routeID == trips_data[j][0]))
264.           {
265.             get<2>(routes[indexOfCurrentRouteinRoutes])[indexOfCurrentTripinList].push_back(trips_data
    [j][2]);
266.           }
267.         }
268.       }
269.     }
270.
271.     //calculation of RFr
272.     for (int i = 0; i < routes.size(); i++)
273.     {
274.       for (int j = 0; j < get<1>(routes[i]).size(); j++)
275.       {
```

```
276.         std::sort(get<2>(routes[i])[j].begin(), get<2>(routes[i])[j].end());
277.     }
278. }
279.
280. //make replica of routes but with events with stop sequence 1
281. for (int i = 0; i < routes.size(); i++)
282. {
283.     routesWithEvents.push_back(std::make_tuple(get<0>(routes[i]), vector<string>(), vector<vector<vector
<string>>>()));
284.     for (int j = 0; j < get<1>(routes[i]).size(); j++)
285.     {
286.         get<1>(routesWithEvents[i]).push_back(get<1>(routes[i])[j]);
287.         get<2>(routesWithEvents[i]).push_back(vector<vector<string>>());
288.         for (int k = 0; k < get<2>(routes[i])[j].size(); k++)
289.         {
290.             for (int l = 0; l < stop_times_rows; l++)
291.             {
292.                 if (stop_times_data[l][4] == "1")
293.                 {
294.                     if (stop_times_data[l][0] == get<2>(routes[i])[j][k])
295.                     {
296.                         std::vector<string> event;
297.                         for (int column = 0; column < stop_times_cols; column++)
298.                         {
299.                             event.push_back(stop_times_data[l][column]);
300.                         }
301.                         get<2>(routesWithEvents[i])[j].push_back(event);
302.                         break;
303.                     }
304.                 }
305.             }
306.         }
307.     }
308. }
309.
310. /*for (int i = 0; i < routes.size(); i++)
311. {
312.     for (int j = 0; j < get<1>(routes[i]).size(); j++)
313.     {
314.         for (int k = 0; k < get<2>(routes[i])[j].size(); k++)
315.         {
316.             cout << get<0>(routesWithEvents[i]) << " " << get<1>(routesWithEvents[i])[j] << " " << get<2>(rout
esWithEvents[i])[j][k][0] << endl;
317.         }
318.     }
319.     cout << endl;
320. }*/
321.
322. for (int i = 0; i < routesWithEvents.size(); i++)
323. {
324.     routesWithContTime.push_back(std::make_tuple(get<0>(routesWithEvents[i]), vector<string>(), vector
<vector<int>>()));
325.     for (int j = 0; j < get<1>(routesWithEvents[i]).size(); j++)
326.     {
327.         get<1>(routesWithContTime[i]).push_back(get<1>(routesWithEvents[i])[j]);
328.         get<2>(routesWithContTime[i]).push_back(vector<int>());
329.         for (int k = 0; k < get<2>(routesWithEvents[i])[j].size(); k++)
```

```
330.          {
331.             get<2>(routesWithContTime[i])[j].push_back(conversionToContinuousTime(get<2>(routesWithEv
      ents[i])[j][k][1]));
332.          }
333.       }
334.    }
335.
336.    for (int i = 0; i < routesWithContTime.size(); i++)
337.    {
338.       timeDifferencesBetweenTrips.push_back(std::make_tuple(get<0>(routesWithContTime[i]), vector<strin
      g>(), vector<vector<int>>()));
339.       for (int j = 0; j < get<1>(routesWithContTime[i]).size(); j++)
340.       {
341.          get<1>(timeDifferencesBetweenTrips[i]).push_back(get<1>(routesWithContTime[i])[j]);
342.          get<2>(timeDifferencesBetweenTrips[i]).push_back(vector<int>());
343.          get<2>(timeDifferencesBetweenTrips[i])[j].push_back(0);
344.          for (int k = 1; k < get<2>(routesWithContTime[i])[j].size(); k++)
345.          {
346.             if (((get<2>(routesWithContTime[i])[j][k] - get<2>(routesWithContTime[i])[j][k -
      1])<0) || ((get<2>(routesWithContTime[i])[j][k] - get<2>(routesWithContTime[i])[j][k - 1])>1000))
347.             {
348.                get<2>(timeDifferencesBetweenTrips[i])[j].push_back(0);
349.             }
350.             else
351.             {
352.                get<2>(timeDifferencesBetweenTrips[i])[j].push_back(get<2>(routesWithContTime[i])[j][k] -
      get<2>(routesWithContTime[i])[j][k - 1]);
353.             }
354.          }
355.       }
356.    }
357.
358.    for (int i = 0; i < timeDifferencesBetweenTrips.size(); i++)
359.    {
360.       RFrt.push_back(std::make_tuple(get<0>(routes[i]), vector<string>(), vector<double>()));
361.       for (int j = 0; j < get<1>(timeDifferencesBetweenTrips[i]).size(); j++)
362.       {
363.          get<1>(RFrt[i]).push_back(get<1>(timeDifferencesBetweenTrips[i])[j]);
364.          double  sum           = 0;
365.          int     counterOfZeros  = 0;
366.          for (int k = 0; k < get<2>(timeDifferencesBetweenTrips[i])[j].size(); k++)
367.          {
368.             sum += get<2>(timeDifferencesBetweenTrips[i])[j][k];
369.
370.             if (get<2>(timeDifferencesBetweenTrips[i])[j][k] == 0)
371.             {
372.                counterOfZeros += 1;
373.             }
374.          }
375.          double mean = sum / (get<2>(timeDifferencesBetweenTrips[i])[j].size() - counterOfZeros);
376.          get<2>(RFrt[i]).push_back(mean);
377.       }
378.    }
379.
380.    stringstream ss_pathToWriteFile;
381.    ss_pathToWriteFile << "Data folder/athens_metro/toFeedToModel/interfaceToModel/parameters.txt";
382.    string pathToWriteFile = ss_pathToWriteFile.str();
```

```
383.    std::ofstream out(pathToWriteFile);
384.
385.    for (int i = 0; i < RFrt.size(); i++)
386.    {
387.        for (int j = 0; j < get<1>(RFrt[i]).size(); j++)
388.        {
389.            int routeIndex = findRouteIndex(get<0>(RFrt[i]));
390.            int tripIndex = findTripIndex(get<1>(RFrt[i])[j]);
391.            out << routeIndex << "," << tripIndex << "," << get<2>(RFrt[i])[j] << endl;
392.        }
393.    }
394.
395.    /*for (int i = 0; i < routesWithContTime.size(); i++)
396.    {
397.        for (int j = 0; j < get<1>(routesWithContTime[i]).size(); j++)
398.        {
399.            for (int k = 0; k < get<2>(routesWithContTime[i])[j].size(); k++)
400.            {
401.                cout << get<0>(routesWithContTime[i]) << " < " << get<1>(routesWithContTime[i])[j] << " < " << g
    et<2>(routes[i])[j][k] << " < " << get<2>(routesWithContTime[i])[j][k] << " < " << get<2>(routesWithContTi
    me[i])[j][k - 1] << " < " << get<2>(timeDifferencesBetweenTrips[i])[j][k] << endl;
402.            }
403.            cout << endl;
404.        }
405.        cout << endl;
406.    }*/
407.
408.    /*stringstream ss_pathToWriteFile;
409.    ss_pathToWriteFile << "Data folder/athens_metro/toFeedToModel/interfaceToModel/parameters.txt";
410.    string pathToWriteFile = ss_pathToWriteFile.str();
411.    std::ofstream out(pathToWriteFile);
412.
413.    for (int i = 0; i < routesWithEvents.size(); i++)
414.    {
415.        for (int j = 0; j < get<1>(routesWithEvents[i]).size(); j++)
416.        {
417.            for (int k = 0; k < get<2>(routesWithEvents[i])[j].size(); k++)
418.            {
419.                out << get<0>(routesWithEvents[i]) << " < " << get<1>(routesWithEvents[i])[j] << " < " << get<2>(r
    outesWithEvents[i])[j][k][0] << endl;
420.            }
421.            out << endl;
422.        }
423.        out << endl;
424.    }*/
425.
426.    /*stringstream ss_pathToWriteFile;
427.    ss_pathToWriteFile << "Data folder/athens_metro/toFeedToModel/decisionVars.txt";
428.    string pathToWriteFile = ss_pathToWriteFile.str();
429.    std::ofstream out(pathToWriteFile);*/
430.
431.    ////write arriving event
432.    /*for (int i = 0; i < routes.size(); i++)
433.    {
434.        for (int j = 0; j < get<1>(eventsToFindMeanPerRouteInIntegerMins[i]).size(); j++)
435.        {
```

```
436.            cout << get<0>(eventsToFindMeanPerRouteInIntegerMins[i]) << " " << get<1>(eventsToFindMeanPe
    rRouteInIntegerMins[i])[j] << " ";
437.            if (j>0)
438.            {
439.               cout << get<1>(timeDifferencesBetweenTrips[i])[j] << endl;
440.            }
441.         }
442.         cout << endl;
443.      }*/
444.      //out.close();
445.
446.      /*for (int i = 0; i < routes.size(); i++)
447.      {
448.         cout << RFr[i] << endl;
449.      }*/
450.
451.      return 1;
452.
453. }
```

8.2.7 interfaceToModel_transitionTimes.cpp – used to calculate transition time parameters of the model

```
#include <ilcplex/ilocplex.h>
#include <time.h>
#include <vector>
#include <sstream>
#include <fstream>
#include <utility>
#include <string>
#include <iostream>
#include <chrono>
#include <map>
#include <thread>
#include <list>
#include <stdio.h>
#include <array>
#include "interfaceToModel_transitionTimes.h"
#include "identifyIndex.h"
#include "findRouteIDbyTripID.h"
#include "conversionFromHHMMSStoContinuousTimeRepresentation.h"

using namespace std;

int interfaceToModel_transitionTimes()
{
    int const stop_times_rows = 69863;
    int const stop_times_cols = 8;
    int const routes_rows = 4;
    int const routes_cols = 9;
    int const trips_rows = 3636;
    int const trips_cols = 7;

    string readData[stop_times_rows][stop_times_cols];

    string stopIDOfEvent;
    string routeIDOfEvent;
    string tripIDOfEvent;

    string tripID_1;
    string tripID_2;

    int time_1;
    int time_2;
    int TT;

    std::vector<string> event;
    vector<vector<string>> listOfEvents;

    bool addEvent;

    std::vector<std::tuple<std::string, vector<vector<string>>>>    tripAssociations;
    std::vector<std::tuple<std::string, vector<int>>>           TTs1s2rt;
    vector<int>                                         listOfTTs;

```

```
53.     int stopIndex_1;
54.     int stopIndex_2;
55.     int routeIndex;
56.     int tripIndex;
57.
58.     int stopIndexOfEvent;
59.     int routeIndexOfEvent;
60.     int tripIndexOfEvent;
61.
62.     stringstream ss_pathToReadFile;
63.     ss_pathToReadFile << "Data folder/athens_metro/toFeedToModel/ALL_EVENTS_FOR_ONE_DAY.txt";

64.     string pathToReadFile = ss_pathToReadFile.str();
65.     std::ifstream file(pathToReadFile);
66.
67.     if (file)
68.     {
69.        for (int row = 0; row < stop_times_rows; ++row)
70.        {
71.           std::string line;
72.           std::getline(file, line);
73.
74.           std::stringstream iss(line);
75.           for (int col = 0; col < stop_times_cols; ++col)
76.           {
77.              std::string val;
78.              std::getline(iss, val, ',');
79.
80.              std::stringstream convertor(val);
81.              convertor >> readData[row][col];
82.
83.              if (!iss.good())
84.                 break;
85.           }
86.           if (!file.good())
87.              break;
88.        }
89.     }
90.
91.     for (int row = 0; row < stop_times_rows; ++row)
92.     {
93.        if ((readData[row][0] == "") && (readData[row][1] == ""))
94.        {
95.           break;
96.        }
97.
98.        tripID_1 = readData[row][0];
99.        addEvent = true;
100.
101.       for (int i = 0; i < tripAssociations.size(); i++)
102.       {
103.          if (get<0>(tripAssociations[i]) == tripID_1)
104.          {
105.             addEvent = false;
106.          }
107.       }
108.
```

```
109.      listOfEvents.clear();
110.      if (addEvent)
111.      {
112.         for (int row_2 = 0; row_2 < stop_times_rows; row_2++)
113.         {
114.            if ((readData[row_2][0] == "") && (readData[row_2][1] == ""))
115.            {
116.               break;
117.            }
118.
119.            tripID_2 = readData[row_2][0];
120.
121.            if (tripID_1 == tripID_2)
122.            {
123.               event.clear();
124.               for (int column = 0; column < stop_times_cols; column++)
125.               {
126.                  event.push_back(readData[row_2][column]);
127.               }
128.               listOfEvents.push_back(event);
129.            }
130.
131.         }
132.         tripAssociations.push_back(std::make_tuple(tripID_1, listOfEvents));
133.      }
134.
135.   }
136.
137.   for (int i = 0; i < tripAssociations.size(); i++)
138.   {
139.      std::sort(get<1>(tripAssociations[i]).begin(), get<1>(tripAssociations[i]).end(), [](const vector<string> &
      a, const vector<string> & b){ return stoi(a[4]) < stoi(b[4]); });
140.   }
141.
142.   for (int i = 0; i < tripAssociations.size(); i++)
143.   {
144.      for (int j = 0; j < get<1>(tripAssociations[i]).size(); j++)
145.      {
146.         cout << get<0>(tripAssociations[i]) << " ";
147.         for (int k = 0; k < stop_times_cols; k++)
148.         {
149.            cout << get<1>(tripAssociations[i])[j][k] << " ";
150.         }
151.         cout << endl;
152.      }
153.      cout << endl;
154.   }
155.
156.   //taksinomisi os pros trip id kai os pros sequence
157.   for (int i = 0; i < tripAssociations.size(); i++)
158.   {
159.      for (int j = 0; j < get<1>(tripAssociations[i]).size(); j++)
160.      {
161.         for (int k = 0; k < stop_times_cols; k++)
162.         {
163.            cout << get<1>(tripAssociations[i])[j][k] << " ";
164.         }
```

```
165.            cout << endl;
166.        }
167.    }
168.
169.    //upologismos ton TT
170.    for (int i = 0; i < tripAssociations.size(); i++)
171.    {
172.        listOfTTs.clear();
173.        listOfTTs.push_back(0);
174.        for (int j = 1; j < get<1>(tripAssociations[i]).size(); j++)
175.        {
176.
177.            time_1 = conversionToContinuousTime(get<1>(tripAssociations[i])[j-1][1]);
178.            time_2 = conversionToContinuousTime(get<1>(tripAssociations[i])[j][1]);
179.
180.            TT = time_2 - time_1;
181.
182.            listOfTTs.push_back(TT);
183.        }
184.        TTs1s2rt.push_back(std::make_tuple(get<0>(tripAssociations[i]), listOfTTs));
185.    }
186.
187.    //write to file
188.    stringstream ss_pathToWriteFile;
189.    ss_pathToWriteFile << "Data folder/athens_metro/toFeedToModel/interfaceToModel/TTs1s2rt.txt";
190.    string pathToWriteFile = ss_pathToWriteFile.str();
191.    std::ofstream out(pathToWriteFile);
192.
193.    //write arriving event
194.    for (int i = 0; i < TTs1s2rt.size(); i++)
195.    {
196.        cout << "Examining transition: " << i << endl;
197.        for (int j = 1; j < get<1>(TTs1s2rt[i]).size(); j++)
198.        {
199.
200.            stopIndex_1 = findStopIndex(get<1>(tripAssociations[i])[j - 1][3]);
201.
202.            stopIndex_2 = findStopIndex(get<1>(tripAssociations[i])[j][3]);
203.
204.            string routeID = findRouteIDByTripID(get<1>(tripAssociations[i])[j][0]);
205.
206.            routeIndex = findRouteIndex(routeID);
207.
208.            tripIndex = findTripIndex(get<1>(tripAssociations[i])[j][0]);
209.
210.            out << stopIndex_1 << "," << stopIndex_2 << "," << routeIndex << "," << tripIndex << "," << get<1>(T
    Ts1s2rt[i])[j] << endl;
211.            //cout << stopIndex_1 << "," << stopIndex_2 << "," << routeIndex << "," << tripIndex << "," << get<1
    >(TTs1s2rt[i])[j] << endl;
212.            //cout << endl;
213.        }
214.    }
215.    out.close();
216.
217.
218.    /*for (int i = 0; i < tripAssociations.size(); i++)
219.    {
```

```
220.        for (int j = 0; j < get<1>(tripAssociations[i]).size(); j++)
221.        {
222.            cout << get<0>(tripAssociations[i]) << " ";
223.            for (int column = 0; column < stop_times_cols; column++)
224.            {
225.                cout << get<1>(tripAssociations[i])[j][column] << " ";
226.            }
227.            cout << get<1>(TTs1s2rt[i])[j] << " ";
228.            cout << endl;
229.        }
230.        cout << endl;
231.    }*/
232.
233.    return 1;
234.
235. }
```

8.3 theriticalOneAndTwoStationApproaches – The files for the two theoretical cases

8.3.1 oneStationProblem.cpp – Self-explanatory title

```
#include <ilcplex/ilocplex.h>
#include <time.h>
#include <vector>
#include <sstream>
#include <fstream>
#include <string>
#include <iostream>
#include <chrono>
#include <map>
#include <thread>
#include <list>
#include <stdio.h>
#include "OneStationProblem.h"

//Data structures used for CPLEX declaration(CPLEX Construct Matrices)
//(used in order to declare later in code with fewer lines of code)
typedef IloArray<IloNumArray>        IloNumMatrix2x2;
typedef IloArray<IloNumMatrix2x2>    IloNumMatrix3x3;
typedef IloArray<IloNumMatrix3x3>    IloNumMatrix4x4;

typedef IloArray<IloNumVarArray>     IloNumVarMatrix2x2;
typedef IloArray<IloNumVarMatrix2x2> IloNumVarMatrix3x3;
typedef IloArray<IloNumVarMatrix3x3> IloNumVarMatrix4x4;

typedef IloArray<IloRangeArray>      IloRangeMatrix2x2;
typedef IloArray<IloRangeMatrix2x2>  IloRangeMatrix3x3;
typedef IloArray<IloRangeMatrix3x3>  IloRangeMatrix4x4;

using namespace std;

int oneStationProblem_approach_1()
{

    //CPLEX parameters
    IloEnv env;
    char label[70];
    IloModel model(env);
    IloCplex Cplex(env);

    //data of the problem
    std::vector<int>        vector_arrivalAndDepartureTimes;        //data for the problem/model
    std::vector<vector<int>>   arrivalAndDepartureTimes;            //data for the problem/model
    std::vector<double> FRi;                                        //Frequency of Route i
    std::vector<double> APi;                                        //Allowed Percentage to change for route i

    vector_arrivalAndDepartureTimes.push_back(166);                 //ATA11, DTA11
    vector_arrivalAndDepartureTimes.push_back(290);                 //ATA12, DTA12
    vector_arrivalAndDepartureTimes.push_back(313);                 //ATA13, DTA13
```

```
49.     vector_arrivalAndDepartureTimes.push_back(456);                 //ATA14, DTA14
50.     arrivalAndDepartureTimes.push_back(vector_arrivalAndDepartureTimes);
51.     FRi.push_back(306.25);                                  //frequency for first route
52.     APi.push_back(0.1);
53.
54.     vector_arrivalAndDepartureTimes.clear();
55.     vector_arrivalAndDepartureTimes.push_back(312);                 //ATA21, DTA21
56.     vector_arrivalAndDepartureTimes.push_back(313);                 //ATA22, DTA22
57.     vector_arrivalAndDepartureTimes.push_back(314);                 //ATA23, DTA23
58.     arrivalAndDepartureTimes.push_back(vector_arrivalAndDepartureTimes);
59.     FRi.push_back(1);                                   //frequency for second route
60.     APi.push_back(0.1);
61.
62.     vector_arrivalAndDepartureTimes.clear();
63.     vector_arrivalAndDepartureTimes.push_back(290);                 //ATA31, DTA31
64.     vector_arrivalAndDepartureTimes.push_back(516);                 //ATA32, DTA32
65.     vector_arrivalAndDepartureTimes.push_back(742);                 //ATA33, DTA33
66.     arrivalAndDepartureTimes.push_back(vector_arrivalAndDepartureTimes);
67.     FRi.push_back(226);                                 //frequency for third route
68.     APi.push_back(0.1);
69.
70.     //declaration of the decision variables
71.
72.     IloNumVarMatrix2x2 VAR_ATij(env, 0);
73.     IloNumVarMatrix2x2 VAR_DTij(env, 0);
74.
75.     //declaration of the arrival times variables
76.
77.     for (int i = 0; i < 3; i++)
78.     {
79.       IloNumVarArray VAR_ATi(env, 0);
80.       for (int j = 0; j < 4; j++)
81.       {
82.         if (!((i == 1) && (j == 3)) && !((i == 2) && (j == 3)))
83.         {
84.           sprintf_s(label, "AT(%d)(%d)", i, j);
85.           IloNumVar VAR_AT(env, 0, 2880, ILOINT, label);
86.           VAR_ATi.add(VAR_AT);
87.         }
88.       }
89.       VAR_ATij.add(VAR_ATi);
90.     }
91.
92.     //declaration of the departure times variables
93.
94.     for (int i = 0; i < 3; i++)
95.     {
96.       IloNumVarArray VAR_DTi(env, 0);
97.       for (int j = 0; j < 4; j++)
98.       {
99.         if (!((i == 1) && (j == 3)) && !((i == 2) && (j == 3)))
100.        {
101.          sprintf_s(label, "DT(%d)(%d)", i, j);
102.          IloNumVar VAR_DT(env, 0, 2880, ILOINT, label);
103.          VAR_DTi.add(VAR_DT);
104.        }
105.      }
```

```
106.     VAR_DTij.add(VAR_DTi);
107.   }
108.
109.   //Constraints for the model
110.
111.   //AT == DT
112.   for (int i = 0; i < 3; i++)
113.   {
114.     for (int j = 0; j < 4; j++)
115.     {
116.       if (!((i == 1) && (j == 3)) && !((i == 2) && (j == 3)))
117.       {
118.         IloExpr expr(env, 0);
119.         expr = VAR_ATij[i][j] - VAR_DTij[i][j];
120.         sprintf_s(label, "");
121.         double Con_LB = 0;
122.         double Con_UB = 0;
123.         IloRange Constraint_1(env, Con_LB, expr, Con_UB, label);
124.         model.add(Constraint_1);
125.         expr.end();
126.       }
127.     }
128.   }
129.
130.   for (int i = 0; i < 3; i++)
131.   {
132.     for (int j = 0; j < 4; j++)
133.     {
134.       if (!((i == 1) && (j == 3)) && !((i == 2) && (j == 3)))
135.       {
136.         IloExpr expr1(env, 0);
137.         expr1 = VAR_ATij[i][j];
138.         double Con_LB = arrivalAndDepartureTimes[i][j] - (APi[i] * FRi[i]);
139.         double Con_UB = arrivalAndDepartureTimes[i][j] + (APi[i] * FRi[i]);
140.         IloRange Constraint_1(env, Con_LB, expr1, Con_UB, label);
141.         model.add(Constraint_1);
142.         expr1.end();
143.
144.         IloExpr expr2(env, 0);
145.         expr2 = VAR_DTij[i][j];
146.         Con_LB = arrivalAndDepartureTimes[i][j] - (APi[i] * FRi[i]);
147.         Con_UB = arrivalAndDepartureTimes[i][j] + (APi[i] * FRi[i]);
148.         IloRange Constraint_2(env, Con_LB, expr2, Con_UB, label);
149.         model.add(Constraint_2);
150.         expr2.end();
151.       }
152.     }
153.   }
154.
155.   IloExpr expr(env, 0);
156.   expr += (VAR_DTij[1][0] - VAR_ATij[0][0]) + //1
157.         (VAR_DTij[2][0] - VAR_ATij[0][0]) + //2
158.         (VAR_DTij[1][0] - VAR_ATij[0][1]) + //3
159.         (VAR_DTij[2][1] - VAR_ATij[0][1]) + //4
160.         (VAR_DTij[1][2] - VAR_ATij[0][2]) + //5
161.         (VAR_DTij[2][1] - VAR_ATij[0][2]) + //6
162.         (VAR_DTij[2][2] - VAR_ATij[0][3]) + //7
```

```
        (VAR_DTij[0][2] - VAR_ATij[1][0]) + //8
        (VAR_DTij[2][1] - VAR_ATij[1][0]) + //9
        (VAR_DTij[0][3] - VAR_ATij[1][1]) + //10
        (VAR_DTij[2][1] - VAR_ATij[1][1]) + //11
        (VAR_DTij[0][3] - VAR_ATij[1][2]) + //12
        (VAR_DTij[2][1] - VAR_ATij[1][2]) + //13
        (VAR_DTij[0][2] - VAR_ATij[2][0]) + //14
        (VAR_DTij[1][0] - VAR_ATij[2][0]) ; //15

  model.add(IloMinimize(env, expr));
  expr.end();

  Cplex.extract(model);
  Cplex.exportModel("oneStationProblem_approach_1.lp");

  Cplex.solve();

  cout << "\n";

  cout << "Waiting time before optimization was: " << 1948 << "\n";
  cout << "Waiting time after optimization is (Value of the objective function): " << Cplex.getObjValue();

  cout << "\n";

  for (int i = 0; i < 3; i++)
  {
    for (int j = 0; j < 4; j++)
    {
      if (!((i == 1) && (j == 3)) && !((i == 2) && (j == 3)))
      {
        cout << "\n";

        cout << "The Arrival time to the node with " << i+1 << " " << j+1 << " is " << Cplex.getValue(VAR_
ATij[i][j]);

        cout << "\n";
      }
    }
  }

  for (int i = 0; i < 3; i++)
  {
    for (int j = 0; j < 4; j++)
    {
      if (!((i == 1) && (j == 3)) && !((i == 2) && (j == 3)))
      {
        cout << "\n";

        cout << "The Departure time to the node with " << i + 1 << " " << j + 1 << " is " << Cplex.getValue(V
AR_DTij[i][j]);

        cout << "\n";
      }
    }
  }

  return 1;
```

```
}

//approach with the network events
int oneStationProblem_approach_2()
{

  //CPLEX parameters
  IloEnv env;
  char label[70];
  IloModel model(env);
  IloCplex Cplex(env);

  //data of the problem
  std::vector<int>        arrivalAndDepartureTimes;   //data for the problem/model
  std::vector<double> FRi;                            //Frequency of Route i
  std::vector<double> APi;                            //Allowed Percentage to change for route i

  arrivalAndDepartureTimes.push_back(166);
  arrivalAndDepartureTimes.push_back(290);
  arrivalAndDepartureTimes.push_back(313);
  arrivalAndDepartureTimes.push_back(456);

  arrivalAndDepartureTimes.push_back(312);
  arrivalAndDepartureTimes.push_back(313);
  arrivalAndDepartureTimes.push_back(314);

  arrivalAndDepartureTimes.push_back(290);
  arrivalAndDepartureTimes.push_back(516);
  arrivalAndDepartureTimes.push_back(742);

  FRi.push_back(306.25);                //frequency for first route
  APi.push_back(0.1);                   //percentage allowed to change

  FRi.push_back(1);                     //frequency for second route
  APi.push_back(0.1);                   //percentage allowed to change

  FRi.push_back(226);                   //frequency for third route
  APi.push_back(0.1);                   //percentage allowed to change

  //declaration of the decision variables

  IloNumVarArray VAR_NEi(env, 0);

  //declaration of the arrival times variables

  for (int i = 0; i < 10; i++)
  {
    sprintf_s(label, "ÍÅ(%d)", i);
    IloNumVar VAR_NE(env, 0, 2880, ILOINT, label);
    VAR_NEi.add(VAR_NE);
  }

  for (int i = 0; i < 10; i++)
  {
    int j;
    if (i<=3)
    {
```

```
            j = 0;
        }
        else if (i<=6)
        {
            j = 1;
        }
        else
        {
            j = 2;
        }
        IloExpr expr(env, 0);
        expr = VAR_NEi[i];
        double Con_LB = arrivalAndDepartureTimes[i] - (APi[j] * FRi[j]);
        double Con_UB = arrivalAndDepartureTimes[i] + (APi[j] * FRi[j]);
        IloRange Constraint_1(env, Con_LB, expr, Con_UB, label);
        model.add(Constraint_1);
        expr.end();

    }

    IloExpr expr(env, 0);
    expr += (VAR_NEi[4] - VAR_NEi[0]) + //1
            (VAR_NEi[7] - VAR_NEi[0]) + //2
            (VAR_NEi[4] - VAR_NEi[1]) + //3
            (VAR_NEi[8] - VAR_NEi[1]) + //4
            (VAR_NEi[6] - VAR_NEi[2]) + //5
            (VAR_NEi[8] - VAR_NEi[2]) + //6
            (VAR_NEi[9] - VAR_NEi[3]) + //7
            (VAR_NEi[2] - VAR_NEi[4]) + //8
            (VAR_NEi[8] - VAR_NEi[4]) + //9
            (VAR_NEi[3] - VAR_NEi[5]) + //10
            (VAR_NEi[8] - VAR_NEi[5]) + //11
            (VAR_NEi[3] - VAR_NEi[6]) + //12
            (VAR_NEi[8] - VAR_NEi[6]) + //13
            (VAR_NEi[2] - VAR_NEi[7]) + //14
            (VAR_NEi[4] - VAR_NEi[7]) ; //15

    model.add(IloMinimize(env, expr));
    expr.end();

    Cplex.extract(model);
    Cplex.exportModel("oneStationProblem_approach_2.lp");

    Cplex.solve();

    cout << "\n";

    cout << "Waiting time before optimization was: " << 1948 << "\n";
    cout << "Waiting time after optimization is (Value of the objective function): " << Cplex.getObjValue();

    cout << "\n";

    for (int i = 0; i < 10; i++)
    {
        cout << "\n";

        cout << "The Network Event with index " << i + 1 << " is " << Cplex.getValue(VAR_NEi[i]);
```

```
332.
333.        cout << "\n";
334.    }
335.
336.    return 1;
337. }
338.
339. //approach with the network events
340. int oneStationProblem_approach_3()
341. {
342.
343.    //CPLEX parameters
344.    IloEnv env;
345.    char label[70];
346.    IloModel model(env);
347.    IloCplex Cplex(env);
348.
349.    //data of the problem
350.    std::vector<int>        vector_arrivalAndDepartureTimes;          //data for the problem/model
351.    std::vector<vector<int>>    arrivalAndDepartureTimes;              //data for the problem/model
352.    std::vector<double> FRi;                                  //Frequency of Route i
353.    std::vector<double> APi;                                  //Allowed Percentage to change for route i
354.
355.    vector_arrivalAndDepartureTimes.push_back(166);                    //ATA11, DTA11
356.    vector_arrivalAndDepartureTimes.push_back(290);                    //ATA12, DTA12
357.    vector_arrivalAndDepartureTimes.push_back(313);                    //ATA13, DTA13
358.    vector_arrivalAndDepartureTimes.push_back(456);                    //ATA14, DTA14
359.    arrivalAndDepartureTimes.push_back(vector_arrivalAndDepartureTimes);
360.    FRi.push_back(306.25);                                    //frequency for first route
361.    APi.push_back(0.1);
362.
363.    vector_arrivalAndDepartureTimes.clear();
364.    vector_arrivalAndDepartureTimes.push_back(312);                    //ATA21, DTA21
365.    vector_arrivalAndDepartureTimes.push_back(313);                    //ATA22, DTA22
366.    vector_arrivalAndDepartureTimes.push_back(314);                    //ATA23, DTA23
367.    arrivalAndDepartureTimes.push_back(vector_arrivalAndDepartureTimes);
368.    FRi.push_back(1);                                     //frequency for second route
369.    APi.push_back(0.1);
370.
371.    vector_arrivalAndDepartureTimes.clear();
372.    vector_arrivalAndDepartureTimes.push_back(290);                    //ATA31, DTA31
373.    vector_arrivalAndDepartureTimes.push_back(516);                    //ATA32, DTA32
374.    vector_arrivalAndDepartureTimes.push_back(742);                    //ATA33, DTA33
375.    arrivalAndDepartureTimes.push_back(vector_arrivalAndDepartureTimes);
376.    FRi.push_back(226);                                   //frequency for third route
377.    APi.push_back(0.1);
378.
379.    //declaration of the decision variables
380.
381.    IloNumVarMatrix2x2 VAR_TOSij(env, 0);
382.
383.    //declaration of the arrival times variables
384.
385.    for (int i = 0; i < 3; i++)
386.    {
387.        IloNumVarArray VAR_TOSi(env, 0);
388.        for (int j = 0; j < 4; j++)
```

```
389.      {
390.         if (!((i == 1) && (j == 3)) && !((i == 2) && (j == 3)))
391.         {
392.            sprintf_s(label, "TOS(%d)(%d)", i, j);
393.            IloNumVar VAR_TOS(env, 0, 2880, ILOINT, label);
394.            VAR_TOSi.add(VAR_TOS);
395.         }
396.      }
397.      VAR_TOSij.add(VAR_TOSi);
398.   }
399.
400.   for (int i = 0; i < 3; i++)
401.   {
402.      for (int j = 0; j < 4; j++)
403.      {
404.         if (!((i == 1) && (j == 3)) && !((i == 2) && (j == 3)))
405.         {
406.            IloExpr expr1(env, 0);
407.            expr1 = VAR_TOSij[i][j];
408.            double Con_LB = arrivalAndDepartureTimes[i][j] - (APi[i] * FRi[i]);
409.            double Con_UB = arrivalAndDepartureTimes[i][j] + (APi[i] * FRi[i]);
410.            IloRange Constraint_1(env, Con_LB, expr1, Con_UB, label);
411.            model.add(Constraint_1);
412.            expr1.end();
413.
414.         }
415.      }
416.   }
417.
418.   IloExpr expr(env, 0);
419.   expr += (VAR_TOSij[1][0] - VAR_TOSij[0][0]) + //1
420.         (VAR_TOSij[2][0] - VAR_TOSij[0][0]) + //2
421.         (VAR_TOSij[1][0] - VAR_TOSij[0][1]) + //3
422.         (VAR_TOSij[2][1] - VAR_TOSij[0][1]) + //4
423.         (VAR_TOSij[1][2] - VAR_TOSij[0][2]) + //5
424.         (VAR_TOSij[2][1] - VAR_TOSij[0][2]) + //6
425.         (VAR_TOSij[2][2] - VAR_TOSij[0][3]) + //7
426.         (VAR_TOSij[0][2] - VAR_TOSij[1][0]) + //8
427.         (VAR_TOSij[2][1] - VAR_TOSij[1][0]) + //9
428.         (VAR_TOSij[0][3] - VAR_TOSij[1][1]) + //10
429.         (VAR_TOSij[2][1] - VAR_TOSij[1][1]) + //11
430.         (VAR_TOSij[0][3] - VAR_TOSij[1][2]) + //12
431.         (VAR_TOSij[2][1] - VAR_TOSij[1][2]) + //13
432.         (VAR_TOSij[0][2] - VAR_TOSij[2][0]) + //14
433.         (VAR_TOSij[1][0] - VAR_TOSij[2][0]) ; //15
434.
435.   model.add(IloMinimize(env, expr));
436.   expr.end();
437.
438.   Cplex.extract(model);
439.   Cplex.exportModel("oneStationProblem_approach_3.lp");
440.
441.   Cplex.solve();
442.
443.   cout << "\n";
444.
445.   cout << "Waiting time before optimization was: " << 1948 << "\n";
```

```
446.    cout << "Waiting time after optimization is (Value of the objective function): " << Cplex.getObjValue();
447.
448.    cout << "\n";
449.
450.    for (int i = 0; i < 3; i++)
451.    {
452.       for (int j = 0; j < 4; j++)
453.       {
454.          if (!((i == 1) && (j == 3)) && !((i == 2) && (j == 3)))
455.          {
456.             cout << "\n";
457.
458.             cout << "The Time of Service to the node with " << i + 1 << " " << j + 1 << " is " << Cplex.getValue(V
      AR_TOSij[i][j]);
459.
460.             cout << "\n";
461.          }
462.       }
463.    }
464.
465.    return 1;
466. }
```

8.3.2 TwoStationProblem.cpp – Self-explanatory title

```
#include <ilcplex/ilocplex.h>
#include <time.h>
#include <vector>
#include <sstream>
#include <fstream>
#include <string>
#include <iostream>
#include <chrono>
#include <map>
#include <thread>
#include <list>
#include <stdio.h>
#include "twoStationProblem.h"

//Data structures used for CPLEX declaration(CPLEX Construct Matrices)
//(used in order to declare later in code with fewer lines of code)
typedef IloArray<IloNumArray>        IloNumMatrix2x2;
typedef IloArray<IloNumMatrix2x2>    IloNumMatrix3x3;
typedef IloArray<IloNumMatrix3x3>    IloNumMatrix4x4;

typedef IloArray<IloNumVarArray>     IloNumVarMatrix2x2;
typedef IloArray<IloNumVarMatrix2x2> IloNumVarMatrix3x3;
typedef IloArray<IloNumVarMatrix3x3> IloNumVarMatrix4x4;

typedef IloArray<IloRangeArray>          IloRangeMatrix2x2;
typedef IloArray<IloRangeMatrix2x2>      IloRangeMatrix3x3;
typedef IloArray<IloRangeMatrix3x3>      IloRangeMatrix4x4;

using namespace std;

int twoStationProblem_approach_1()
{

    //CPLEX parameters
    IloEnv env;
    char label[70];
    IloModel model(env);
    IloCplex Cplex(env);

    //Decision variables
    IloNumVarMatrix2x2 VAR_ATA(env, 0);
    IloNumVarMatrix2x2 VAR_DTA(env, 0);
    IloNumVarMatrix2x2 VAR_ATB(env, 0);
    IloNumVarMatrix2x2 VAR_DTB(env, 0);

    //Data

    std::vector<int>        vector_arrivalAndDepartureTimes_A;
    std::vector<vector<int>>    arrivalAndDepartureTimes_A;             //data for the problem/model

    std::vector<int>        vector_arrivalAndDepartureTimes_B;
    std::vector<vector<int>>    arrivalAndDepartureTimes_B;             //data for the problem/model

```

```
54.  std::vector<double> FRi_A;                    //Frequency of Route i for A
55.  std::vector<double> APi_A;                    //Allowed Percentage to change for route i for A
56.
57.  FRi_A.push_back(2);
58.  APi_A.push_back(0.5);
59.
60.  FRi_A.push_back(15);
61.  APi_A.push_back(0.334);
62.
63.  FRi_A.push_back(300);
64.  APi_A.push_back(1);
65.
66.  FRi_A.push_back(21);
67.  APi_A.push_back(0.55);
68.
69.  FRi_A.push_back(487);
70.  APi_A.push_back(0.334);
71.
72.  std::vector<double> FRi_B;                    //Frequency of Route i for B
73.  std::vector<double> APi_B;                    //Allowed Percentage to change for route i for B
74.
75.  FRi_B.push_back(2);
76.  APi_B.push_back(0.5);
77.
78.  FRi_B.push_back(15);
79.  APi_B.push_back(0.334);
80.
81.  FRi_B.push_back(300);
82.  APi_B.push_back(1);
83.
84.  FRi_B.push_back(3);
85.  APi_B.push_back(0.334);
86.
87.  FRi_B.push_back(300);
88.  APi_B.push_back(1);
89.
90.  //Data for the first stop
91.
92.  //ATA
93.
94.  vector_arrivalAndDepartureTimes_A.push_back(211); //ATA11
95.  vector_arrivalAndDepartureTimes_A.push_back(213); //ATA12
96.  vector_arrivalAndDepartureTimes_A.push_back(0);   //ATA13, does not exists
97.  arrivalAndDepartureTimes_A.push_back(vector_arrivalAndDepartureTimes_A);
98.  vector_arrivalAndDepartureTimes_A.clear();
99.
100. vector_arrivalAndDepartureTimes_A.push_back(415); //ATA21
101. vector_arrivalAndDepartureTimes_A.push_back(430); //ATA22
102. vector_arrivalAndDepartureTimes_A.push_back(445); //ATA23
103. arrivalAndDepartureTimes_A.push_back(vector_arrivalAndDepartureTimes_A);
104. vector_arrivalAndDepartureTimes_A.clear();
105.
106. vector_arrivalAndDepartureTimes_A.push_back(550); //ATA31
107. vector_arrivalAndDepartureTimes_A.push_back(0);   //ATA32, does not exist
108. vector_arrivalAndDepartureTimes_A.push_back(0);   //ATA33, does not exist
109. arrivalAndDepartureTimes_A.push_back(vector_arrivalAndDepartureTimes_A);
110. vector_arrivalAndDepartureTimes_A.clear();
```

```
111.
112.    vector_arrivalAndDepartureTimes_A.push_back(290); //ATA41
113.    vector_arrivalAndDepartureTimes_A.push_back(311); //ATA42
114.    vector_arrivalAndDepartureTimes_A.push_back(0);   //ATA43, does not exist
115.    arrivalAndDepartureTimes_A.push_back(vector_arrivalAndDepartureTimes_A);
116.    vector_arrivalAndDepartureTimes_A.clear();
117.
118.    vector_arrivalAndDepartureTimes_A.push_back(512); //ATA51
119.    vector_arrivalAndDepartureTimes_A.push_back(999); //ATA52
120.    vector_arrivalAndDepartureTimes_A.push_back(0);   //ATA53, does not exist
121.    arrivalAndDepartureTimes_A.push_back(vector_arrivalAndDepartureTimes_A);
122.    vector_arrivalAndDepartureTimes_A.clear();
123.
124.    //DTA
125.
126.    vector_arrivalAndDepartureTimes_A.push_back(211); //DTA11
127.    vector_arrivalAndDepartureTimes_A.push_back(213); //DTA12
128.    vector_arrivalAndDepartureTimes_A.push_back(0);   //DTA13, does not exists
129.    arrivalAndDepartureTimes_A.push_back(vector_arrivalAndDepartureTimes_A);
130.    vector_arrivalAndDepartureTimes_A.clear();
131.
132.    vector_arrivalAndDepartureTimes_A.push_back(415); //DTA21
133.    vector_arrivalAndDepartureTimes_A.push_back(430); //DTA22
134.    vector_arrivalAndDepartureTimes_A.push_back(445); //DTA23
135.    arrivalAndDepartureTimes_A.push_back(vector_arrivalAndDepartureTimes_A);
136.    vector_arrivalAndDepartureTimes_A.clear();
137.
138.    vector_arrivalAndDepartureTimes_A.push_back(550); //DTA31
139.    vector_arrivalAndDepartureTimes_A.push_back(0);   //DTA32, does not exist
140.    vector_arrivalAndDepartureTimes_A.push_back(0);   //DTA33, does not exist
141.    arrivalAndDepartureTimes_A.push_back(vector_arrivalAndDepartureTimes_A);
142.    vector_arrivalAndDepartureTimes_A.clear();
143.
144.    vector_arrivalAndDepartureTimes_A.push_back(290); //DTA41
145.    vector_arrivalAndDepartureTimes_A.push_back(311); //DTA42
146.    vector_arrivalAndDepartureTimes_A.push_back(0);   //DTA43, does not exist
147.    arrivalAndDepartureTimes_A.push_back(vector_arrivalAndDepartureTimes_A);
148.    vector_arrivalAndDepartureTimes_A.clear();
149.
150.    vector_arrivalAndDepartureTimes_A.push_back(512); //DTA51
151.    vector_arrivalAndDepartureTimes_A.push_back(999); //DTA52
152.    vector_arrivalAndDepartureTimes_A.push_back(0);   //DTA53, does not exist
153.    arrivalAndDepartureTimes_A.push_back(vector_arrivalAndDepartureTimes_A);
154.    vector_arrivalAndDepartureTimes_A.clear();
155.
156.    //Data for the second stop
157.
158.    //ATB
159.
160.    vector_arrivalAndDepartureTimes_B.push_back(212); //ATB11
161.    vector_arrivalAndDepartureTimes_B.push_back(214); //ATB12
162.    vector_arrivalAndDepartureTimes_B.push_back(0);   //ATB13, does not exist
163.    arrivalAndDepartureTimes_B.push_back(vector_arrivalAndDepartureTimes_B);
164.    vector_arrivalAndDepartureTimes_B.clear();
165.
166.    vector_arrivalAndDepartureTimes_B.push_back(416); //ATB21
167.    vector_arrivalAndDepartureTimes_B.push_back(431); //ATB22
```

```
168.  vector_arrivalAndDepartureTimes_B.push_back(446); //ATB23
169.  arrivalAndDepartureTimes_B.push_back(vector_arrivalAndDepartureTimes_B);
170.  vector_arrivalAndDepartureTimes_B.clear();
171.
172.  vector_arrivalAndDepartureTimes_B.push_back(551); //ATB31
173.  vector_arrivalAndDepartureTimes_B.push_back(0);   //ATB32, does not exist
174.  vector_arrivalAndDepartureTimes_B.push_back(0);   //ATB33, does not exist
175.  arrivalAndDepartureTimes_B.push_back(vector_arrivalAndDepartureTimes_B);
176.  vector_arrivalAndDepartureTimes_B.clear();
177.
178.  vector_arrivalAndDepartureTimes_B.push_back(516); //ATB61
179.  vector_arrivalAndDepartureTimes_B.push_back(519); //ATB62
180.  vector_arrivalAndDepartureTimes_B.push_back(0);   //ATB63, does not exist
181.  arrivalAndDepartureTimes_B.push_back(vector_arrivalAndDepartureTimes_B);
182.  vector_arrivalAndDepartureTimes_B.clear();
183.
184.  vector_arrivalAndDepartureTimes_B.push_back(650); //ATB71
185.  vector_arrivalAndDepartureTimes_B.push_back(0);   //ATB72, does not exist
186.  vector_arrivalAndDepartureTimes_B.push_back(0);   //ATB73, does not exist
187.  arrivalAndDepartureTimes_B.push_back(vector_arrivalAndDepartureTimes_B);
188.  vector_arrivalAndDepartureTimes_B.clear();
189.
190.  //DTB
191.
192.  vector_arrivalAndDepartureTimes_B.push_back(212); //DTB11
193.  vector_arrivalAndDepartureTimes_B.push_back(214); //DTB12
194.  vector_arrivalAndDepartureTimes_B.push_back(0);   //DTB13, does not exist
195.  arrivalAndDepartureTimes_B.push_back(vector_arrivalAndDepartureTimes_B);
196.  vector_arrivalAndDepartureTimes_B.clear();
197.
198.  vector_arrivalAndDepartureTimes_B.push_back(416); //DTB21
199.  vector_arrivalAndDepartureTimes_B.push_back(431); //DTB22
200.  vector_arrivalAndDepartureTimes_B.push_back(446); //DTB23
201.  arrivalAndDepartureTimes_B.push_back(vector_arrivalAndDepartureTimes_B);
202.  vector_arrivalAndDepartureTimes_B.clear();
203.
204.  vector_arrivalAndDepartureTimes_B.push_back(551); //DTB31
205.  vector_arrivalAndDepartureTimes_B.push_back(0);   //DTB32, does not exist
206.  vector_arrivalAndDepartureTimes_B.push_back(0);   //DTB33, does not exist
207.  arrivalAndDepartureTimes_B.push_back(vector_arrivalAndDepartureTimes_B);
208.  vector_arrivalAndDepartureTimes_B.clear();
209.
210.  vector_arrivalAndDepartureTimes_B.push_back(516); //DTB61
211.  vector_arrivalAndDepartureTimes_B.push_back(519); //DTB62
212.  vector_arrivalAndDepartureTimes_B.push_back(0);   //DTB63, does not exist
213.  arrivalAndDepartureTimes_B.push_back(vector_arrivalAndDepartureTimes_B);
214.  vector_arrivalAndDepartureTimes_B.clear();
215.
216.  vector_arrivalAndDepartureTimes_B.push_back(650); //DTB71
217.  vector_arrivalAndDepartureTimes_B.push_back(0);   //DTB72, does not exist
218.  vector_arrivalAndDepartureTimes_B.push_back(0);   //DTB73, does not exist
219.  arrivalAndDepartureTimes_B.push_back(vector_arrivalAndDepartureTimes_B);
220.  vector_arrivalAndDepartureTimes_B.clear();
221.
222.  //Declaration of the mathematical model
223.
224.  //variables for A
```

```
225.
226.    for (int i = 0; i < 5; i++)
227.    {
228.       IloNumVarArray ATAi(env, 0);
229.       for (int j = 0; j < 3; j++)
230.       {
231.          if (arrivalAndDepartureTimes_A[i][j] != 0)
232.          {
233.             sprintf_s(label, "ATA(%d)(%d)", i, j);
234.             IloNumVar ATA(env, 0, 2880, ILOINT, label);
235.             ATAi.add(ATA);
236.          }
237.       }
238.       VAR_ATA.add(ATAi);
239.    }
240.
241.    for (int i = 0; i < 5; i++)
242.    {
243.       IloNumVarArray DTAi(env, 0);
244.       for (int j = 0; j < 3; j++)
245.       {
246.          if (arrivalAndDepartureTimes_A[i][j] != 0)
247.          {
248.             sprintf_s(label, "DTA(%d)(%d)", i, j);
249.             IloNumVar DTA(env, 0, 2880, ILOINT, label);
250.             DTAi.add(DTA);
251.          }
252.       }
253.       VAR_DTA.add(DTAi);
254.    }
255.
256.    //Variables for B
257.
258.    for (int i = 0; i < 5; i++)
259.    {
260.       IloNumVarArray ATBi(env, 0);
261.       for (int j = 0; j < 3; j++)
262.       {
263.          if (arrivalAndDepartureTimes_B[i][j] != 0)
264.          {
265.             sprintf_s(label, "ATB(%d)(%d)", i, j);
266.             IloNumVar ATB(env, 0, 2880, ILOINT, label);
267.             ATBi.add(ATB);
268.          }
269.       }
270.       VAR_ATB.add(ATBi);
271.    }
272.
273.    for (int i = 0; i < 5; i++)
274.    {
275.       IloNumVarArray DTBi(env, 0);
276.       for (int j = 0; j < 3; j++)
277.       {
278.          if (arrivalAndDepartureTimes_B[i][j] != 0)
279.          {
280.             sprintf_s(label, "DTB(%d)(%d)", i, j);
281.             IloNumVar DTB(env, 0, 2880, ILOINT, label);
```

```
            DTBi.add(DTB);
        }
    }
    VAR_DTB.add(DTBi);
}

//Construction of the model

//Constraints of the model

//ATA has to be 5% bigger or smaller
sprintf_s(label, "");
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < 3; j++)
    {
        if (arrivalAndDepartureTimes_A[i][j] != 0)
        {
            IloExpr expr(env, 0);
            expr = VAR_ATA[i][j];
            double Con_LB = arrivalAndDepartureTimes_A[i][j] - (APi_A[i] * FRi_A[i]);
            double Con_UB = arrivalAndDepartureTimes_A[i][j] + (APi_A[i] * FRi_A[i]);
            IloRange Constraint(env, Con_LB, expr, Con_UB, label);
            model.add(Constraint);
            expr.end();
        }
    }
}

//DTA has to be 5% bigger or smaller
sprintf_s(label, "");
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < 3; j++)
    {
        if (arrivalAndDepartureTimes_A[i][j] != 0)
        {
            IloExpr expr(env, 0);
            expr = VAR_DTA[i][j];
            double Con_LB = arrivalAndDepartureTimes_A[i][j] - (APi_A[i] * FRi_A[i]);
            double Con_UB = arrivalAndDepartureTimes_A[i][j] + (APi_A[i] * FRi_A[i]);
            IloRange Constraint(env, Con_LB, expr, Con_UB, label);
            model.add(Constraint);
            expr.end();
        }
    }
}

//ATB has to be 5% bigger or smaller
sprintf_s(label, "");
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < 3; j++)
    {
        if (arrivalAndDepartureTimes_B[i][j] != 0)
        {
            IloExpr expr(env, 0);
```

```
            expr = VAR_ATB[i][j];
            double Con_LB = arrivalAndDepartureTimes_B[i][j] - (APi_B[i] * FRi_B[i]);
            double Con_UB = arrivalAndDepartureTimes_B[i][j] + (APi_B[i] * FRi_B[i]);
            IloRange Constraint(env, Con_LB, expr, Con_UB, label);
            model.add(Constraint);
            expr.end();
        }
    }
}

//DTB has to be 5% bigger or smaller
sprintf_s(label, "");
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < 3; j++)
    {
        if (arrivalAndDepartureTimes_B[i][j] != 0)
        {
            IloExpr expr(env, 0);
            expr = VAR_DTB[i][j];
            double Con_LB = arrivalAndDepartureTimes_B[i][j] - (APi_B[i] * FRi_B[i]);
            double Con_UB = arrivalAndDepartureTimes_B[i][j] + (APi_B[i] * FRi_B[i]);
            IloRange Constraint(env, Con_LB, expr, Con_UB, label);
            model.add(Constraint);
            expr.end();
        }
    }
}

//Constraints about: ATA has to be equal to DTA

sprintf_s(label, "");
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < 3; j++)
    {
        if (arrivalAndDepartureTimes_A[i][j] != 0)
        {
            IloExpr expr(env, 0);
            expr = VAR_ATA[i][j] - VAR_DTA[i][j];
            double Con_LB = 0;
            double Con_UB = 0;
            IloRange Constraint(env, Con_LB, expr, Con_UB, label);
            model.add(Constraint);
            expr.end();
        }
    }
}

//Constraints about: ATB has to be equal to DTB

sprintf_s(label, "");
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < 3; j++)
    {
        if (arrivalAndDepartureTimes_B[i][j] != 0)
```

```
396.          {
397.             IloExpr expr(env, 0);
398.             expr = VAR_ATB[i][j] - VAR_DTB[i][j];
399.             double Con_LB = 0;
400.             double Con_UB = 0;
401.             IloRange Constraint(env, Con_LB, expr, Con_UB, label);
402.             model.add(Constraint);
403.             expr.end();
404.          }
405.       }
406.    }
407.
408.    //Constraints about: DTA has to be equal to ATB - 1
409.
410.    sprintf_s(label, "");
411.    for (int i = 0; i < 3; i++)
412.    {
413.       for (int j = 0; j < 3; j++)
414.       {
415.          if (arrivalAndDepartureTimes_B[i][j] != 0)
416.          {
417.             IloExpr expr(env, 0);
418.             expr = VAR_DTA[i][j] - VAR_ATB[i][j];
419.             double Con_LB = -1;
420.             double Con_UB = -1;
421.             IloRange Constraint(env, Con_LB, expr, Con_UB, label);
422.             model.add(Constraint);
423.             expr.end();
424.          }
425.       }
426.    }
427.
428.    //Objective function of the problem
429.    IloExpr expr14(env, 0);
430.    expr14 = \
431.       //Waiting times at A
432.       (VAR_DTA[1][0] - VAR_ATA[0][0]) + (VAR_DTA[2][0] - VAR_ATA[0][0]) + (VAR_DTA[3][0] -
    VAR_ATA[0][0]) + (VAR_DTA[4][0] - VAR_ATA[0][0]) + \
433.       (VAR_DTA[1][0] - VAR_ATA[0][1]) + (VAR_DTA[3][0] - VAR_ATA[0][1]) + (VAR_DTA[2][0] -
    VAR_ATA[0][1]) + (VAR_DTA[4][0] - VAR_ATA[0][1]) + \
434.       (VAR_DTA[2][0] - VAR_ATA[1][0]) + (VAR_DTA[4][0] - VAR_ATA[1][0]) + \
435.       (VAR_DTA[2][0] - VAR_ATA[1][1]) + (VAR_DTA[4][0] - VAR_ATA[1][1]) + \
436.       (VAR_DTA[2][0] - VAR_ATA[1][2]) + (VAR_DTA[4][0] - VAR_ATA[1][2]) + \
437.       (VAR_DTA[4][1] - VAR_ATA[2][0]) + \
438.       (VAR_DTA[1][0] - VAR_ATA[3][0]) + (VAR_DTA[2][0] - VAR_ATA[3][0]) + (VAR_DTA[4][0] -
    VAR_ATA[3][0]) + \
439.       (VAR_DTA[1][0] - VAR_ATA[3][1]) + (VAR_DTA[2][0] - VAR_ATA[3][1]) + (VAR_DTA[4][0] -
    VAR_ATA[3][1]) + \
440.       (VAR_DTA[2][0] - VAR_ATA[4][0]) + \
441.       //Waiting times at B
442.       (VAR_DTB[1][0] - VAR_ATB[0][0]) + (VAR_DTB[2][0] - VAR_ATB[0][0]) + (VAR_DTB[3][0] -
    VAR_ATB[0][0]) + (VAR_DTB[4][0] - VAR_ATB[0][0]) + \
443.       (VAR_DTB[1][0] - VAR_ATB[0][1]) + (VAR_DTB[2][0] - VAR_ATB[0][1]) + (VAR_DTB[3][0] -
    VAR_ATB[0][1]) + (VAR_DTB[4][0] - VAR_ATB[0][1]) +\
444.       (VAR_DTB[2][0] - VAR_ATB[1][0]) + (VAR_DTB[3][0] - VAR_ATB[1][0]) + (VAR_DTB[4][0] -
    VAR_ATB[1][0]) + \
```

```
445.       (VAR_DTB[2][0] - VAR_ATB[1][1]) + (VAR_DTB[3][0] - VAR_ATB[1][1]) + (VAR_DTB[4][0] -
    VAR_ATB[1][1]) + \
446.       (VAR_DTB[2][0] - VAR_ATB[1][2]) + (VAR_DTB[3][0] - VAR_ATB[1][2]) + (VAR_DTB[4][0] -
    VAR_ATB[1][2]) + \
447.       (VAR_DTB[4][0] - VAR_ATB[2][0]) + (VAR_DTB[4][0] - VAR_ATB[3][0]) + (VAR_DTB[4][0] -
    VAR_ATB[3][1]);
448.
449.    model.add(IloMinimize(env, expr14));
450.    expr14.end();
451.
452.    Cplex.extract(model);
453.    Cplex.exportModel("twoStationProblem_approach_1.lp");
454.
455.    Cplex.solve();
456.
457.    cout << "\n";
458.
459.    cout << "Waiting time before optimization was: " << 8280 << "\n";
460.    cout << "Waiting time after optimization is (Value of the objective function): " << Cplex.getObjValue();
461.
462.    cout << "\n";
463.    cout << "\n";
464.
465.    for (int i = 0; i < 5; i++)
466.    {
467.       for (int j = 0; j < 3; j++)
468.       {
469.          if (arrivalAndDepartureTimes_A[i][j] != 0)
470.          {
471.             cout << "\n";
472.
473.             cout << "The Arrival time to node A with " << i + 1 << " " << j + 1 << " is " << Cplex.getValue(VAR_
    ATA[i][j]) << \
474.                " and departure is " << Cplex.getValue(VAR_DTA[i][j]);
475.
476.             cout << "\n";
477.          }
478.       }
479.    }
480.
481.    for (int i = 0; i < 5; i++)
482.    {
483.       for (int j = 0; j < 3; j++)
484.       {
485.          if (arrivalAndDepartureTimes_B[i][j] != 0)
486.          {
487.             cout << "\n";
488.
489.             cout << "The Arrival time to node B with " << i + 1 << " " << j + 1 << " is " << Cplex.getValue(VAR_
    ATB[i][j]) << \
490.                " and departure is " << Cplex.getValue(VAR_DTB[i][j]);
491.
492.             cout << "\n";
493.          }
494.       }
495.    }
496.
```

```
497.    return 1;
498. }
499.
500. int twoStationProblem_approach_2()
501. {
502.    //CPLEX parameters
503.    IloEnv env;
504.    char label[70];
505.    IloModel model(env);
506.    IloCplex Cplex(env);
507.
508.    IloNumVarArray VAR_NEAi(env, 0);
509.    IloNumVarArray VAR_NEBi(env, 0);
510.
511.    //data
512.
513.    std::vector<int> arrivalAndDepartureTimes_A;
514.    std::vector<int> arrivalAndDepartureTimes_B;
515.
516.    //Data for first stop
517.
518.    arrivalAndDepartureTimes_A.push_back(211); //TOSA11
519.    arrivalAndDepartureTimes_A.push_back(213); //TOSA12
520.
521.    arrivalAndDepartureTimes_A.push_back(415); //TOSA21
522.    arrivalAndDepartureTimes_A.push_back(430); //TOSA22
523.    arrivalAndDepartureTimes_A.push_back(445); //TOSA23
524.
525.    arrivalAndDepartureTimes_A.push_back(550); //TOSA31
526.
527.    arrivalAndDepartureTimes_A.push_back(290); //TOSA41
528.    arrivalAndDepartureTimes_A.push_back(311); //TOSA42
529.
530.    arrivalAndDepartureTimes_A.push_back(512); //TOSA51
531.    arrivalAndDepartureTimes_A.push_back(999); //TOSA52
532.
533.    //Data for the second stop
534.
535.    arrivalAndDepartureTimes_B.push_back(212); //TOSB11
536.    arrivalAndDepartureTimes_B.push_back(214); //TOSB12
537.
538.    arrivalAndDepartureTimes_B.push_back(416); //TOSB21
539.    arrivalAndDepartureTimes_B.push_back(431); //TOSB22
540.    arrivalAndDepartureTimes_B.push_back(446); //TOSB23
541.
542.    arrivalAndDepartureTimes_B.push_back(551); //TOSB31
543.
544.    arrivalAndDepartureTimes_B.push_back(516); //TOSB61
545.    arrivalAndDepartureTimes_B.push_back(519); //TOSB62
546.
547.    arrivalAndDepartureTimes_B.push_back(650); //TOSB71
548.
549.    IloNumVarArray VAR_NEi_A(env, 0);
550.    IloNumVarArray VAR_NEi_B(env, 0);
551.
552.    std::vector<double> FRi_A;                    //Frequency of Route i for A
553.    std::vector<double> APi_A;                    //Allowed Percentage to change for route i for A
```

```
554.
555.    FRi_A.push_back(2);
556.    APi_A.push_back(0.5);
557.
558.    FRi_A.push_back(15);
559.    APi_A.push_back(0.334);
560.
561.    FRi_A.push_back(300);
562.    APi_A.push_back(1);
563.
564.    FRi_A.push_back(21);
565.    APi_A.push_back(0.55);
566.
567.    FRi_A.push_back(487);
568.    APi_A.push_back(0.334);
569.
570.    std::vector<double> FRi_B;                    //Frequency of Route i for B
571.    std::vector<double> APi_B;                    //Allowed Percentage to change for route i for B
572.
573.    FRi_B.push_back(2);
574.    APi_B.push_back(0.5);
575.
576.    FRi_B.push_back(15);
577.    APi_B.push_back(0.334);
578.
579.    FRi_B.push_back(300);
580.    APi_B.push_back(1);
581.
582.    FRi_B.push_back(3);
583.    APi_B.push_back(0.334);
584.
585.    FRi_B.push_back(300);
586.    APi_B.push_back(1);
587.
588.    //declaration of the model
589.
590.    //declaration of the decision variables
591.
592.    //for stop A
593.
594.    for (int i = 0; i < 10; i++)
595.    {
596.       sprintf_s(label, "NEA(%d)", i);
597.       IloNumVar VAR_NEA(env, 0, 2880, ILOINT, label);
598.       VAR_NEAi.add(VAR_NEA);
599.    }
600.
601.    //for stop B
602.
603.    for (int i = 0; i < 9; i++)
604.    {
605.       sprintf_s(label, "NEB(%d)", i);
606.       IloNumVar VAR_NEB(env, 0, 2880, ILOINT, label);
607.       VAR_NEBi.add(VAR_NEB);
608.    }
609.
610.    //declaration of constraints
```

```
611.
612.    for (int i = 0; i < 10; i++)
613.    {
614.        int j;
615.        if (i <= 1)
616.        {
617.            j = 0;
618.        }
619.        else if (i <= 4)
620.        {
621.            j = 1;
622.        }
623.        else if (i == 5)
624.        {
625.            j = 2;
626.        }
627.        else if (i <= 7)
628.        {
629.            j = 3;
630.        }
631.        else
632.        {
633.            j = 4;
634.        }
635.        sprintf_s(label, "Constraint  A: (%d)", i);
636.        IloExpr expr(env, 0);
637.        expr = VAR_NEAi[i];
638.        double Con_LB = arrivalAndDepartureTimes_A[i] - (APi_A[j] * FRi_A[j]);
639.        double Con_UB = arrivalAndDepartureTimes_A[i] + (APi_A[j] * FRi_A[j]);
640.        IloRange Constraint_1(env, Con_LB, expr, Con_UB, label);
641.        model.add(Constraint_1);
642.        expr.end();
643.    }
644.
645.    for (int i = 0; i < 9; i++)
646.    {
647.        int j;
648.        if (i <= 1)
649.        {
650.            j = 0;
651.        }
652.        else if (i <= 4)
653.        {
654.            j = 1;
655.        }
656.        else if (i == 5)
657.        {
658.            j = 2;
659.        }
660.        else if (i <= 7)
661.        {
662.            j = 3;
663.        }
664.        else
665.        {
666.            j = 4;
667.        }
```

```
668.      sprintf_s(label, "Constraint  B: (%d)", i);
669.      IloExpr expr(env, 0);
670.      expr = VAR_NEBi[i];
671.      double Con_LB = arrivalAndDepartureTimes_B[i] - (APi_B[j] * FRi_B[j]);
672.      double Con_UB = arrivalAndDepartureTimes_B[i] + (APi_B[j] * FRi_B[j]);
673.      IloRange Constraint_1(env, Con_LB, expr, Con_UB, label);
674.      model.add(Constraint_1);
675.      expr.end();
676.   }
677.
678.   sprintf_s(label, "");
679.   int counter = 0;
680.   for (int i = 0; i < 3; i++)
681.   {
682.      for (int j = 0; j < 3; j++)
683.      {
684.         if (!((i == 0) && (j == 2)))
685.         {
686.            if (!((i == 2) && (j > 0)))
687.            {
688.               IloExpr expr(env, 0);
689.               expr = VAR_NEAi[counter] - VAR_NEBi[counter];
690.               double Con_LB = -1;
691.               double Con_UB = -1;
692.               IloRange Constraint(env, Con_LB, expr, Con_UB, label);
693.               model.add(Constraint);
694.               expr.end();
695.               counter++;
696.            }
697.         }
698.      }
699.   }
700.
701.   IloExpr expr14(env, 0);
702.   expr14 = \
703.      //Waiting times at A
704.      (VAR_NEAi[2] - VAR_NEAi[0]) + (VAR_NEAi[5] - VAR_NEAi[0]) + (VAR_NEAi[6] -
      VAR_NEAi[0]) + (VAR_NEAi[8] - VAR_NEAi[0]) + \
705.      (VAR_NEAi[5] - VAR_NEAi[4]) + (VAR_NEAi[8] - VAR_NEAi[4]) + \
706.      (VAR_NEAi[2] - VAR_NEAi[1]) + (VAR_NEAi[6] - VAR_NEAi[1]) + (VAR_NEAi[5] -
      VAR_NEAi[1]) + (VAR_NEAi[8] - VAR_NEAi[1]) + \
707.      (VAR_NEAi[9] - VAR_NEAi[5]) + \
708.      (VAR_NEAi[5] - VAR_NEAi[2]) + (VAR_NEAi[8] - VAR_NEAi[2]) + \
709.      (VAR_NEAi[2] - VAR_NEAi[6]) + (VAR_NEAi[5] - VAR_NEAi[6]) + (VAR_NEAi[8] -
      VAR_NEAi[6]) + \
710.      (VAR_NEAi[5] - VAR_NEAi[3]) + (VAR_NEAi[8] - VAR_NEAi[3]) + \
711.      (VAR_NEAi[2] - VAR_NEAi[7]) + (VAR_NEAi[5] - VAR_NEAi[7]) + (VAR_NEAi[8] -
      VAR_NEAi[7]) + \
712.      (VAR_NEAi[5] - VAR_NEAi[8]) +\
713.      //Waiting times at B
714.      (VAR_NEBi[2] - VAR_NEBi[0]) + (VAR_NEBi[5] - VAR_NEBi[0]) + (VAR_NEBi[6] -
      VAR_NEBi[0]) + (VAR_NEBi[8] - VAR_NEBi[0]) + \
715.      (VAR_NEBi[2] - VAR_NEBi[1]) + (VAR_NEBi[5] - VAR_NEBi[1]) + (VAR_NEBi[6] -
      VAR_NEBi[1]) + (VAR_NEBi[8] - VAR_NEBi[1]) +\
716.      (VAR_NEBi[5] - VAR_NEBi[2]) + (VAR_NEBi[6] - VAR_NEBi[2]) + (VAR_NEBi[8] - VAR_NEBi[2]) + \
717.      (VAR_NEBi[5] - VAR_NEBi[3]) + (VAR_NEBi[6] - VAR_NEBi[3]) + (VAR_NEBi[8] - VAR_NEBi[3]) + \
718.      (VAR_NEBi[5] - VAR_NEBi[4]) + (VAR_NEBi[6] - VAR_NEBi[4]) + (VAR_NEBi[8] - VAR_NEBi[4]) + \
```

```
719.       (VAR_NEBi[8] - VAR_NEBi[5]) + (VAR_NEBi[8] - VAR_NEBi[6]) + (VAR_NEBi[8] - VAR_NEBi[7]);
720.
721.    model.add(IloMinimize(env, expr14));
722.    expr14.end();
723.
724.    Cplex.extract(model);
725.    Cplex.exportModel("twoStationProblem_approach_2.lp");
726.
727.    Cplex.solve();
728.
729.    cout << "\n";
730.
731.    cout << "Waiting time before optimization was: " << 8280 << "\n";
732.    cout << "Waiting time after optimization is (Value of the objective function): " << Cplex.getObjValue();
733.
734.    cout << "\n";
735.
736.    for (int i = 0; i < 10; i++)
737.    {
738.       cout << "\n";
739.
740.       cout << "The Network Event at A with index " << i + 1 << " is " << Cplex.getValue(VAR_NEAi[i]);
741.
742.       cout << "\n";
743.    }
744.
745.    for (int i = 0; i < 9; i++)
746.    {
747.       cout << "\n";
748.
749.       cout << "The Network Event at B with index " << i + 1 << " is " << Cplex.getValue(VAR_NEBi[i]);
750.
751.       cout << "\n";
752.    }
753.
754.    return 1;
755. }
756.
757. int twoStationProblem_approach_3()
758. {
759.
760.    //CPLEX parameters
761.    IloEnv env;
762.    char label[70];
763.    IloModel model(env);
764.    IloCplex Cplex(env);
765.
766.    IloNumVarMatrix2x2 VAR_TOSAij(env, 0);
767.    IloNumVarMatrix2x2 VAR_TOSBij(env, 0);
768.
769.    //Data
770.
771.    std::vector<int>        vector_arrivalAndDepartureTimes_A;
772.    std::vector<vector<int>>    arrivalAndDepartureTimes_A;          //data for the problem/model
773.
774.    std::vector<int>        vector_arrivalAndDepartureTimes_B;
775.    std::vector<vector<int>>    arrivalAndDepartureTimes_B;          //data for the problem/model
```

```
776.
777.  std::vector<double> FRi_A;                    //Frequency of Route i for A
778.  std::vector<double> APi_A;                     //Allowed Percentage to change for route i for A
779.
780.  FRi_A.push_back(2);
781.  APi_A.push_back(0.5);
782.
783.  FRi_A.push_back(15);
784.  APi_A.push_back(0.334);
785.
786.  FRi_A.push_back(300);
787.  APi_A.push_back(1);
788.
789.  FRi_A.push_back(21);
790.  APi_A.push_back(0.55);
791.
792.  FRi_A.push_back(487);
793.  APi_A.push_back(0.334);
794.
795.  std::vector<double> FRi_B;                    //Frequency of Route i for B
796.  std::vector<double> APi_B;                     //Allowed Percentage to change for route i for B
797.
798.  FRi_B.push_back(2);
799.  APi_B.push_back(0.5);
800.
801.  FRi_B.push_back(15);
802.  APi_B.push_back(0.334);
803.
804.  FRi_B.push_back(300);
805.  APi_B.push_back(1);
806.
807.  FRi_B.push_back(3);
808.  APi_B.push_back(0.334);
809.
810.  FRi_B.push_back(300);
811.  APi_B.push_back(1);
812.
813.  //Data for the first stop
814.
815.  //Non synced TOSAij(data)
816.
817.  vector_arrivalAndDepartureTimes_A.push_back(211); //ATA11
818.  vector_arrivalAndDepartureTimes_A.push_back(213); //ATA12
819.  vector_arrivalAndDepartureTimes_A.push_back(0);   //ATA13, does not exists
820.  arrivalAndDepartureTimes_A.push_back(vector_arrivalAndDepartureTimes_A);
821.  vector_arrivalAndDepartureTimes_A.clear();
822.
823.  vector_arrivalAndDepartureTimes_A.push_back(415); //ATA21
824.  vector_arrivalAndDepartureTimes_A.push_back(430); //ATA22
825.  vector_arrivalAndDepartureTimes_A.push_back(445); //ATA23
826.  arrivalAndDepartureTimes_A.push_back(vector_arrivalAndDepartureTimes_A);
827.  vector_arrivalAndDepartureTimes_A.clear();
828.
829.  vector_arrivalAndDepartureTimes_A.push_back(550); //ATA31
830.  vector_arrivalAndDepartureTimes_A.push_back(0);   //ATA32, does not exist
831.  vector_arrivalAndDepartureTimes_A.push_back(0);   //ATA33, does not exist
832.  arrivalAndDepartureTimes_A.push_back(vector_arrivalAndDepartureTimes_A);
```

```
833.  vector_arrivalAndDepartureTimes_A.clear();
834.
835.  vector_arrivalAndDepartureTimes_A.push_back(290); //ATA41
836.  vector_arrivalAndDepartureTimes_A.push_back(311); //ATA42
837.  vector_arrivalAndDepartureTimes_A.push_back(0);   //ATA43, does not exist
838.  arrivalAndDepartureTimes_A.push_back(vector_arrivalAndDepartureTimes_A);
839.  vector_arrivalAndDepartureTimes_A.clear();
840.
841.  vector_arrivalAndDepartureTimes_A.push_back(512); //ATA51
842.  vector_arrivalAndDepartureTimes_A.push_back(999); //ATA52
843.  vector_arrivalAndDepartureTimes_A.push_back(0);   //ATA53, does not exist
844.  arrivalAndDepartureTimes_A.push_back(vector_arrivalAndDepartureTimes_A);
845.  vector_arrivalAndDepartureTimes_A.clear();
846.
847.  //Data for the second stop
848.
849.  //Non synced TOSBij(data)
850.
851.  vector_arrivalAndDepartureTimes_B.push_back(212); //ATB11
852.  vector_arrivalAndDepartureTimes_B.push_back(214); //ATB12
853.  vector_arrivalAndDepartureTimes_B.push_back(0);   //ATB13, does not exist
854.  arrivalAndDepartureTimes_B.push_back(vector_arrivalAndDepartureTimes_B);
855.  vector_arrivalAndDepartureTimes_B.clear();
856.
857.  vector_arrivalAndDepartureTimes_B.push_back(416); //ATB21
858.  vector_arrivalAndDepartureTimes_B.push_back(431); //ATB22
859.  vector_arrivalAndDepartureTimes_B.push_back(446); //ATB23
860.  arrivalAndDepartureTimes_B.push_back(vector_arrivalAndDepartureTimes_B);
861.  vector_arrivalAndDepartureTimes_B.clear();
862.
863.  vector_arrivalAndDepartureTimes_B.push_back(551); //ATB31
864.  vector_arrivalAndDepartureTimes_B.push_back(0);   //ATB32, does not exist
865.  vector_arrivalAndDepartureTimes_B.push_back(0);   //ATB33, does not exist
866.  arrivalAndDepartureTimes_B.push_back(vector_arrivalAndDepartureTimes_B);
867.  vector_arrivalAndDepartureTimes_B.clear();
868.
869.  vector_arrivalAndDepartureTimes_B.push_back(516); //ATB61
870.  vector_arrivalAndDepartureTimes_B.push_back(519); //ATB62
871.  vector_arrivalAndDepartureTimes_B.push_back(0);   //ATB63, does not exist
872.  arrivalAndDepartureTimes_B.push_back(vector_arrivalAndDepartureTimes_B);
873.  vector_arrivalAndDepartureTimes_B.clear();
874.
875.  vector_arrivalAndDepartureTimes_B.push_back(650); //ATB71
876.  vector_arrivalAndDepartureTimes_B.push_back(0);   //ATB72, does not exist
877.  vector_arrivalAndDepartureTimes_B.push_back(0);   //ATB73, does not exist
878.  arrivalAndDepartureTimes_B.push_back(vector_arrivalAndDepartureTimes_B);
879.  vector_arrivalAndDepartureTimes_B.clear();
880.
881.  //declaration of the model
882.  for (int i = 0; i < 5; i++)
883.  {
884.     IloNumVarArray TOSAi(env, 0);
885.     for (int j = 0; j < 3; j++)
886.     {
887.        if (arrivalAndDepartureTimes_A[i][j] != 0)
888.        {
889.           sprintf_s(label, "TOSAij(%d)(%d)", i, j);
```

```
            IloNumVar TOSA(env, 0, 2880, ILOINT, label);
            TOSAi.add(TOSA);
        }
    }
    VAR_TOSAij.add(TOSAi);
}

for (int i = 0; i < 5; i++)
{
    IloNumVarArray TOSBi(env, 0);
    for (int j = 0; j < 3; j++)
    {
        if (arrivalAndDepartureTimes_B[i][j] != 0)
        {
            sprintf_s(label, "TOSBij(%d)(%d)", i, j);
            IloNumVar TOSB(env, 0, 2880, ILOINT, label);
            TOSBi.add(TOSB);
        }
    }
    VAR_TOSBij.add(TOSBi);
}

sprintf_s(label, "");
for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < 3; j++)
    {
        if (arrivalAndDepartureTimes_A[i][j] != 0)
        {
            IloExpr expr(env, 0);
            expr = VAR_TOSAij[i][j];
            double Con_LB = arrivalAndDepartureTimes_A[i][j] - (APi_A[i] * FRi_A[i]);
            double Con_UB = arrivalAndDepartureTimes_A[i][j] + (APi_A[i] * FRi_A[i]);
            IloRange Constraint(env, Con_LB, expr, Con_UB, label);
            model.add(Constraint);
            expr.end();
        }
    }
}

for (int i = 0; i < 5; i++)
{
    for (int j = 0; j < 3; j++)
    {
        if (arrivalAndDepartureTimes_B[i][j] != 0)
        {
            IloExpr expr(env, 0);
            expr = VAR_TOSBij[i][j];
            double Con_LB = arrivalAndDepartureTimes_B[i][j] -(APi_B[i] * FRi_B[i]);
            double Con_UB = arrivalAndDepartureTimes_B[i][j] +(APi_B[i] * FRi_B[i]);
            IloRange Constraint(env, Con_LB, expr, Con_UB, label);
            model.add(Constraint);
            expr.end();
        }
    }
}

```

```
947.    //Constraints about: DTA has to be equal to ATB - 1
948.
949.    sprintf_s(label, "");
950.    for (int i = 0; i < 3; i++)
951.    {
952.       for (int j = 0; j < 3; j++)
953.       {
954.          if (arrivalAndDepartureTimes_A[i][j] != 0)
955.          {
956.             IloExpr expr(env, 0);
957.             expr = VAR_TOSAij[i][j] - VAR_TOSBij[i][j];
958.             double Con_LB = -1;
959.             double Con_UB = -1;
960.             IloRange Constraint(env, Con_LB, expr, Con_UB, label);
961.             model.add(Constraint);
962.             expr.end();
963.          }
964.       }
965.    }
966.
967.    //Objective function of the problem
968.    IloExpr expr14(env, 0);
969.    expr14 = \
970.       //Waiting times at A
971.       (VAR_TOSAij[1][0] - VAR_TOSAij[0][0]) + (VAR_TOSAij[2][0] -
       VAR_TOSAij[0][0]) + (VAR_TOSAij[3][0] - VAR_TOSAij[0][0]) + (VAR_TOSAij[4][0] -
       VAR_TOSAij[0][0]) + \
972.       (VAR_TOSAij[1][0] - VAR_TOSAij[0][1]) + (VAR_TOSAij[3][0] -
       VAR_TOSAij[0][1]) + (VAR_TOSAij[2][0] - VAR_TOSAij[0][1]) + (VAR_TOSAij[4][0] -
       VAR_TOSAij[0][1]) + \
973.       (VAR_TOSAij[2][0] - VAR_TOSAij[1][0]) + (VAR_TOSAij[4][0] - VAR_TOSAij[1][0]) + \
974.       (VAR_TOSAij[2][0] - VAR_TOSAij[1][1]) + (VAR_TOSAij[4][0] - VAR_TOSAij[1][1]) + \
975.       (VAR_TOSAij[2][0] - VAR_TOSAij[1][2]) + (VAR_TOSAij[4][0] - VAR_TOSAij[1][2]) + \
976.       (VAR_TOSAij[4][1] - VAR_TOSAij[2][0]) + \
977.       (VAR_TOSAij[1][0] - VAR_TOSAij[3][0]) + (VAR_TOSAij[2][0] -
       VAR_TOSAij[3][0]) + (VAR_TOSAij[4][0] - VAR_TOSAij[3][0]) + \
978.       (VAR_TOSAij[1][0] - VAR_TOSAij[3][1]) + (VAR_TOSAij[2][0] -
       VAR_TOSAij[3][1]) + (VAR_TOSAij[4][0] - VAR_TOSAij[3][1]) + \
979.       (VAR_TOSAij[2][0] - VAR_TOSAij[4][0]) + \
980.       //Waiting times at B
981.       (VAR_TOSBij[1][0] - VAR_TOSBij[0][0]) + (VAR_TOSBij[2][0] -
       VAR_TOSBij[0][0]) + (VAR_TOSBij[3][0] - VAR_TOSBij[0][0]) + (VAR_TOSBij[4][0] -
       VAR_TOSBij[0][0]) + \
982.       (VAR_TOSBij[1][0] - VAR_TOSBij[0][1]) + (VAR_TOSBij[2][0] -
       VAR_TOSBij[0][1]) + (VAR_TOSBij[3][0] - VAR_TOSBij[0][1]) + (VAR_TOSBij[4][0] -
       VAR_TOSBij[0][1]) + \
983.       (VAR_TOSBij[2][0] - VAR_TOSBij[1][0]) + (VAR_TOSBij[3][0] -
       VAR_TOSBij[1][0]) + (VAR_TOSBij[4][0] - VAR_TOSBij[1][0]) + \
984.       (VAR_TOSBij[2][0] - VAR_TOSBij[1][1]) + (VAR_TOSBij[3][0] -
       VAR_TOSBij[1][1]) + (VAR_TOSBij[4][0] - VAR_TOSBij[1][1]) + \
985.       (VAR_TOSBij[2][0] - VAR_TOSBij[1][2]) + (VAR_TOSBij[3][0] -
       VAR_TOSBij[1][2]) + (VAR_TOSBij[4][0] - VAR_TOSBij[1][2]) + \
986.       (VAR_TOSBij[4][0] - VAR_TOSBij[2][0]) + (VAR_TOSBij[4][0] -
       VAR_TOSBij[3][0]) + (VAR_TOSBij[4][0] - VAR_TOSBij[3][1]);
987.
988.    model.add(IloMinimize(env, expr14));
989.    expr14.end();
```

```
990.
991.    Cplex.extract(model);
992.    Cplex.exportModel("twoStationProblem_approach_3.lp");
993.
994.    Cplex.solve();
995.
996.    cout << "\n";
997.
998.    cout << "Waiting time before optimization was: " << 8280 << "\n";
999.    cout << "Waiting time after optimization is (Value of the objective function): " << Cplex.getObjValue() <<
        endl;
1000.
1001.               for (int i = 0; i < 5; i++)
1002.               {
1003.                  for (int j = 0; j < 3; j++)
1004.                  {
1005.                     if (arrivalAndDepartureTimes_A[i][j] != 0)
1006.                     {
1007.                        cout << "\n";
1008.
1009.                        cout << "The Time of Service for node A with " << i + 1 << " " << j + 1 << " is " << Cplex.ge
        tValue(VAR_TOSAij[i][j]);
1010.
1011.                        cout << "\n";
1012.                     }
1013.                  }
1014.               }
1015.
1016.               for (int i = 0; i < 5; i++)
1017.               {
1018.                  for (int j = 0; j < 3; j++)
1019.                  {
1020.                     if (arrivalAndDepartureTimes_B[i][j] != 0)
1021.                     {
1022.                        cout << "\n";
1023.
1024.                        cout << "The Time of Service for node B with " << i + 1 << " " << j + 1 << " is " << Cplex.get
        Value(VAR_TOSBij[i][j]);
1025.
1026.                        cout << "\n";
1027.                     }
1028.                  }
1029.               }
1030.
1031.               return 1;
1032.            }
```

8.4 Athens Metro case study

Below is the file that implements the main optimization phase(declaration and solution of the mathematical model).

athensMetroCaseStudyAllTripsPerRoute.cpp

```
#include <ilcplex/ilocplex.h>
#include <time.h>
#include <vector>
#include <sstream>
#include <fstream>
#include <utility>
#include <string>
#include <iostream>
#include <chrono>
#include <map>
#include <thread>
#include <list>
#include <stdio.h>
#include <array>

using namespace std;

//The model used in this file:
//..
//..
//..
//..

int athensMetroAllTrips_1() {

    //cols of the gtfs, fixed for any gtfs file
    int const stops_cols = 8;
    int const routes_cols = 9;
    int const trips_cols = 7;
    int const calendar_cols = 10;
    int const stop_times_cols = 7;

    //rows of the gtfs, change for each gtfs file
    int const stops_rows = 390;
    int const routes_rows = 17;
    int const trips_rows = 2303;
    int const calendar_rows = 4;
    int const stop_times_rows = 61575;

    int index_1;
    int index_2;
    int index_3;
    int index_4;
    int index_5;
```

```
45.     int index_6;
46.
47.     double quantity;
48.
49.     vector<vector<string>> decisionVars;
50.     vector<vector<string>> parameters;
51.     vector<vector<string>> TTs1s2rt;
52.     vector<vector<string>> objFun;
53.
54.     //read file
55.     stringstream ss_pathToReadFile;
56.     ss_pathToReadFile << "Data folder/athens_metro/toFeedToModel/interfaceToModel/decisionVars.txt";
57.     string pathToReadFile = ss_pathToReadFile.str();
58.     std::ifstream file(pathToReadFile);
59.
60.     if (file)
61.     {
62.         for (int row = 0; row < stop_times_rows; row++)
63.         {
64.             std::string line;
65.             std::getline(file, line);
66.             if (!file.good()) break;
67.
68.             std::stringstream iss(line);
69.             vector<string> rowOfFile;
70.             for (int col = 0; col < stop_times_cols; col++)
71.             {
72.                 std::string val;
73.                 std::getline(iss, val, ',');
74.
75.                 std::stringstream convertor(val);
76.                 //convertor >> readData[row][col];
77.                 rowOfFile.push_back(convertor.str());
78.
79.                 if (!iss.good())
80.                     break;
81.             }
82.             decisionVars.push_back(rowOfFile);
83.         }
84.     }
85.
86.     stringstream ss_pathToReadFile1;
87.     ss_pathToReadFile1 << "Data folder/athens_metro/toFeedToModel/interfaceToModel/parameters.txt";
88.     string pathToReadFile1 = ss_pathToReadFile1.str();
89.     std::ifstream file1(pathToReadFile1);
90.
91.     if (file1)
92.     {
93.         for (int row = 0; row < stop_times_rows; row++)
94.         {
95.             std::string line;
96.             std::getline(file1, line);
97.             if (!file1.good()) break;
98.
99.             std::stringstream iss(line);
100.            vector<string> rowOfFile;
```

```
        for (int col = 0; col < stop_times_cols; col++)
        {
            std::string val;
            std::getline(iss, val, ',');

            std::stringstream convertor(val);
            rowOfFile.push_back(convertor.str());

            if (!iss.good()) break;
        }
        parameters.push_back(rowOfFile);

    }
}

stringstream ss_pathToReadFile2;
ss_pathToReadFile2 << "Data folder/athens_metro/toFeedToModel/interfaceToModel/TTs1s2rt.txt";
string pathToReadFile2 = ss_pathToReadFile2.str();
std::ifstream file2(pathToReadFile2);

if (file2)
{
    for (int row = 0; row < stop_times_rows; row++)
    {
        std::string line;
        std::getline(file2, line);
        if (!file2.good()) break;

        std::stringstream iss(line);
        vector<string> rowOfFile;
        for (int col = 0; col < stop_times_cols; col++)
        {
            std::string val;
            std::getline(iss, val, ',');

            std::stringstream convertor(val);
            rowOfFile.push_back(convertor.str());

            if (!iss.good()) break;
        }
        TTs1s2rt.push_back(rowOfFile);
    }
}

stringstream ss_pathToReadFile3;
ss_pathToReadFile3 << "Data folder/athens_metro/toFeedToModel/interfaceToModel/objFun.txt";
string pathToReadFile3 = ss_pathToReadFile3.str();
std::ifstream file3(pathToReadFile3);

if (file3)
{
    for (int row = 0; row < stop_times_rows; row++)
    {
        std::string line;
        std::getline(file3, line);
        if (!file3.good()) break;

```

```
158.          std::stringstream iss(line);
159.          vector<string> rowOfFile;
160.          for (int col = 0; col < stop_times_cols; col++)
161.          {
162.             std::string val;
163.             std::getline(iss, val, ',');
164.
165.             std::stringstream convertor(val);
166.             rowOfFile.push_back(convertor.str());
167.
168.             if (!iss.good()) break;
169.          }
170.          objFun.push_back(rowOfFile);
171.       }
172.    }
173.
174.    /*for (int row = 0; row < objFun.size(); row++)
175.    {
176.    for (int col = 0; col < objFun[row].size(); col++)
177.    {
178.    cout << objFun[row][col] << " ";
179.    }
180.    cout << endl;
181.    }*/
182.
183.    //CPLEX parameters
184.    IloEnv      env;
185.    char        label[70];
186.    IloModel    model(env);
187.    IloCplex    Cplex(env);
188.
189.    //data of the problem
190.    std::vector<vector<vector<int>>> NSDTsrt;          //data for the problem/model
191.    std::vector<vector<vector<int>>> NSATsrt;          //data for the problem/model
192.
193.    std::vector<vector<int>> NSDTrt;
194.    std::vector<vector<int>> NSATrt;
195.
196.    vector<int> NSDTt;
197.    vector<int> NSATt;
198.
199.    //--------Construct Matrices----------------
200.    typedef IloArray<IloNumArray>        IloNumMatrix2x2;
201.    typedef IloArray<IloNumMatrix2x2>    IloNumMatrix3x3;
202.    typedef IloArray<IloNumMatrix3x3>    IloNumMatrix4x4;
203.
204.    typedef IloArray<IloNumVarArray>     IloNumVarMatrix2x2;
205.    typedef IloArray<IloNumVarMatrix2x2>  IloNumVarMatrix3x3;
206.    typedef IloArray<IloNumVarMatrix3x3>  IloNumVarMatrix4x4;
207.
208.    typedef IloArray<IloRangeArray>         IloRangeMatrix2x2;
209.    typedef IloArray<IloRangeMatrix2x2>     IloRangeMatrix3x3;
210.    typedef IloArray<IloRangeMatrix3x3>     IloRangeMatrix4x4;
211.
212.    //NSTOSsrt - Parameter
213.    std::vector<std::tuple<int, int, int, int>> NSTOSsrt;
214.
```

```
215.  for (int i = 0; i < decisionVars.size(); i++)
216.  {
217.     index_1 = stoi(decisionVars[i][0]);
218.     index_2 = stoi(decisionVars[i][1]);
219.     index_3 = stoi(decisionVars[i][2]);
220.     index_4 = stoi(decisionVars[i][3]);
221.     NSTOSsrt.push_back(std::make_tuple(index_1, index_2, index_3, index_4));
222.  }
223.
224.  //APrt - Parameter
225.  std::vector<std::tuple<int, int, double>> APrt; //we will see what we gonna do with dat
226.
227.  //RFrt - Parameter
228.  std::vector<std::tuple<int, int, double>> RFrt;
229.
230.  for (int j = 0; j < parameters.size(); j++)
231.  {
232.     index_1 = stoi(parameters[j][0]);
233.     index_2 = stoi(parameters[j][1]);
234.     quantity = stod(parameters[j][2]);
235.
236.     RFrt.push_back(std::make_tuple(index_1, index_2, quantity));
237.  }
238.
239.  //TTs1s2rt - Parameter
240.  std::vector<std::tuple<int, int, int, int, int>> parameter_TTs1s2rt;
241.
242.  for (int i = 0; i < TTs1s2rt.size(); i++)
243.  {
244.     index_1    = stoi(TTs1s2rt[i][0]);
245.     index_2    = stoi(TTs1s2rt[i][1]);
246.     index_3    = stoi(TTs1s2rt[i][2]);
247.     index_4    = stoi(TTs1s2rt[i][3]);
248.     quantity   = stoi(TTs1s2rt[i][4]);
249.
250.     parameter_TTs1s2rt.push_back(std::make_tuple(index_1, index_2, index_3, index_4, quantity));
251.  }
252.
253.  //TOSsrt - decision variable
254.
255.  std::vector<std::tuple<int, int, int, IloNumVar>> TOSsrt;
256.
257.  for (int i = 0; i < NSTOSsrt.size(); i++)
258.  {
259.     index_1 = get<0>(NSTOSsrt[i]);
260.     index_2 = get<1>(NSTOSsrt[i]);
261.     index_3 = get<2>(NSTOSsrt[i]);
262.     sprintf_s(label, "VAR_TOS(%d,%d,%d)", index_1, index_2, index_3);
263.     IloNumVar VAR(env, 0, 2880, ILOINT, label);
264.     TOSsrt.push_back(std::make_tuple(index_1, index_2, index_3, VAR));
265.  }
266.
267.  //Constraint 1 - Sindeseis ti metavliti TOSsrt me to NSTOS
268.
269.  std::vector<std::tuple<int, int, int, IloRange>> Constraint_1;
270.
271.  for (int i = 0; i < TOSsrt.size(); i++)
```

```
{
    index_1 = get<0>(TOSsrt[i]);
    index_2 = get<1>(TOSsrt[i]);
    index_3 = get<2>(TOSsrt[i]);

    int    NSTOS;
    double  parameter_RFrt;
    double  parameter_APrt = 1;

    for (int j = 0; j < NSTOSsrt.size(); j++)
    {
        int index_1_2 = get<0>(NSTOSsrt[i]);
        int index_2_2 = get<1>(NSTOSsrt[i]);
        int index_3_2 = get<2>(NSTOSsrt[i]);
        if (index_1 == index_1_2)
        {
            if (index_2 == index_2_2)
            {
                if (index_3 == index_3_2)
                {
                    NSTOS = get<3>(NSTOSsrt[i]);
                }
            }
        }
    }

    for (int j = 0; j < RFrt.size(); j++)
    {
        int index_2_2 = get<0>(RFrt[j]);
        int index_3_2 = get<1>(RFrt[j]);

        if (index_2 == index_2_2)
        {
            if (index_3 == index_3_2)
            {
                parameter_RFrt = get<2>(RFrt[j]);
            }
        }
    }

    sprintf_s(label, "Constraint 1: TOS(%d,%d,%d)", index_1, index_2, index_3);
    IloExpr expr(env, 0);
    expr = get<3>(TOSsrt[i]);
    double Con_LB = NSTOS - 45;// -(parameter_RFrt*parameter_APrt);
    double Con_UB = NSTOS + 45;// +(parameter_RFrt*parameter_APrt);
    IloRange Constraint(env, Con_LB, expr, Con_UB, label);
    model.add(Constraint);
    Constraint_1.push_back(std::make_tuple(index_1, index_2, index_3, Constraint));
    expr.end();
}

//Constraint 2

std::vector<std::tuple<int, int, int, int, IloRange>> Constraint_2;

for (int i = 0; i < parameter_TTs1s2rt.size(); i++)
{
```

```
329.      IloNumVar TOS_1;
330.      IloNumVar TOS_2;
331.
332.      index_1 = get<0>(parameter_TTs1s2rt[i]);
333.      index_2 = get<1>(parameter_TTs1s2rt[i]);
334.      index_3 = get<2>(parameter_TTs1s2rt[i]);
335.      index_4 = get<3>(parameter_TTs1s2rt[i]);
336.
337.      for (int j = 0; j < TOSsrt.size(); j++)
338.      {
339.         int index_1_2 = get<0>(TOSsrt[j]);
340.         int index_2_2 = get<1>(TOSsrt[j]);
341.         int index_3_2 = get<2>(TOSsrt[j]);
342.
343.         if (index_1 == index_1_2)
344.         {
345.            if (index_3 == index_2_2)
346.            {
347.               if (index_4 == index_3_2)
348.               {
349.                  TOS_1 = get<3>(TOSsrt[j]);
350.                  break;
351.               }
352.            }
353.         }
354.      }
355.
356.      for (int j = 0; j < TOSsrt.size(); j++)
357.      {
358.         int index_1_2 = get<0>(TOSsrt[j]);
359.         int index_2_2 = get<1>(TOSsrt[j]);
360.         int index_3_2 = get<2>(TOSsrt[j]);
361.
362.         if (index_2 == index_1_2)
363.         {
364.            if (index_3 == index_2_2)
365.            {
366.               if (index_4 == index_3_2)
367.               {
368.                  TOS_2 = get<3>(TOSsrt[j]);
369.                  break;
370.               }
371.            }
372.         }
373.      }
374.
375.      sprintf_s(label, "Constraint 2: TT(%d,%d,%d,%d)", index_1, index_2, index_3, index_4);
376.      IloExpr expr(env, 0);
377.      expr = TOS_2 - TOS_1;
378.      double Con_LB = get<4>(parameter_TTs1s2rt[i]);
379.      double Con_UB = get<4>(parameter_TTs1s2rt[i]);
380.      IloRange Constraint(env, Con_LB, expr, Con_UB, label);
381.      model.add(Constraint);
382.      Constraint_2.push_back(std::make_tuple(index_1, index_2, index_3, index_4, Constraint));
383.      expr.end();
384.   }
385.
```

```
386.  //Constraint 3
387.
388.  std::vector<std::tuple<int, int, int, int, int, int, IloRange>> Constraint_3;
389.
390.  for (int i = 0; i < objFun.size(); i++)
391.  {
392.     IloNumVar TOS_1;
393.     IloNumVar TOS_2;
394.
395.     index_1 = stoi(objFun[i][0]);
396.     index_2 = stoi(objFun[i][1]);
397.     index_3 = stoi(objFun[i][2]);
398.     index_4 = stoi(objFun[i][3]);
399.     index_5 = stoi(objFun[i][4]);
400.     index_6 = stoi(objFun[i][5]);
401.
402.     for (int j = 0; j < TOSsrt.size(); j++)
403.     {
404.        int index_1_2 = get<0>(TOSsrt[j]);
405.        int index_2_2 = get<1>(TOSsrt[j]);
406.        int index_3_2 = get<2>(TOSsrt[j]);
407.
408.        if (index_1 == index_1_2)
409.        {
410.           if (index_2 == index_2_2)
411.           {
412.              if (index_3 == index_3_2)
413.              {
414.                 TOS_2 = get<3>(TOSsrt[j]);
415.                 break;
416.              }
417.           }
418.        }
419.     }
420.
421.     for (int j = 0; j < TOSsrt.size(); j++)
422.     {
423.        int index_1_2 = get<0>(TOSsrt[j]);
424.        int index_2_2 = get<1>(TOSsrt[j]);
425.        int index_3_2 = get<2>(TOSsrt[j]);
426.
427.        if (index_4 == index_1_2)
428.        {
429.           if (index_5 == index_2_2)
430.           {
431.              if (index_6 == index_3_2)
432.              {
433.                 TOS_1 = get<3>(TOSsrt[j]);
434.                 break;
435.              }
436.           }
437.        }
438.     }
439.
440.     IloExpr expr(env, 0);
441.     expr = (TOS_2 - TOS_1);
```

```
442.       sprintf_s(label, "Constraint 3: TOS(%d,%d,%d) -
       TOS(%d,%d,%d)", index_4, index_5, index_6, index_1, index_2, index_3);
443.       double Con_LB = 2;
444.       double Con_UB = IloInfinity;
445.       IloRange Constraint(env, Con_LB, expr, Con_UB, label);
446.       model.add(Constraint);
447.       Constraint_3.push_back(std::make_tuple(index_1, index_2, index_3, index_4, index_5, index_6, Constra
    int));
448.       expr.end();
449.    }
450.
451.    //Objective function
452.    IloExpr expr(env, 0);
453.
454.    for (int i = 0; i < objFun.size(); i++)
455.    {
456.       IloNumVar TOS_1;
457.       IloNumVar TOS_2;
458.
459.       index_1 = stoi(objFun[i][0]);
460.       index_2 = stoi(objFun[i][1]);
461.       index_3 = stoi(objFun[i][2]);
462.       index_4 = stoi(objFun[i][3]);
463.       index_5 = stoi(objFun[i][4]);
464.       index_6 = stoi(objFun[i][5]);
465.
466.       for (int j = 0; j < TOSsrt.size(); j++)
467.       {
468.          int index_1_2 = get<0>(TOSsrt[j]);
469.          int index_2_2 = get<1>(TOSsrt[j]);
470.          int index_3_2 = get<2>(TOSsrt[j]);
471.
472.          if (index_1 == index_1_2)
473.          {
474.             if (index_2 == index_2_2)
475.             {
476.                if (index_3 == index_3_2)
477.                {
478.                   TOS_2 = get<3>(TOSsrt[j]);
479.                   break;
480.                }
481.             }
482.          }
483.       }
484.
485.       for (int j = 0; j < TOSsrt.size(); j++)
486.       {
487.          int index_1_2 = get<0>(TOSsrt[j]);
488.          int index_2_2 = get<1>(TOSsrt[j]);
489.          int index_3_2 = get<2>(TOSsrt[j]);
490.
491.          if (index_4 == index_1_2)
492.          {
493.             if (index_5 == index_2_2)
494.             {
495.                if (index_6 == index_3_2)
496.                {
```

```
                    TOS_1 = get<3>(TOSsrt[j]);
                    break;
                }
            }
        }
    }

    expr += (TOS_2 - TOS_1);
  }

  model.add(IloMinimize(env, expr));
  expr.end();

  //Solution etc

  Cplex.extract(model);
  Cplex.exportModel("athensMetroCaseStudyAllTripsPerRoute_approach_1.lp");

  Cplex.solve();


  //cout << "Waiting time before optimization was: " << 78923 << endl;
  cout << "Waiting time after optimization is (Value of the objective function): " << Cplex.getObjValue() <<
endl;

  /*sprintf_s(label, "");
  for (int i = 0; i < TOSsrt.size(); i++)
  {
    index_1 = get<0>(TOSsrt[i]);
    index_2 = get<1>(TOSsrt[i]);
    index_3 = get<2>(TOSsrt[i]);

    int    NSTOS;

    for (int j = 0; j < NSTOSsrt.size(); j++)
    {
      int index_1_2 = get<0>(NSTOSsrt[i]);
      int index_2_2 = get<1>(NSTOSsrt[i]);
      int index_3_2 = get<2>(NSTOSsrt[i]);
      if (index_1 == index_1_2)
      {
        if (index_2 == index_2_2)
        {
          if (index_3 == index_3_2)
          {
            NSTOS = get<3>(NSTOSsrt[i]);
          }
        }
      }
    }

    cout << get<3>(TOSsrt[i]) << "Before optimization: " << NSTOS << " And after optimization: " << Cplex
.getValue(get<3>(TOSsrt[i])) << endl;

  }*/
```

```
552.     return 1;
553. 
554. }
555. 
556. int athensMetroAllTrips_2()
557. {
558. 
559.     return 1;
560. }
```

8.5 Utilities – Code snippets/functions used for calculations of time windows and distances

8.5.1 conversionFromHHMMSStoContinuousTime – Used to handle event times

```
#include "conversionFromHHMMSStoContinuousTimeRepresentation.h"
#include <sstream>
#include <fstream>
#include <string>
#include <iostream>
#include <tuple>

using namespace std;

std::tuple<int, int, int> createTupleToReturn(int token_1, int token_2, int token_3)
{
    return std::make_tuple(token_1, token_2, token_3);
}

int conversionToContinuousTime(std::string time_initial_format)
{

    std::string delimiter = ":";

    //std::string token_1 = str.substr(0, str.find(delimiter));
    std::string token_1 = time_initial_format.substr(0, 2);
    std::string token_2 = time_initial_format.substr(3, 2);
    std::string token_3 = time_initial_format.substr(6, 2);

    //cout << token_1;
    //cout << token_2;
    //cout << token_3;

    int minutes = std::stoi(token_1) * 60 + std::stoi(token_2);

    //auto resultTuple = createTupleToReturn(std::stoi(token_1), std::stoi(token_2), std::stoi(token_3));

    return minutes;
}
```

8.5.2 findRouteIDByTripID – Used to identify correlations between relational IDs of GTFS

```
#include <ilcplex/ilocplex.h>
#include <time.h>
#include <vector>
#include <sstream>
#include <fstream>
#include <utility>
#include <string>
#include <iostream>
#include <chrono>
#include <map>
#include <thread>
#include <list>
#include <stdio.h>
#include <array>
#include "interfaceToModel_parameters.h"
#include "identifyIndex.h"
#include "conversionFromHHMMSStoContinuousTimeRepresentation.h"
#include "findRouteIDbyTripID.h"

using namespace std;

string findRouteIDByTripID(std::string tripID)
{
    int const trips_cols = 7;
    int const trips_rows = 3636;

    string tripID_row;
    string routeID = "";

    string trips_data[trips_rows][trips_cols];
    std::ifstream file2("Data folder/athens_metro/trips.txt");

    for (int row = 0; row < trips_rows; ++row)
    {
        std::string line;
        std::getline(file2, line);

        std::stringstream iss(line);

        for (int col = 0; col < trips_cols; ++col)
        {
            std::string val;
            std::getline(iss, val, ',');

            std::stringstream convertor(val);
            convertor >> trips_data[row][col];

            if (!iss.good())
                break;
        }
        if (!file2.good())
            break;
```

```
53.     }
54.
55.     for (int row = 0; row < trips_rows; ++row)
56.     {
57.
58.         tripID_row = trips_data[row][2];
59.
60.         if (tripID_row == tripID)
61.         {
62.             routeID = trips_data[row][0];
63.             break;
64.         }
65.     }
66.
67.     return routeID;
68.
69.
70.
71.
72. }
```

8.5.3 haversineDistance – Calcualtion of distance between two points on map according to the Haversine formula

```
#include "havershineDistance.h"
#include <math.h>
#include <cmath>
#define earthRadiusKm 6371.0
#define M_PI 3.14159

using namespace std;

// This function converts decimal degrees to radians
double deg2rad(double deg) {
  return (deg * M_PI / 180);
}

//  This function converts radians to decimal degrees
double rad2deg(double rad) {
  return (rad * 180 / M_PI);
}

/**
* Returns the distance between two points on the Earth.
* Direct translation from http://en.wikipedia.org/wiki/Haversine_formula
* @param lat1d Latitude of the first point in degrees
* @param lon1d Longitude of the first point in degrees
* @param lat2d Latitude of the second point in degrees
* @param lon2d Longitude of the second point in degrees
* @return The distance between the two points in kilometers
*/

double distanceEarth(double lat1d, double lon1d, double lat2d, double lon2d) {
  double lat1r, lon1r, lat2r, lon2r, u, v;
  lat1r = deg2rad(lat1d);
  lon1r = deg2rad(lon1d);
  lat2r = deg2rad(lat2d);
  lon2r = deg2rad(lon2d);
  u = sin((lat2r - lat1r) / 2);
  v = sin((lon2r - lon1r) / 2);
  return 2.0 * earthRadiusKm * asin(sqrt(u * u + cos(lat1r) * cos(lat2r) * v * v));
}
```

** End of document/thesis **