Decision support systems for scheduling and routing problems in a home care business

Isabel Méndez Fernández

Doctoral Thesis UDC/2023
Advisors: Silvia María Lorenzo Freire, Ángel Manuel González Rueda
Doctoral Program in Statistics and Operations Research

UNIVERSIDADE DA CORUÑA

Decision support systems for scheduling and routing problems in a home care business

Isabel Méndez Fernández

Doctoral Thesis UDC/2023
Advisors: Silvia María Lorenzo Freire, Ángel Manuel González Rueda
Doctoral Program in Statistics and Operations Research
Department of Mathematics
Faculty of Computer Science

UNIVERSIDADE DA CORUÑA

The undersigned, Silvia Lorenzo Freire and Ángel Manuel González Rueda, certify that they are the advisors of the Doctoral Thesis with International Doctorate Mention entitled “Decision support systems for scheduling and routing problems in a home care business”, developed by Isabel Méndez Fernández at the University of A Coruña (Department of Mathematics), as part of the interuniversity doctoral program (UDC, USC and UVigo) of Statistics and Operational Research, and hereby give they consent to the author to proceed with the thesis presentation and the subsequent defense.

A Coruña, ________________.

Advisors:

Silvia Lorenzo Freire   Ángel Manuel González Rueda

PhD candidate:

Isabel Méndez Fernández

Agradecimientos

En primer lugar agradecer a mis directores de tesis, Ángel Manuel González Rueda y Silvia Lorenzo Freire. Gracias por haberme guiado durante estos años, vuestra supervisión y apoyo han sido claves para que el día de hoy haya llegado. También a Ignacio García Jurado, porque gracias a él comenzó toda esta aventura. Por otro lado, agradecer a Isabel Correia, por su apoyo durante mi gratificante estancia en Lisboa. Al tribunal de seguimiento, Leticia Lorenzo Picado, Rubén Fernández Casal y Julio González Díaz, por sus aportaciones que contribuyeron a la mejora de esta tesis. Gracias también a Federico Perea Rojas-Marcos, María Luisa Carpente Rodriguez y María Isabel Azevedo Rodrigues Gomes, por formar parte del tribunal de defensa.

Quiero agradecer a todas las personas que me han estado a mi lado durante estos años. A Marina por haber estado siempre ahí, animándome para seguir adelante. A las personas que han aparecido para quedarse: Bea, Eva, Silvia, Luis, Juan, Jonatan y Rebeca. Hoy se cierra una etapa de la que imperturbablemente habéis formado parte y que, sin duda, habéis hecho mejor.

A mis padres, Rafael y María Jesús, por convertirme en quien soy y acompañarme a lo largo de este camino. A mi hermana, Carmen, por ser siempre mi faro y por saber cómo gestionar mis momentos de agobio.

Funding

The PhD candidate's research was sponsored by the Spanish Grant for Predoctoral Research Trainees RD 103/2019.

This research was funded by MICINN/AEI/10.13039/501100011033/ and ERDF/EU through R+D+I project grants MTM2017-87197-C3-1-P and PID2021-124030NB-C31 and by Consellería de Cultura, Educación e Universidades, Xunta de Galicia (Grupos de Referencia Competitiva ED431C-2020/14 and Centro de Investigación del Sistema universitario de Galicia ED431G-2019/01).

Abstract

Home care services aim to help elderly, sick or dependent people in maintain their quality of life without having to leave their homes. This type of problem is denoted as Home Care Scheduling Problem (HCSP) and the goal is to obtain the routes that the company's caregivers must follow, as well as the timeframe in which each service should be carried out. That is, a HCSP can be seen as a routing and scheduling problem.

This thesis studies a real HCSP presented by a company called Mayores. First, the original problem proposed by the company is presented, which consists in updating the weekly schedules of the caregivers in order to solve a set of possible incidents. This problem is tackled using the Simulated Annealing method embedded inside a custom heuristic algorithm. Second, a more general version of the problem is described, which aims to obtain the best possible schedules of the caregivers from scratch, considering two objectives: the welfare of the users and the cost of the schedule. The problem is solved considering three approaches: two hierarchical ones (prioritizing the welfare or prioritizing the cost) and a biobjective one. Finally, a computational study is presented in order to evaluate the resolution approaches, using instances from the literature and real data from the company.

Resumen

Los servicios de atención a domicilio tienen como objetivo ayudar a personas mayores o dependientes a mantener o mejorar su calidad de vida sin necesidad de abandonar sus hogares. Esta clase de problema se denomina Home Care Scheduling Problem (HCSP) y su objetivo es obtener las rutas que deben seguir las auxiliares de la empresa, así como los horarios en los que debe realizarse cada servicio. Es decir, los HCSP pueden ser considerados como problemas de planificación de rutas y horarios.

En esta tesis se estudia un HCSP real presentado por una empresa llamada Mayores. En primer lugar, se presenta el problema original propuesto por la empresa, que consiste en actualizar los horarios semanales de las auxiliares con el fin de resolver un conjunto de incidencias. En segundo lugar, se describe una versión más general del problema, cuyo objetivo es obtener, partiendo desde cero, las mejores planificaciones para las auxiliares, considerando dos objetivos: el bienestar de los usuarios y el coste de la planificación. El problema se resuelve considerando tres enfoques: dos jerárquicos (dando prioridad al bienestar o al coste) y uno biobjetivo. Por último, se presenta un estudio computacional para evaluar los enfoques de resolución propuestos, utilizando instancias de la literatura y datos reales proporcionados por la empresa.

Resumo

Os servizos de atención a domicilio teñen como obxectivo axudar a persoas maiores ou dependentes a manter ou mellorar a súa calidade de vida sen necesidade de abandonar os seus fogares. Esta clase de problema denomínase Home Care Scheduling Problem (HCSP) e o seu obxectivo é obter as rutas que deben seguir as auxiliares da empresa, así como os horarios nos que debe realizarse cada servizo. É dicir, os HCSP poden ser considerados como problemas de planificación de rutas e horarios.

Nesta tese estúdase un HCSP real presentado por unha empresa chamada Mayores. En primeiro lugar, preséntase o problema orixinal proposto pola empresa, que consiste en actualizar os horarios semanais das auxiliares co fin de resolver un conxunto de incidencias. En segundo lugar, descríbese unha versión máis xeral do problema, cuxo obxectivo é obter, partindo desde cero, as mellores planificacións para as auxiliares, considerando dous obxectivos: o benestar dos usuarios e o custo da planificación. O problema resólvese considerando tres enfoques: dous xerárquicos (dando prioridade ao benestar ou ao custo) e un biobxetivo. Por último, preséntase un estudo computacional para avaliar os enfoques de resolución propostos, empregando instancias da literatura e datos reais proporcionados pola empresa.

Contents

List of Figures

List of Tables

List of Algorithms

List of most used abbreviations

Abbreviation	Meaning
ALNS	Adaptive Large Neighborhood Search
ALNS_CPSAT	Combination of the ALNS method with CPSAT
ALSN_CW	Combination of the ALNS method with Algorithm 5.1
ALSN_WC	Combination of the ALNS method with Algorithm 4.1
AUGMECON2	Improved version of the Augmented Epsilon Constraint method
BIALNS	Biobjective metaheuristic algorithm
COP	Constraint Optimization Problem
CP	Constraint Programming
CPSAT	CP solver included in the Google OR Tools
CSP	Constraint Satisfaction Problems
CV	Coverage indicator
EPS	Epsilon indicator
GD	Generational Distance indicator
GIRO	Generación, Gestión e Integración de Rutas en OLAP
HCSP	Home Care Scheduling Problem
HTW	Hard Time Window
IGD	Inverted Generational Distance indicator
LBD	Database Lab
MILP	Mixed Integer Linear Programming
NDP	Number of non dominated points
OLAP	On-Line Analytical Processing
PT	Pay-off Table of a multiobjective problem
RPD	Relative Percentage Deviation
SA	Simulated Annealing
STW	Soft Time Window
TP	Total number of points
VRP	Vehicle Routing Problem

Introduction

This work arose from the project Innterconecta GIRO - Generación, Gestión e Integración de Rutas en OLAP (ITC-20151247), which involved a group of companies from different sectors, but with similar challenges in terms of optimizing their organizational processes. The activity of these companies required the mobility of their employees, i.e. their workers should travel to complete tasks, or provide services, in different locations. Another common characteristic to all these companies is the necessity for a continuous adjustment of their schedules as a result of a series of contingencies: the appearance/disappearance of clients, the absence of a worker, the incompatibility of tasks and schedules, etc. Therefore, the logistic problems of the companies were all related to task planning, scheduling and routing. Thus, our task in the project was to explore the integration of operational research and mathematical optimization techniques into the scheduling and route planning processes of the participating companies. The final objective of the project was to design automatic tools to support decision-making for the companies. So, once the situation of the workers was known, the tools would be able to provide a work plan for each of them and adapt it according to the contingencies that might arise.

The following six companies were involved in the project:

Gesuga (gesuga.com) This company provides a collection service for meat by-products not intended for human consumption. It has a fleet of vehicles, located at several plants, which follow the daily schedule established by the company's routing department. These schedules have to cover the daily collection orders while satisfying a number of constraints. The company aims to improve the planning of these routes, minimizing the distances traveled by the vehicles and their fuel consumption. In addition, it is also necessary to deal with unforeseen contingencies, such as the arrival of a new collection order. It means that the initial routes should be adapted to the requirements that come in during the day.

Biogas Fuel Cell This company is specialized in the management and processing of organic waste into biogas. The company has its own biogas plant with a reception area to receive waste. The waste is transported both by external contractors and by the company's own vehicles, so it is necessary to coordinate the arrivals of both types of vehicles. It results in two problems: organizing the reception of the waste at the plant and establishing the routes for their own vehicles. These routes aim to minimize the travel distance and meet the schedules given by the first problem.

Mayores (mayores.es) This company provides home care services to dependent people and their families. For this purpose, a team of specialized staff visit the users in their own homes. Therefore, the company needs to determine the routes to be followed by its caregivers, with the objectives of maintaining patient schedules and reducing travel time. An important feature of this problem is that routes should encourage the continuity of care of the caregivers that attended the users in the past. The proposed tool must be able to obtain a schedule and route planning that dynamically adapts to all kinds of contingencies: requests for scheduling changes, services cancelled by some users, modification of visits duration, etc.

Grupo On (seguridadon.es) This company is specialized in offering security systems involving the installation of alarm devices, both for companies and residential homes. A team of technicians are constantly traveling for the installation of alarms and their maintenance, which includes periodic inspections and repair work. The company faces scheduling problems due to the need of organizing the routes of its technicians in order to keep the appointments with each client. The objective is to reduce travel times and maximize the priority of the clients, while satisfying customer availability.

Mugatra (mugatra.es) This company offers a set of services to its clients that provide total coverage for the workplace safety and risk prevention. The most important service is workplace health monitoring, whose objective is to carry out periodic evaluations of the client's staff, in order to assess whether their health condition is satisfactory to carry out their duties in the company. The problem presented by this company is a combination of three different planning problems:

- Setting the service dates for each client.
- Deciding whether to use mobile units or to rent rooms, which entails scheduling the shipment of medical equipment.
- Scheduling the examinations for all employees of the client.

The objective is to minimize the travel times between clients and maximize the number of clients that can be attended at the same location.

Taprega (taprega.com) The company offers its clients services in the field of workplace safety, specializing in the prevention of work-related risks. There are two kind of workers in the company: technicians and sales representatives. The technicians are in charge of carrying out customer inspections. The sales representatives visit customers, or potential customers, with the aim of advertising the company and formalizing new contracts. There are two planning problems in the development of the activity of the company: on one hand, the planning of the agenda of its technicians and, on the other hand, the planning of its sales representatives. Regarding the problem of scheduling the technicians routes, the objectives are to minimize the travel times and maximize the urgency of the visits carried out. In the sales representatives issue, the objective is to minimize travel times. Furthermore, in both problems a periodicity in the visits must be taken into account.

This thesis focuses on the problem presented by Mayores, the home care company.

Mayores' company

Mayores has been providing home care services in the city of A Coruña and its neighboring municipalities (Abegondo, Arteixo, Cambre, Carral, Culleredo or Sada) since 1997. Figure 1 illustrates the web page of Mayores, showing the different services they provide (homecare, day care center for the elderly and nursing home), as well as a general description of the company.

Home care is a service that allows elderly and/or dependent people to continue living at home despite being in a situation of dependency or in need of assistance with different day-to-day tasks. This type of service allows users, and their families, to improve their quality of life by providing domestic and individual care services guaranteeing that users can continue to live at home in the best possible conditions.

CUIDADO Y ATENCIÓN DE PERSONAS MAYORES

Mayores atiende desde 1997 a las personas dependientes y a sus familias, en todo aquello que precisen, tanto en sus domicilios como en nuestro Centro de Día. Nuestro personal está en constante preparación y formación para dotarles de la mejor cualificación profesional.

Figure 1: Mayores web page.

The main goals of home care services are:

- To improve the quality of live of its users.
- To encourage the acquisition of skills that allow a more independent development in daily life.
- To support the user's family members in their care responsibilities.
- To avoid, or delay, the transfer of the user to a nursing home.

The users are people who need care or help in carrying out certain tasks and come to the company to improve their quality of life. Users demand a set of services which will have to be attended by the company's employees in the user's homes. During these services the caregivers must carry out certain tasks, that can be domestic, personal or heath related. Some of these activities must be performed at a specific range of times, for example getting up and going to bed must be performed in the morning and evening, respectively. Other activities can be carried out at any time of the day, such as dusting or doing the laundry. For each one of the services, the following information must be specified: its duration, the time frame it belongs to, the day of the week it must be carried out and a time window.

Figure 2 represents the number of services, by day, carried out by the company during 60 weeks, from November 2016 to December 2017. Note that the number of services during the weekends (roughly between 25 and 75 services per day) is much lower than on the other days (roughly between 125 and 225 services per day). This happens because, in many cases, the family members of the users take care of them at the weekends, so that they only need to be attended by the company from Monday to Friday. Another interesting feature that can be observed is that some days (Monday to Friday) the number of services drops drastically. These days correspond to holidays, and behave similarly to weekends, because the families of the users can take care of them during such days. For example, the first of May (Monday 05/01/17) was a holiday in Spain, so during that day the number of services required by the users was lower.

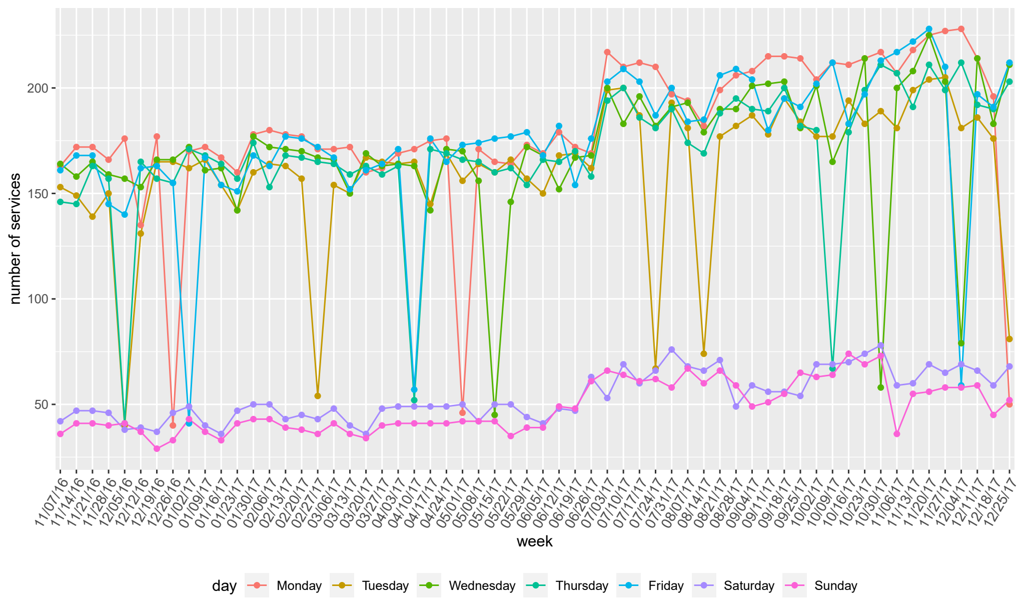


Figure 2: Services attended by the company from November 16 to December 17.

The caregivers[1] are the company's employees, who are in charge of visiting users' homes in order to carry out the assigned tasks.

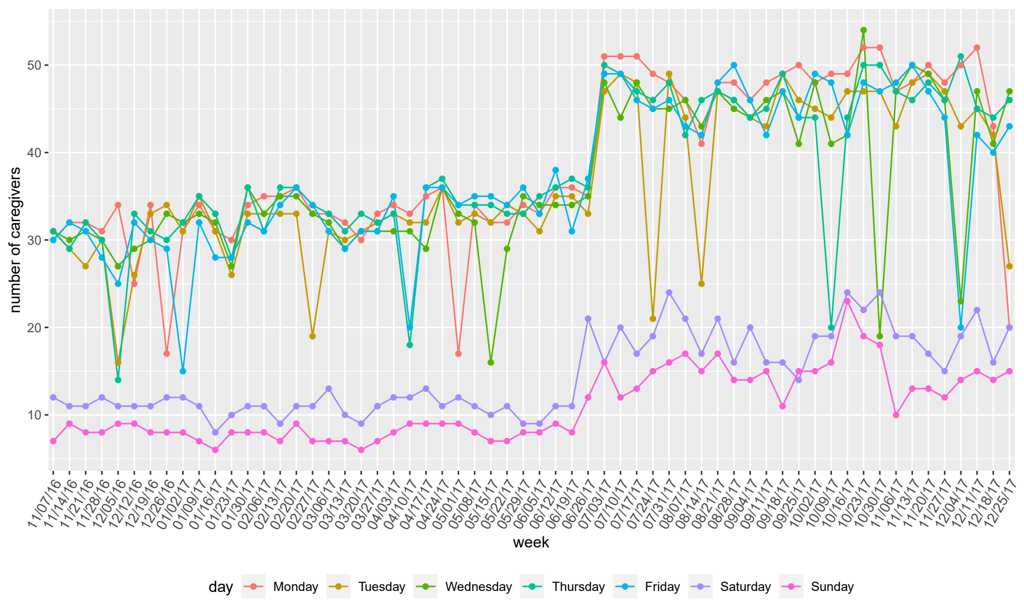


Figure 3: Number of employed caregivers from November 16 to December 17.

Figure 3 presents the weekly number of caregivers employed by the company from November 2016 to December 2017. It can be seen that the behavior is similar to the one observed with the

[1]Notice that the employees of the company are women. Therefore, the feminine form will be used throughout the work to refer to them.

number of services. Naturally, if users require fewer services, then the number of caregivers needed to perform them will also be lower. Therefore, the number of employed caregivers is smaller during the weekends and on holidays.

The caregivers have contracts that specify the number of hours they should work during the week. The following elements are considered as working time:

- The time spent at the users' homes carrying out services.
- The time spent traveling between services.
- All the breaks between services (free time) they may have during the day, with the exception of the largest one if it lasts two or more hours.

Travel times can vary depending on the region of activity. In urban areas users are more concentrated so caregivers can often travel on foot or using public transport. The contrary happens in rural areas, where users are generally widely distributed and the caregivers are usually required to use their own vehicle. It is worth noting that the company is only paid for the hours that the caregivers are attending the users. For this reason, the travel and free times of the caregivers can be seen as financial losses for the company.

The geographical distribution of the 291 users, that the company attends in a certain region, is presented in Figure 4. It can be seen that the region is divided into two parts: the urban center and a rural area. In the urban center (surrounded by the frame) there are more services over a small area, which reduces travel times. In the rural area there are fewer users scattered over a large area, which results in longer travel times.

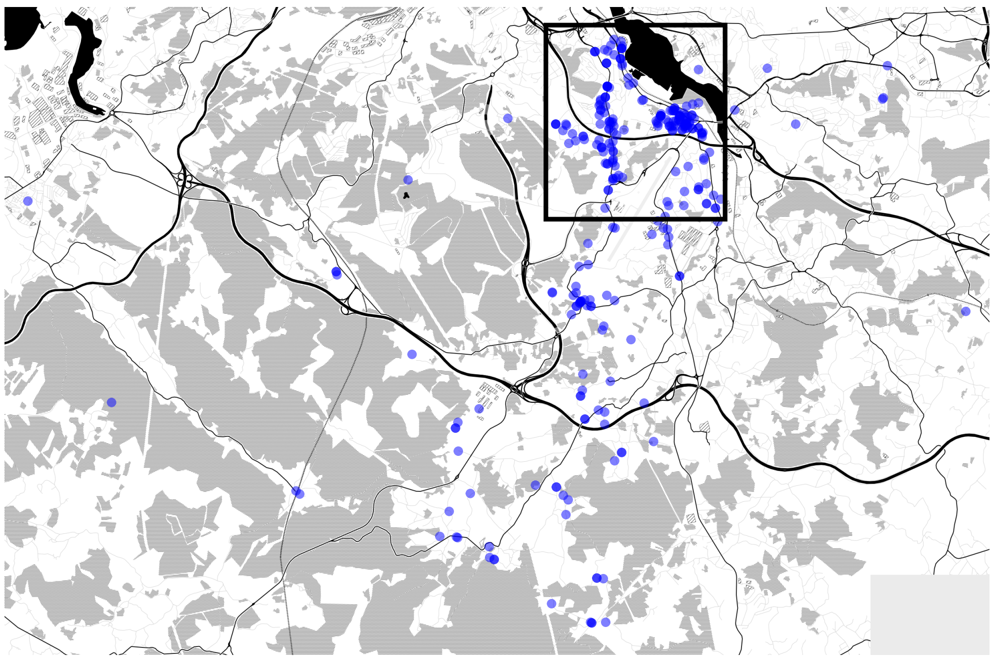


Figure 4: Geographical location of users.

Figure 5 shows the weekly working time (in hours) of a caregiver from November 2016 to December 2017. It can be seen that, in general, it is slightly affected by the holidays. For example, during the first week of May (the one that starts on 05/01/17) the caregiver worked fewer hours than during the one before. Which happened because on Monday, which was holiday, the worked time of the caregiver was lower (see Figure 6).

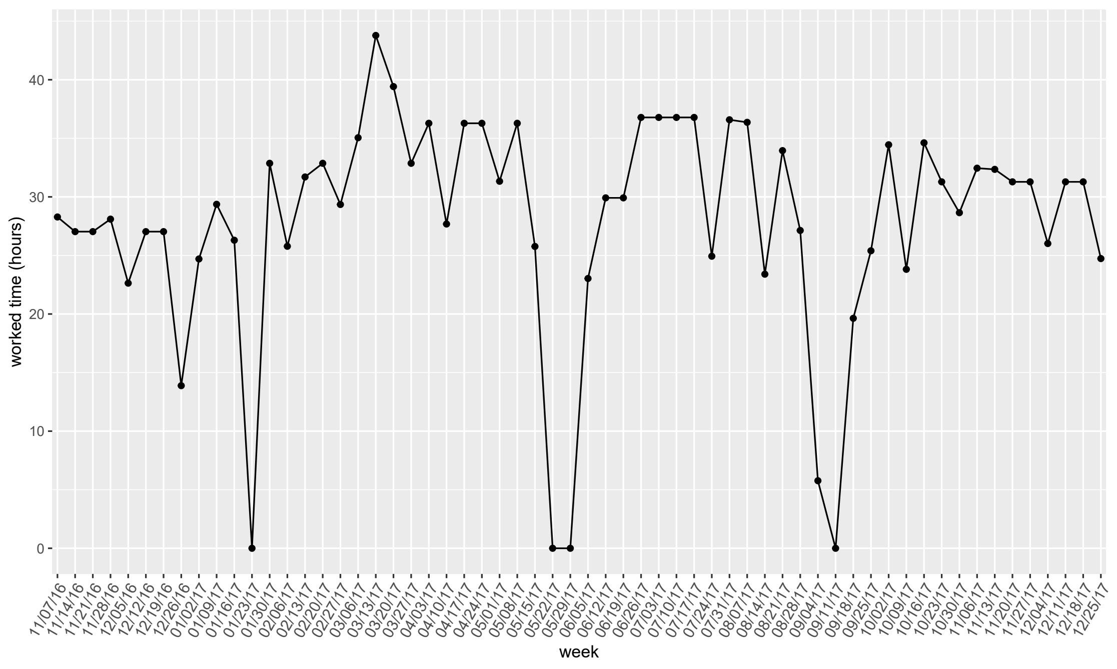
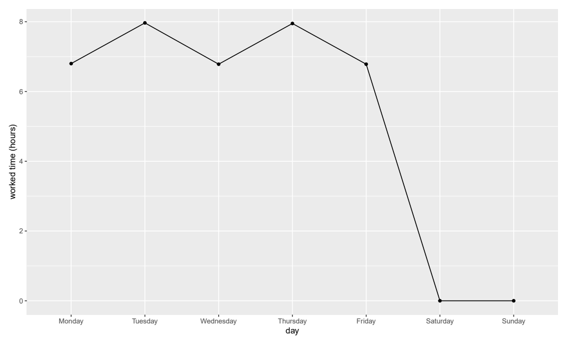
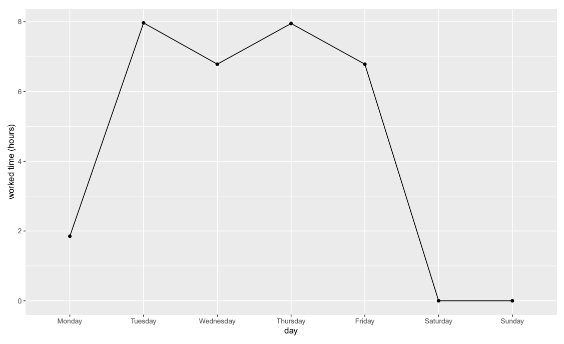


Figure 5: Working time of a caregiver.

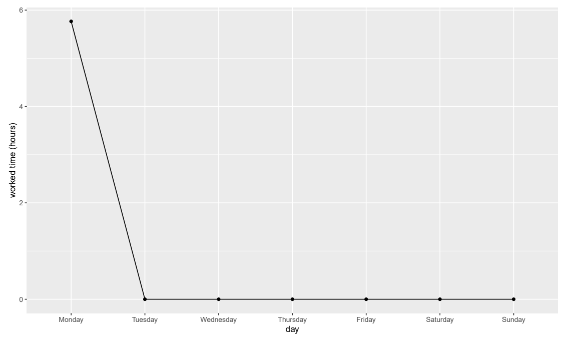


(a) Week that starts on 04/24/17.

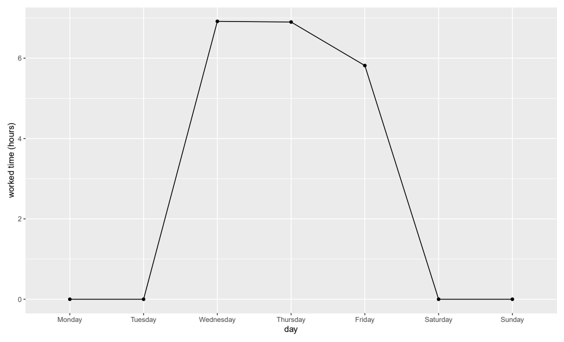


(b) Week that starts on 05/01/17.

Figure 6: Daily working hours of a caregiver.



(a) Week that starts on 09/04/17.

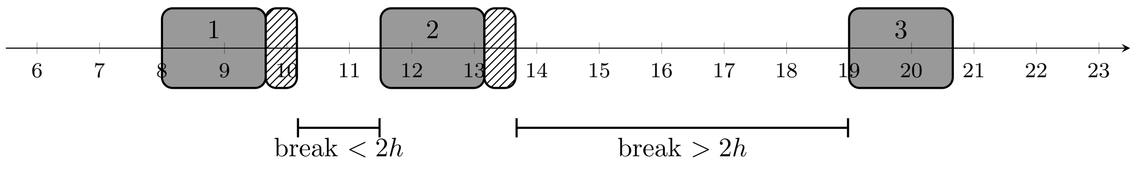


(b) Week that starts on 09/18/17.

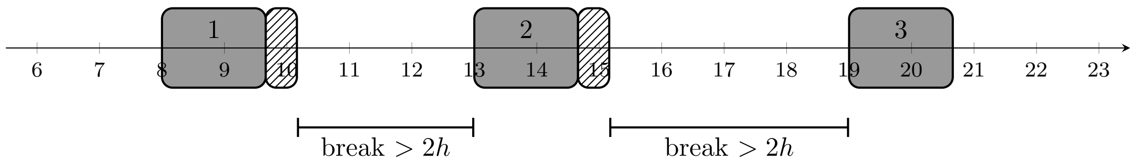
Figure 7: Daily working hours of a caregiver (vacation).

The effect that the holidays have on the worked time can be more difficult to appreciate because, in some cases, the caregiver worked less hours during a week for some other reasons (such as having a free day, vacations or fewer services to carry out). For example, analyzing the working time of the caregiver during September in Figure 5, it can be deduced that she did not work on the week that starts on 09/11/17 and she worked less than usual in the weeks before and after it. This is because the caregiver was on vacation between the 5th and 18th of September, which can be seen in Figure 7.

In terms of breaks, all of them will be considered as worked time with just one exception: in case the largest break of the working day has a duration greater than or equal to two hours it will not be paid.



(a) The only break larger than 2 hours is not considered as working time.



(b) The largest break is not considered as working time.

Figure 8: Types of breaks.

Thus, in the schedule of Figure 8a, where the services duration is 100 minutes (represented with dark blocks) and the travel time between each pair of services is 30 minutes (see hatched blocks), only the largest break (with a duration of 5 hours and 20 minutes) will be discounted. Then, the total worked time will be 7 hours and 20 minutes. In the schedule of Figure 8b, even though it has two breaks longer than 2 hours, only the second one (with a duration of 3 hours and 50 minutes) will not be paid. This means that the working time will be 8 hours and 50 minutes. Therefore, Figure 8 reveals the importance of correctly scheduling the largest break of a route.

The company employs a set of supervisors, who have been assigned a number of users and caregivers. The assignment of users is made taking into account their location and it cannot be modified, because the supervisors are in contact with the users' families (they write reports and monitor the evolution of the users).

The company considers that the consistency of the assignment of users to caregivers is very important, to the extent that it is preferable to change the schedule of a visit so that the caregivers who carries it out can always be the same one. Sometimes it is even necessary for a user to be attended by more than one caregiver. So the past assignments, and the preferences or problems that may have arisen between users and caregivers, are used to obtain a list of affinity levels. The affinity establishes a compatibility level between users and caregivers, being these levels defined as follows:

Level 0. The user should not be attended by this caregiver under any circumstances. This situation can happen because the caregiver is not qualified to attend the user or because there has been some kind of incident between them.

Level 1. The caregiver could attend the user only if there is no other option. It could happen when the caregiver received some mild complaints from the user.

Level 2. The caregiver could attend the user although the degree of compatibility between user and caregiver has not been established yet. In this level, the user does not require any kind of special characteristics from the caregivers.

Level 3. The caregiver has not attended the user yet but, due to the characteristics of the caregiver, the supervisor thinks that she could be a very good candidate to carry out the services.

Level 4. The user was successfully attended by the caregiver in the past, either sporadically or continuously.

Level 5. The caregiver is already attending the user, on a continuous basis, in a satisfactory manner.

In the company, supervisors are responsible of manually organizing the work plans of all the caregivers assigned to them. That is, they assign each service required by a user, during the week, to a caregiver and establish the specific times at which these services will start.

Related literature

The problem under study is known as Home Care Scheduling Problem (from now on HCSP). This kind of problem deals with real life situations where some users require home assistance services and they should be attended by a set of caregivers throughout the week. Thus, the goal of an HCSP is to design the routes and schedules for caregivers, indicating the services to carry out, in which order and at what time. Schedules should also satisfy all the requirements imposed by both, the company and the users, while pursuing a set of objectives.

Due to the increase demand of home care services in recent years, the literature on this type of problems has grown significantly. Detailed surveys on HCSP can be found in Cissé *et al.* (2017), Fikar & Hirsch (2017), Di Mascolo *et al.* (2021) and Euchi *et al.* (2022). Some of the most common characteristics discussed in these surveys will be described in this section.

Numerous HCSPs are inspired by real applications. For example, in Akjiratikarl *et al.* (2007) a real situation arising in the UK is studied, in collaboration with The Welsh Systems Consortium, a partnership between seven local government authorities in Wales. Grenouilleau *et al.* (2019) work with Alayacare, a start-up sited in Montreal (Canada) that is developing a decision support system for home health care companies. In Habibnejad-Ledari *et al.* (2019) the authors study the problem of Dam homecare medical center, a non-governmental organization in Iran. The problem studied in Luna *et al.* (2018) was developed in collaboration with EULEN, an international company offering cleaning, health, and logistics services (among others). In Malagodi *et al.* (2021) the problem is inspired by a real home care provider in the USA. Maya Duque *et al.* (2015) explore home care services in Belgium, considering the case of Landelijke Thuiszorg, an organization that provides home care services in four Belgian regions. The research of Mosquera *et al.* (2019) was motivated by Flemish (Belgium) home care companies that realized that their manual scheduling procedures were becoming obsolete. The problem presented in Rest & Hirsch (2016) was based on a collaborative project with the Austrian Red Cross, which is one of the leading home care service providers in Austria.

Reference	HTW	SKILL	NUMC	MIND	MAXD	FREQ	SYNC
Bard *et al.* (2014)	x						
Akjiratikarl *et al.* (2007)	x						
Bertels & Fahle (2006)	x	x					
Braekers *et al.* (2016)	x						
Erdem & Bulkan (2017)	x	x					
Chaieb *et al.* (2020)	x	x					
Garaix *et al.* (2018)	x	x	x				
Grenouilleau *et al.* (2019)	x	x					
Rest & Hirsch (2016)	x	x	x				
Kergosien *et al.* (2009)	x	x					x
Liu *et al.* (2017)	x	x					
Riazi *et al.* (2019)	x	x					
Mankowska *et al.* (2014)	x	x					
Mosquera *et al.* (2019)	x	x		x	x	x	
Bachouch *et al.* (2011)	x	x	x				
Nickel *et al.* (2012)	x	x				x	
Haddadene *et al.* (2016)	x	x					x
Ait Haddadene *et al.* (2019)	x	x					x
Chaieb & Ben Sassi (2021)	x	x					
Decerle *et al.* (2021)	x	x					x
Euchi *et al.* (2020)	x						
Kandakoglu *et al.* (2020)	x						
Frifita & Masmoudi (2020)	x						x
Grenouilleau *et al.* (2020)	x	x					x
Khodabandeh *et al.* (2021)	x	x					
Lin *et al.* (2018)	x	x					
Liu *et al.* (2021a)	x	x					x
Liu *et al.* (2021b)	x	x					
Manavizadeh *et al.* (2020)	x	x					x
Martin *et al.* (2020)	x						
Nasir & Kuo (2020)	x	x					x
Taieb *et al.* (2019)	x						x
Vieira *et al.* (2022)	x	x					x
Wang *et al.* (2020)	x						
Xiang *et al.* (2023)	x	x					
Zhan & Wan (2018)	x						
Fathollahi-Fard *et al.* (2018)	x						
Fathollahi-Fard *et al.* (2019)	x						
Fathollahi-Fard *et al.* (2020)	x						
Carello & Lanzarone (2014)		x					
Cappanera *et al.* (2018)		x					
Carello *et al.* (2018)		x					
Decerle *et al.* (2018a)		x					
Habibnejad-Ledari *et al.* (2019)		x					
Li *et al.* (2022)		x					
Liu *et al.* (2018)		x				x	
Ma *et al.* (2022)		x					
Trautsamwieser & Hirsch (2011)	x						
Decerle *et al.* (2018b)		x					

HTW = hard time window, SKILL = required skills of the caregiver, NUMC = required number of caregivers to carry out the service, MIND = minimum service duration, MAXD = maximum service duration, FREQ = service frequency, SYNC = synchronization of services.

Table 1: Constraints involving services.

Reference	HTW	MAXWT	MINWT	PREF	MAXSERV	OVERQ
Bard *et al.* (2014)	x					
Bertels & Fahle (2006)	x		x			
Braekers *et al.* (2016)	x	x				
Erdem & Bulkan (2017)	x					
Chaieb *et al.* (2020)	x			x		
Rest & Hirsch (2016)	x	x	x			
Kergosien *et al.* (2009)	x					
Liu *et al.* (2017)	x					
Riazi *et al.* (2019)	x					
Mosquera *et al.* (2019)	x				x	
Bachouch *et al.* (2011)	x	x				
Nickel *et al.* (2012)	x					
Decerle *et al.* (2018a)	x					
Decerle *et al.* (2019b)	x					
Decerle *et al.* (2021)	x					
Frifita & Masmoudi (2020)	x					
Grenouilleau *et al.* (2020)	x	x				
Liu *et al.* (2021a)	x					
Liu *et al.* (2021b)	x	x				
Vieira *et al.* (2022)	x	x				
Decerle *et al.* (2018b)	x					
Akjiratikarl *et al.* (2007)		x				
Trautsamwieser & Hirsch (2011)		x				
Maya Duque *et al.* (2015)		x				
Cappanera & Scutellà (2015)		x				
Haddadene *et al.* (2016)		x				
Ait Haddadene *et al.* (2019)		x				
Cappanera *et al.* (2018)		x				
Carello *et al.* (2018)		x				
Chaieb & Ben Sassi (2021)		x				
Euchi *et al.* (2020)		x				
Kandakoglu *et al.* (2020)		x				
Habibnejad-Ledari *et al.* (2019)		x				
Li *et al.* (2022)		x				
Lin *et al.* (2018)		x				
Ma *et al.* (2022)		x				
Malagodi *et al.* (2021)		x				
Martin *et al.* (2020)		x				
Milburn & Spicer (2013)		x				
Moussavi *et al.* (2019)		x				
Nasir & Kuo (2020)		x				
Xiang *et al.* (2023)		x				x
Yang *et al.* (2021)		x				
Belhor *et al.* (2023)		x			x	

HTW = hard time window, MAXWT = maximum working time, MINWT = minimum working time, PREF = preferences of services, MAXSERV = max number of services per caregiver, OVERQ = maximum allowed overqualification.

Table 2: Constraints involving caregivers.

Table 1 shows some characteristics of the HCSP related to the services. The hard time windows and the skills are the most common ones. The hard time windows specify the interval where the services must be completed. The skills indicate the qualifications for a caregiver to correctly carry out a service. Garaix *et al.* (2018), Rest & Hirsch (2016) and Bachouch *et al.* (2011) consider that some services could require more than one caregiver and introduce constraints specifying the number of required caregivers per service. In Mosquera *et al.* (2019) a minimum and maximum duration that must be upheld is introduced, since the services duration is not fixed. Mosquera *et al.* (2019), Nickel *et al.* (2012) and Liu *et al.* (2018) assume that the services are not assigned to predefined days indicating the number of times the services need to be completed throughout the time horizon. Another characteristic of some HCSPs is the synchronization of services, which means that two or more services need to be carried out at the same time.

The characteristics related to the caregivers are presented in Table 2. Usually, there are hard time windows outside of which caregivers cannot work. Another very common characteristic is the maximum allowed working time of the caregivers, that states how many hours they can work during each day (Akjiratikarl *et al.* (2007) and Liu *et al.* (2021b)) or during the whole time horizon (Maya Duque *et al.* (2015) and Martin *et al.* (2020)). There can also be a minimum working time for caregivers (Bertels & Fahle (2006) and Rest & Hirsch (2016)) and hard preferences in terms of the users that the caregivers are willing to attend, which means that they may have excluded users that cannot be assigned to them (Chaieb *et al.* (2020)). Finally, a maximum overqualification for caregivers may be considered, that is, there is an upper bound of how much high skilled caregivers can be assigned to low skill level services (Xiang *et al.* (2023)).

Table 3 outlines the constraints related to the planning of the schedule. The precedence constraints are used to state the relations between services. In Mankowska *et al.* (2014) some services may need to be carried out before others, whereas in Frifita & Masmoudi (2020) some services could be carried out at a different time than others but the order is irrelevant. In Rest & Hirsch (2016) and Martin *et al.* (2020) there is a maximum amount of idle time between services, which is given by the time that caregivers are not traveling between them. In a similar way, Bachouch *et al.* (2011) and Martin *et al.* (2020) consider that there is a maximum distance for a caregiver to travel between two consecutive services.

Reference	CC	PREC	MIDLE	DIST	BREAK	BREAKTW	BREAKDUR
Maya Duque *et al.* (2015)	x						
Bachouch *et al.* (2011)	x			x	x		
Cappanera & Scutellà (2015)	x						
Carello & Lanzarone (2014)	x						
Carello *et al.* (2018)	x						
Cappanera *et al.* (2018)	x						
Lahrichi *et al.* (2022)	x						
Liu *et al.* (2018)	x						
Vieira *et al.* (2022)	x	x			x	x	x
Chaieb *et al.* (2020)		x					
Mankowska *et al.* (2014)		x					
Haddadene *et al.* (2016)		x					
Ait Haddadene *et al.* (2019)		x					
Frifita & Masmoudi (2020)		x					
Nasir & Kuo (2020)		x					
Taieb *et al.* (2019)		x			x	x	x
Rest & Hirsch (2016)			x		x	x	
Martin *et al.* (2020)			x	x			
Liu *et al.* (2017)					x	x	
Euchi *et al.* (2020)					x	x	x
Kandakoglu *et al.* (2020)					x	x	x
Trautsamwieser & Hirsch (2011)					x	x	x
Bard *et al.* (2014)					x	x	x
Liu *et al.* (2021a)					x	x	x
Liu *et al.* (2021b)					x	x	x

CC = continuity of care, PREC = precedence constraints, MIDLE = maximum idle time, DIST = maximum distance between consecutive services, BREAK = lunch break, BREAKTW = time window for the break, BREAKDUR = required duration of the break.

Table 3: Constraints involving the schedule.

There are two other characteristics that are deeply related to the problem under study: the continuity of care and the unpaid breaks. The continuity of care intends to guarantee that users will be attended by the same caregivers and it is handled in multiple ways in the literature. In Carello & Lanzarone (2014) and Carello *et al.* (2018) the authors consider three kinds of patients: the ones with hard continuity of care (whose caregiver cannot be changed), the ones with partial continuity of care (whose caregiver can be changed by adding a penalization cost) and the ones that do not require continuity of care. Other authors, like Cappanera & Scutellà (2015), Cappanera *et al.* (2018) and Liu *et al.* (2018), tackle the continuity of care by setting the maximum number of caregivers that can visit a user. Another option is the one considered in Maya Duque *et al.* (2015), where the number of caregivers who can attend a user depends on the amount of services they require. In Bachouch *et al.* (2011), Lahrichi *et al.* (2022) and Vieira *et al.* (2022) the authors consider that all the services demanded by a user during the week must be carried out by the same caregiver.

The breaks are related to the possible existence of an unpaid break in the daily schedule of the caregivers, which may have a hard time window or a required duration. In some cases, like Bard *et al.* (2014), Kandakoglu *et al.* (2020), Liu *et al.* (2017), Rest & Hirsch (2016), Trautsamwieser & Hirsch (2011) and Vieira *et al.* (2022), caregivers must have a mandatory break if they work a certain amount of time. In Bard *et al.* (2014), if caregivers work 6 or more hours they have a 30 minutes break between 11:00 and 13:00. In Vieira *et al.* (2022) part time caregivers usually have a 20 minutes break if they work 2 consecutive hours, full time caregivers have a meal break that happens after 13:00 and can last between 30 to 120 minutes, depending on the caregiver. In Rest & Hirsch (2016) the required break may be divided into smaller parts, in order to have better schedules while guaranteeing that the caregiver will enjoy her required break time.

In Bachouch *et al.* (2011) and Euchi *et al.* (2020) caregivers always have a mandatory lunch break. In Liu *et al.* (2021a) and Liu *et al.* (2021b) caregivers must have a lunch break if they work during that period of time during the day. In others, multiple breaks during the day are allowed, as in Taieb *et al.* (2019), where caregivers may take 3 breaks: 15 minutes in the morning, a 30 minute lunch break and 15 minutes in the afternoon.

Table 4 presents the objective functions of HSCP regarding the cost for the company. The most common objectives are related to the distance traveled by the caregivers to visit the users' homes. In particular, authors want to minimize travel distance (Akjiratikarl *et al.* (2007) and Nickel *et al.* (2012)), the total traveling time (Decerle *et al.* (2018a) and Frifita & Masmoudi (2020)) and travel costs (Bard *et al.* (2014) and Milburn & Spicer (2013)). In some problems caregivers have an agreed maximum working time that can be surpassed, although it results in an overtime that the company must compensate them for. In these cases the problems minimize either the total overtime (Trautsamwieser & Hirsch (2011) and Carello & Lanzarone (2014)) or its cost (Kandakoglu *et al.* (2020) and Braekers *et al.* (2016)).

A common characteristic of HCSPs is that, sometimes, caregivers have to wait before they can start a service, because of the hard time windows defined by the users, which results in idle times for them. Therefore, to obtain the best schedules it is usually necessary to minimize the idle times (Vieira *et al.* (2022) and Xiang *et al.* (2023)) or their associated cost (Wang *et al.* (2020) and Zhan & Wan (2018)). Additionally, it is possible to minimize the cost of the schedules by reducing the service cost, which may depend on the service time (Bard *et al.* (2014)) or the required skills of the caregiver (Ma *et al.* (2022)).

In some works, the authors minimize the total working time of the caregivers, which includes the accumulated service, travel and idle times (Alves *et al.* (2019) and Martin *et al.* (2020)). Another possibility is to minimize the total working cost (Garaix *et al.* (2018) and Liu *et al.* (2021a)). In situations when users indicate soft preferences in terms of the caregivers that should attend them, another goal is to minimize the cost of assigning users to unfavorable caregivers

(Mosquera *et al.* (2019)). Finally, to incentivize the continuity of care, one can minimize the cost of reassigning services to new caregivers (Carello & Lanzarone (2014) and Liu *et al.* (2021b)).

Reference	TD	TT	TC	OT	OC	IT	IC	SC	WT	WC	PFC	RC
Bard *et al.* (2014)			x		x			x				
Trautsamwieser & Hirsch (2011)		x		x								
Akjiratikarl *et al.* (2007)	x											
Maya Duque *et al.* (2015)	x											
Riazi *et al.* (2019)	x											
Mankowska *et al.* (2014)	x											
Grenouilleau *et al.* (2019)		x										
Bachouch *et al.* (2011)	x											
Nickel *et al.* (2012)	x			x								
Alves *et al.* (2019)	x								x			
Euchi *et al.* (2020)	x				x							
Kandakoglu *et al.* (2020)	x		x		x							
Manavizadeh *et al.* (2020)	x											
Moussavi *et al.* (2019)	x											
Bertels & Fahle (2006)		x							x			
Erdem & Bulkan (2017)		x										
Liu *et al.* (2017)		x										
Mosquera *et al.* (2019)		x									x	
Decerle *et al.* (2018a)		x							x			
Decerle *et al.* (2021)		x										
Frifita & Masmoudi (2020)		x										
Khodabandeh *et al.* (2021)		x										
Vieira *et al.* (2022)		x				x						
Xiang *et al.* (2023)		x		x		x						
Decerle *et al.* (2018b)		x										
Garaix *et al.* (2018)			x							x		
Kergosien *et al.* (2009)			x									
Haddadene *et al.* (2016)			x									
Ait Haddadene *et al.* (2019)			x									
Euchi *et al.* (2020)			x									
Li *et al.* (2022)			x					x				
Ma *et al.* (2022)			x					x				
Milburn & Spicer (2013)			x									
Nasir & Kuo (2020)			x									
Wang *et al.* (2020)			x		x		x	x				
Yang *et al.* (2021)			x					x				
Zhan & Wan (2018)			x		x		x	x				
Carello & Lanzarone (2014)				x								x
Lin *et al.* (2018)				x						x		
Malagodi *et al.* (2021)				x					x			
Braekers *et al.* (2016)					x							
Carello *et al.* (2018)					x							x
Liu *et al.* (2018)					x					x		
Chaieb *et al.* (2020)						x						
Luna *et al.* (2018)									x			
Martin *et al.* (2020)									x			
Belhor *et al.* (2023)									x			
Liu *et al.* (2021a)										x		
Liu *et al.* (2021b)										x		x

TD = travel distance, TT = travel time, TC = travel cost, OT = overtime, OC = overtime cost, IT = idle time, IC = idle cost, SC = service cost, WT = working time, WC = working cost, PFC = preference (of caregivers) cost, RC = reassignment (of services to caregivers) cost.

Table 4: Objective functions involving cost.

Table 5 shows the objective functions that are related to both users and services, that is, the objectives whose goal is to maximize the overall satisfaction. As it was explained before, services usually have a hard time window that states when the service needs to be scheduled but, sometimes, there can be a soft time window within which the user prefers to be attended. In those cases, the objective is to minimize the penalization of carrying out services outside their soft time windows.

In some situations, there may be users' preferences stating the best caregivers to visit them. In that case, the goal is to maximize the compatibility between caregivers and the users they attend. But users may have other optional requirements, for example, they could have a preferred frequency and duration for their services, and the minimization of the differences between the preferred values and the scheduled ones would increase users satisfaction (Mosquera *et al.* (2019)). In other works (Liu *et al.* (2018)) there is a required length for the services. However, to increase users' satisfaction, caregivers are allowed to spend more time with them. So, they will be interested in maximizing the additional duration of the services. Finally, users may have different priority levels, depending on their physical condition and their needs. Thus, maximizing the priorities of the users will also increase their satisfaction (Li *et al.* (2022)).

Reference	STW	PREF	PFREQ	PDUR	ADDDUR	PRI
Trautsamwieser & Hirsch (2011)	x	x				
Bertels & Fahle (2006)	x	x				
Braekers *et al.* (2016)	x	x				
Erdem & Bulkan (2017)	x					
Mankowska *et al.* (2014)	x					
Decerle *et al.* (2018a)	x					
Decerle *et al.* (2018a)	x					
Li *et al.* (2022)	x					
Ma *et al.* (2022)	x					
Manavizadeh *et al.* (2020)	x					
Wang *et al.* (2020)	x					
Yang *et al.* (2021)	x					
Belhor *et al.* (2023)	x					
Decerle *et al.* (2018b)	x					
Maya Duque *et al.* (2015)		x				
Garaix *et al.* (2018)		x				
Haddadene *et al.* (2016)		x				
Ait Haddadene *et al.* (2019)		x				
Xiang *et al.* (2023)		x				
Fathollahi-Fard *et al.* (2020)		x				
Malagodi *et al.* (2021)		x				
Mosquera *et al.* (2019)			x	x		
Liu *et al.* (2018)					x	
Li *et al.* (2022)						x

STW = services soft time window, UPREF = users preferences (in terms of caregivers), PFREQ = services preferred frequency, PDUR = services preferred duration, ADDDUR = services additional duration, PRI = users priority.

Table 5: Objective functions involving users and services.

Table 6 outlines the objective functions that are related to the satisfaction of caregivers. In some works of the literature, there are soft time windows that state when caregivers would like to work. In those cases, the authors (Bertels & Fahle (2006), Luna *et al.* (2018) and Trautsamwieser & Hirsch (2011)) want to minimize the amount of time working outside their soft time windows. Bertels & Fahle (2006) consider that caregivers may indicate preferences in terms of the users to be visited, so that it is recommended to maximize them.

A very common objective in HSCPs is balancing the workload of the caregivers (Carello *et al.* (2018), Milburn & Spicer (2013), Vieira *et al.* (2022)) to prevent some caregivers to work more hours than others. It will avoid a possible dissatisfaction of the workers, since they can feel that they are not being treated equally.

It is possible for the caregivers to specify the region they would like to work in, so it is recommended to minimize the penalization for carrying out services outside their preferred region (Garaix *et al.* (2018)). Some authors, (Luna *et al.* (2018) and Grenouilleau *et al.* (2019)), consider that caregivers have a preferred minimum and maximum working time. In these cases, the goal is to minimize the deviation from the preferred working times.

When services require a certain skill level to be correctly carried out, there is a possibility of assigning caregivers to services below their skill level. It means that some caregivers could be overqualified and their expertise is not being used effectively. Therefore minimizing the overqualification improves caregivers satisfaction (Trautsamwieser & Hirsch (2011) and Khodabandeh *et al.* (2021)). Finally, caregivers may have general satisfaction requirements: general satisfaction (number of consecutive working days, day-offs during weekends, ...) or overtime preferences, which should be taken into account when solving the problems (Habibnejad-Ledari *et al.* (2019), Lin *et al.* (2018) and Xiang *et al.* (2023)).

Reference	STW	UPREF	BWL	REG	DMAXWT	DMINWT	OQ	SAT	OTP
Trautsamwieser & Hirsch (2011)	x						x		
Bertels & Fahle (2006)	x	x							
Luna *et al.* (2018)	x				x	x			
Erdem & Bulkan (2017)			x						
Chaieb *et al.* (2020)			x						
Cappanera & Scutellà (2015)			x						
Carello *et al.* (2018)			x						
Decerle *et al.* (2018a)			x						
Liu *et al.* (2021b)			x						
Milburn & Spicer (2013)			x						
Vieira *et al.* (2022)			x						
Yang *et al.* (2021)			x						
Garaix *et al.* (2018)				x					
Grenouilleau *et al.* (2019)					x	x			
Khodabandeh *et al.* (2021)							x		
Habibnejad-Ledari *et al.* (2019)			x					x	
Lin *et al.* (2018)								x	
Xiang *et al.* (2023)									x

STW = soft time window, UPREF = preferences of users, BWL = balance workload of caregivers, REG = caregivers preferred region, DMAXWT = deviation from maximum working time, DMINWT = deviation from minimum working time, OQ = overqualification, SAT = satisfaction, OTP = overtime preferences.

Table 6: Objective functions involving caregivers.

Finally, Table 7 presents the objective functions related to the caregivers' schedule. In some works (Chaieb *et al.* (2020), Grenouilleau *et al.* (2019) and Nickel *et al.* (2012)) not all the services have to be assigned to a caregiver. In these cases it is necessary to minimize unscheduled services, since it increments the users satisfaction. The synchronization of services may be optional (Erdem & Bulkan (2017), Decerle *et al.* (2018a) and Decerle *et al.* (2018b)) so the number of unsynchronized services is minimized.

When the continuity of care is not hard, it is considered as an objective function, and there are different ways of optimizing it. One option is to minimize the number of different caregivers per user (Milburn & Spicer (2013)). Other option is to deal with the so-called loyalty, which is just the number of caregivers attending each user minus one (Nickel *et al.* (2012)). Finally, some authors (Grenouilleau *et al.* (2019)) minimize a cost for the continuity of care, which depends on the number of caregivers visiting the user.

A common objective in HSCP is to minimize the total number of caregivers used in the schedule, in order to use them in the most efficient way (Euchi *et al.* (2020), Luna *et al.* (2018) and Martin *et al.* (2020)). The maximum workload of the caregivers can be a soft constraint (Euchi *et al.*

(2020) and Kandakoglu *et al.* (2020)), therefore the deviation of the scheduled workload to the maximum one is minimized. The maximum travel distance by a single caregiver can be reduced as far as possible, in order to address the problem of having some caregivers traveling more distance than the others (Moussavi *et al.* (2019)). Finally, the regularity of the scheduled times of repeated services can also be optimized, to attend users at consistent times.

Reference	USERV	USYNC	SCC	DAYS	NUSERS	NC	MWL	MAXTD	START
Chaieb *et al.* (2020)	x								
Garaix *et al.* (2018)	x								
Grenouilleau *et al.* (2019)	x		x						
Liu *et al.* (2017)	x								
Nickel *et al.* (2012)	x		x						
Vieira *et al.* (2022)	x		x						x
Erdem & Bulkan (2017)		x							
Decerle *et al.* (2018a)		x							
Decerle *et al.* (2018b)		x							
Milburn & Spicer (2013)		x	x						
Mosquera *et al.* (2019)				x					
Grenouilleau *et al.* (2020)					x				
Habibnejad-Ledari *et al.* (2019)									
Euchi *et al.* (2020)						x	x		
Kandakoglu *et al.* (2020)						x	x		
Luna *et al.* (2018)						x			
Manavizadeh *et al.* (2020)						x			
Martin *et al.* (2020)						x			
Nasir & Kuo (2020)						x			
Moussavi *et al.* (2019)								x	

USERV = unscheduled services, USYNC = unsynchronized services, SCC = soft continuity of care, DAYS = number of days between services, NUSERS = number of users visited, NC = total number of used caregivers, MWL = maximum workload, MAXTD = maximum traveled distance by one caregiver, START = starting time consistency of repeated services.

Table 7: Objective functions involving the schedule.

There exist different ways in the literature to tackle the multiple objective functions that one can consider in the HCSP:

- Uniobjective problem. One idea is to formulate the HCSP with only one objective function by adopting one of the following approaches:
  - Sum of objectives (Decerle *et al.* (2018b), Zhan & Wan (2018), Liu *et al.* (2021a), Carello & Lanzarone (2014), Liu *et al.* (2017), Rest & Hirsch (2016), Grenouilleau *et al.* (2019) and Decerle *et al.* (2019a)).
  - Weighted sum with the aim of prioritizing some objectives over others (Wang *et al.* (2020), Taieb *et al.* (2019), Martin *et al.* (2020), Manavizadeh *et al.* (2020), Malagodi *et al.* (2021), Luna *et al.* (2018), Liu *et al.* (2021b), Lin *et al.* (2018), Kandakoglu *et al.* (2020), Euchi *et al.* (2020), Nickel *et al.* (2012), Mankowska *et al.* (2014), Garaix *et al.* (2018), Erdem & Bulkan (2017), Bertels & Fahle (2006) and Trautsamwieser & Hirsch (2011)).
  - Lexicographic objective function to guarantee the different priority levels (Mosquera *et al.* (2019)).
- Biobjective problem. Another option is to consider two different objectives simultaneously (Khodabandeh *et al.* (2021), Fathollahi-Fard *et al.* (2020), Fathollahi-Fard *et al.* (2019), Fathollahi-Fard *et al.* (2018), Belhor *et al.* (2023), Oladzad-Abbasabady & Tavakkoli-Moghaddam (2022), Ma *et al.* (2022), Liu *et al.* (2018), Alves *et al.* (2019), Ait Haddadene *et al.* (2019), Haddadene *et al.* (2016) and Braekers *et al.* (2016)).

- Multiobjective problem. The less frequent option is to consider more than two objectives at the same time (Yang *et al.* (2021), Vieira *et al.* (2022), Milburn & Spicer (2013), Habibnejad-Ledari *et al.* (2019), Decerle *et al.* (2019b) and Decerle *et al.* (2018a)).

To solve the models authors use multiple resolution methods, that can be grouped into exact methods, metaheuristic methods or custom heuristics (designed to tackle the specific characteristics of the problem). For the uniobjective versions, the most common resolution methods are:

- Exact methods:
  - Solving the linear programming model, usually by means of commercial solvers (Bertels & Fahle (2006), Garaix *et al.* (2018), Carello & Lanzarone (2014), Kergosien *et al.* (2009) Bachouch *et al.* (2011), Carello *et al.* (2018), Kandakoglu *et al.* (2020) and Taieb *et al.* (2019)).
- Metaheuristic methods:
  - Simulated Annealing algorithm, which is a method that searches neighborhoods and tries to escape from local optima by accepting worse solutions with certain probability that decreases over the iterations (Bertels & Fahle (2006) and Manavizadeh *et al.* (2020)).
  - Tabu Search algorithm, a memory based local search method to avoid getting stuck in local optima (Bertels & Fahle (2006), Rest & Hirsch (2016), Liu *et al.* (2017), Milburn & Spicer (2013) and Zhan & Wan (2018)).
  - Ant Colony Optimization algorithm, a population-based metaheuristic to find shortest path inspired in real life ants and the pheromone trails they lay down (Decerle *et al.* (2019a), Euchi *et al.* (2020), Martin *et al.* (2020) and Yang *et al.* (2021)).
- Custom heuristic methods:
  - Designing custom heuristic algorithms specifics for the problem under study (Akjiratikarl *et al.* (2007), Erdem & Bulkan (2017), Maya Duque *et al.* (2015), Mosquera *et al.* (2019), Chaieb *et al.* (2020), Cappanera & Scutellà (2015), Chaieb & Ben Sassi (2021), Lahrichi *et al.* (2022), Liu *et al.* (2021a) and Malagodi *et al.* (2021)).

Regarding the multiobjective version of the HCSP, the most commonly used methods are:

- Exact methods:
  - Epsilon constraint method, a resolution method that consists in adding objectives as extra epsilon constraints to the mathematical programming model (Khodabandeh *et al.* (2021), Liu *et al.* (2018) and Khodabandeh *et al.* (2021)).
- Metaheuristic methods:
  - Multi-Directional Local Search algorithm, which consists in exploring neighborhoods using single-objective local searches (Braekers *et al.* (2016), Decerle *et al.* (2018a), Decerle *et al.* (2018a) and Decerle *et al.* (2019b)).
  - Non-dominated Sorting Genetic Algorithm, an elitist evolutionary multi-objective algorithm (Haddadene *et al.* (2016), Ait Haddadene *et al.* (2019), Decerle *et al.* (2019b), Habibnejad-Ledari *et al.* (2019), Xiang *et al.* (2023) and Belhor *et al.* (2023)).

Contents

This section details the organization of the manuscript.

Chapter 1:
Chapter 1 focuses on the original problem proposed by the company. The main goal was to update the weekly schedules of the caregivers, that are repeated over time until it is necessary to modify them. The schedules have to be updated to solve a set of possible incidents, which represent permanent changes: registration/cancellation of users, increase/decrease the number of required services and modification of parameters. A special characteristic of this problem is that Mayores is not excessively interested in modifying the caregivers' original schedules, because for them the well-being of the users is really important. Therefore, the incidents should be solved by modifying the previous schedules as little as possible.

This problem was tackled using the Simulated Annealing (SA) algorithm, a metaheuristic optimization method inspired by the metal tempering process. This method is embedded inside a custom heuristic algorithm developed to solve the considered incidents. The heuristic can be divided into three stages: initial phase (to extract the main information), service scheduling (to assign the services to caregivers) and optimization (to improve the schedules). The SA is used in the optimization phase, it starts from an initial solution and then it moves randomly through the solution space to find better schedules. A detailed description of the heuristic algorithm, the SA metaheuristic and the movements used to explore the solution space, is given in this chapter.

This algorithm was implemented in a software tool, to provide Mayores with a decision support system that will help them to obtain the caregivers' weekly schedules. The schedule planner is presented, as well as an example of how the algorithm would solve a set of incidents (cancel a user, increase the duration of the services of another user and register two new users).

The contents of this chapter have been published in Méndez-Fernández *et al.* (2020).

Chapter 2:
Chapter 2 presents the mathematical formulation (a mixed integer programming model (MILP)) of a more general HCSP. The new version of the problem aims to obtain the solutions from scratch, in order to achieve the best possible schedules. That is, instead of having an initial schedule that should not be excessively modified, there is complete freedom to design an optimal schedule for the caregivers. A very important feature of the problem is that if the longest daily break of a caregiver has a duration of 2 hours or more, the break is not considered as working time and it will not be paid, which adds a remarkably difficulty to the problem.

The main decision variables are the ones that determine the routes that the caregivers will follow and state the times at which the caregivers have to start their services. Two different objective functions are considered: the welfare of users, that analyzes how satisfied the users are with the schedules, and the cost of the schedules, that quantifies the workload of the caregivers. On one hand, the welfare is composed of the affinity between users and the caregivers attending them, and the penalization for carrying out services outside their soft time window. On the other hand, the cost of the schedules is composed by the overtime of the caregivers and their total working time. The model includes constraints involving the requirements of the company (hard time windows of services and caregivers and maximum daily working time), routing constraints (start and end at the dummy services, starting time between services should allow the caregiver to travel between them and services only have one service preceding them) and constraints involving the objectives (services soft time window penalization, duration of the unpaid break and overtime of the caregivers).

Further, a detailed example to illustrate the problem is provided, showing how the objective function values would change according to the schedule. The chapter is concluded with the explanation of three possible approaches, proposed to tackle the problem: prioritize the welfare over the cost, prioritize the cost over the welfare and a biobjective alternative.

The mathematical formulation of the HCSP described in this chapter is included in Méndez-Fernández *et al.* (2023a) (accepted for publication) and in the working paper Méndez-Fernández *et al.* (2023c). This model also describes the biobjective version of the problem, that appears in Méndez-Fernández *et al.* (2023b).

Chapter 3:
Chapter 3 provides a thorough description of the metaheuristic used to solve the problem, the Adaptive Large Neighborhood Search (ALNS) algorithm, which is based on destroying and repairing neighborhoods.

First, the overall scheme of the algorithm is presented. It consists of removing services from the schedules of the caregivers, according to a removal operator, and then an insertion operator is used to place the services back into the routes. The repaired schedule updates the current solution, considering an acceptance criteria. The removal and insertion operators are chosen at random according to their probabilities, which increase when the solution is improved. This destroying and repairing process is repeated until a stopping criteria is met. After presenting the scheme of the ALNS, the removal and insertion operators are described, providing their pseudocode and examples to clarify how they work.

It is important to note that the operators modify the routes but, to obtain complete solutions and to be able to evaluate the objective function, the schedules of the services need to be established. That is, while repairing the solution with the insertion operators, it is necessary to set the time at which each service of a route will start. Given a route, three different ways to obtain the schedule are presented:

- Solve the resulting scheduling problem using Constraint Programming.
- Two heuristic approaches: one to obtain schedules for the case the welfare of users is prioritized over the cost, and a different one to obtain schedules but prioritizing the cost over the welfare.

The ALNS algorithm presented in this chapter is included in Méndez-Fernández *et al.* (2023a) and it will also appear in the working paper Méndez-Fernández *et al.* (2023c). The metaheuristic method will also be used in Méndez-Fernández *et al.* (2023b) to solve the biobjective version of the HCSP.

Chapter 4:
Chapter 4 describes the heuristic algorithm developed to obtain the schedule of a route in order to prioritize the welfare of users over the cost of the schedule. The method only works with a route, therefore the welfare coincides with the soft time window penalization (because the affinity is already fixed) and the cost is the working time of the route (the overtime can only be obtained for the whole week).

First, a basic description of the algorithm is presented. The method can be divided into three phases. In the first one the earliest and latest starting times of the services are obtained, as well as the blocks of services that have conflicting soft time windows. The second step consists in obtaining the schedule of the route that has the lowest soft time window penalization. Finally, in the third step the schedule is updated in order to reduce its cost. A running example, of a route with 6 services, is used to illustrate how each step works. After that, some auxiliary functions are introduced in Appendix 4.A for a better understanding of the algorithm.

The heuristic scheduling algorithm introduced in this chapter appears in Méndez-Fernández *et al.* (2023a). It will also be useful for the biobjective HCSP included in the working paper Méndez-Fernández *et al.* (2023b).

Chapter 5:
Chapter 5 presents the heuristic algorithm designed to obtain the schedule of a route in order to prioritize the cost of the schedule over the welfare of the users. As in the previous chapter, the method only works with a route, so the welfare coincides with the soft time window penalization and the cost is the working time of the route.

The chapter focuses on the main elements of the algorithm, whereas additional functions can be found in Appendix 5.A if more information is required. The algorithm starts obtaining the earliest and latest starting times of the services. Then, the schedules with minimal cost value are found. After that, these schedules are updated in order to improve their soft time window penalization while maintaining their cost value. Finally, the schedule with best penalization is chosen. To illustrate how the algorithm works, a running example is used throughout the chapter.

The heuristic scheduling algorithm presented in this chapter is included in Méndez-Fernández *et al.* (2023c), which is currently in progress. It also appears in Méndez-Fernández *et al.* (2023b).

Chapter 6:
Chapter 6 is related to the biobjective version of the HCSP, that is, the problem that considers both objectives simultaneously. The solution of a biobjective problem is the Pareto frontier, a set composed by non dominated solutions (solutions that cannot be improved without degrading one of the objectives), that can be exactly obtained using the Epsilon Constraint method.

The first part of the chapter focuses on the AUGMECON2 method, an improved version of the Epsilon Constraint method that will be used to tackle the biobjective HCSP. It consists in, given a set of grid points that divide the range of one of the objectives, iteratively solving a version of the HCSP that fixes the objective to the value given by the grid.

The second part describes a custom metaheuristic algorithm developed to obtain approximations of the Pareto frontier. The algorithm is based on the methodologies presented in Chapters 3, 4 and 5. Therefore, the main focus of the chapter is based on the new features developed for the biobjective problem, using a running example to illustrate them. The biobjective algorithm can be divided into three steps. In the first one, the set of non dominated solutions is initialized. The second step obtains solutions composed by different routes. Finally, in the third step, the non dominated solutions are obtained. Extra information of the algorithm is presented in Appendix 6.A.

The contents described in this chapter are summarized in the working paper related to the biobjective HCSP, Méndez-Fernández *et al.* (2023b).

Chapter 7:
Chapter 7 presents the computational results used to evaluate the methods described in the previous chapters. To this aim, two different types of datasets are used in the numerical experiments: the Solomon instances and the real data provided by the company.

To study the hierarchical approaches, the first step is to use a state-of-the-art solver (Gurobi) to directly solve the MILP formulation of the problem, using the Solomon instances as input. This is done to have the exact solutions and to see at which instance size the exact method no longer finds solutions. Then, the results obtained with different configurations of the ALNS (combined with the heuristic scheduling algorithms described in Chapters 4 and 5) are analyzed, in order to evaluate the metaheuristic algorithm and to obtain the best parameter configuration.

After that, the ALNS is used to solve real instances and the obtained solutions are compared with the schedules employed by the company. For the hierarchical approach that prioritizes the welfare of users over the cost of the schedule, the performance of the ALNS algorithm combined with constraint programming is also evaluated.

The results related to the hierarchical approach that prioritizes welfare over cost are included in Méndez-Fernández *et al.* (2023a) and the ones of the hierarchical approach that prioritizes cost over welfare will be reported in the working paper Méndez-Fernández *et al.* (2023c).

The biobjective problem is firstly studied by solving the Solomon instances with the AUGMECON2 method, to analyze the computational times of this exact method. Then, the behavior of the biobjective metaheuristic algorithm is evaluated studying the different parameters involved in its configuration. To this aim, the solutions of the algorithm are compared with the ones given by the AUGMECON2 method. Finally, the Pareto frontier for a real instance is presented.

The results of the biobjective version of the HCSP will be summarized in the working paper Méndez-Fernández *et al.* (2023b).

Chapter 1

Original problem proposal by Mayores

During the development of the project, the goal of Mayores was not to create a new schedule for the caregivers, but the modification of the previous plan in order to deal with any incident that may arise. The new schedule would be repeated, on a weekly basis, from the occurrence of the incident until the moment another one arises. The incidences the company deals with are:

Registrations. In this case, the supervisor decides that a new user is going to be added to the system. Thus, she assigns each of her services to the caregivers she considers most appropriate, updating their routes and schedules in order to correctly plan the new services.

Cancelations. When the services of a user are canceled (due to hospital admission, holidays, etc.), the supervisor must remove the corresponding services from the schedules of the caregiver who were attending him/her. Doing so may create breaks in the schedules of the caregivers; therefore, the supervisor must rearrange the schedules in order to reduce their idle time.

Number of services. Sometimes users change the number of services they require. If users request new services, they need to be added to the caregivers schedules. On the contrary, if users reduce the number of needed services, the superfluous services need to be removed from the schedules.

Modifications. If there is a change of parameters (duration, time window or day) in a service that has been previously assigned to caregivers, it is necessary to arrange it. In these cases, supervisors must modify the caregivers schedules to ensure that all services are correctly planned.

In addition, the supervisor may decide to combine these incidents if she considers it convenient. For example, she may cancel the services of a user while increasing the number of services required by another one.

It is important to mention that the company wants to address these incidents in a way that does not excessively modify the previous schedules. This is because the company prioritizes the well-being of its users, meaning that it is advisable not to alter their schedules unless it is strictly necessary.

The chapter is structured as follows. Section 1.1 presents a description of the resolution method based on the Simulated Annealing algorithm. Then, Section 1.2 shows the software application that uses the algorithm to obtain the schedules. Finally, Section 1.3 presents an example that illustrates how the resolution method works.

1.1 Resolution method: a Simulated Annealing algorithm

The problem of updating the schedules presents high difficulty. Therefore, it is necessary to tackle it using metaheuristic techniques, in order to obtain acceptable solutions in reduced computational times. This section describes the algorithm developed to solve the original problem proposed by Mayores. It is based on the Simulated Annealing (SA) method (Kirkpatrick *et al.* (1983)), which is a metaheuristic optimization method inspired by the metal tempering process. The objective of this algorithm is to generate schedules that solve the incidents while adhering to the requirements of the company.

Algorithm 1.1 and Figure 1.1 show the process followed to generate the schedules. The algorithm can be divided into three stages: an initial one in which the main information is extracted, a second one in which the services are assigned to the caregivers and a third one to optimize the schedules.

Algorithm 1.1: Heuristic to solve the incidents

```
   Data: the initial solution $x$, the services to be scheduled $S$ and the services to be removed $\hat{S}$
1  $x \leftarrow$ removeServices($\hat{S}$, $x$) //Remove services from the solution
2  if $S \neq \emptyset$ then
       //If there are services whose schedule needs to be modified
3      $U \leftarrow$ getUsers($S$) //Obtain the list of users that require these services
4      for $u \in U$ do
           //Obtain the time frames of the user's required services
5          $TF \leftarrow$ getTimeFrames($S$, $u$)
6          for $tf \in TF$ do
7              $S_{tf} \leftarrow$ getServices($S$, $u$, $tf$) //Get user's services in the time frame
8              $C_{tf} \leftarrow$ getSortedCaregivers($S_{tf}$, $C$, $u$, $x$) //Sorted list of caregivers
9              for $s \in S_{tf}$ do
10                 for $c \in C_{tf}$ do
11                     $x^* \leftarrow$ scheduleService($s$, $c$, $x$) //Assign the service to the caregiver
12                     $x' \leftarrow$ modifiedSA($x^*$) //Optimize the schedule
13                     if $x'$ is feasible then
14                         $x \leftarrow x'$ //Update the schedule
15                         exit loop
16                     else if $c$ is the last of $C$ then
                           //Assign the service to a dummy caregiver
17                         $x' \leftarrow$ scheduleService($s$, $dummy$, $x$)
18                         $x \leftarrow x'$ //Update the schedule
19 else
       //Check if there are no services whose scheduled needs to be modified
20     $x' \leftarrow$ modifiedSA($x$)
21 return $x$
```

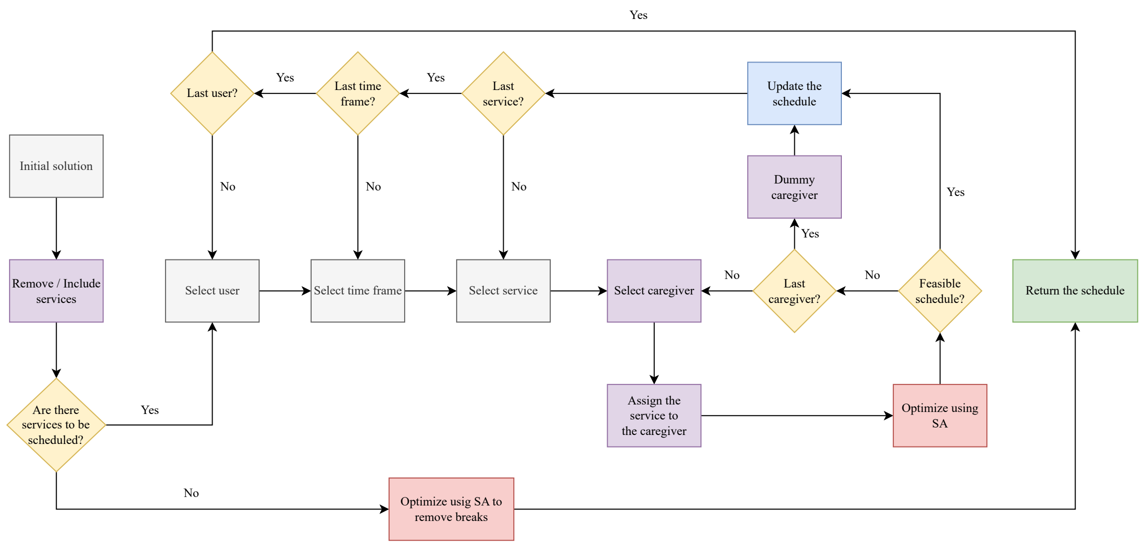


Figure 1.1: Scheme of the algorithm.

1.1.1 Phase 1: initialization

The algorithm begins with an initial solution: the previous schedules of the considered caregivers. The services that have to be deleted (cancelations or reduction of services) are removed from the schedule. The services that need to be scheduled (registration, increase of services and modifications) are selected.

If no services must be scheduled, the solution is optimized applying the Simulated Annealing method, to remove any break that may exist, after which the algorithm ends.

1.1.2 Phase 2: service scheduling

This second phase of the algorithm is performed when there are services to be scheduled. In this step, the caregiver who must perform each of these services is established, as well as the schedule during which they must be conducted to minimize costs and obtain feasible schedules.

The first step in this phase is to select the service to be scheduled and to obtain a list of the available caregivers[1], sorted from best to worst, considering the following elements:

a. The affinity level between the caregiver and the considered user.

b. The difference between the time worked by the caregiver and her maximum time allowed.

c. The average travel time between the considered user and all the users served by the caregiver.

Then, the selected service is assigned to the best caregiver available, scheduling it so that the least overlap with other services of this caregiver is obtained. In this way, the minimal break with other services of this caregiver is produced. After that, as shown in Figure 1.1, the schedules are optimized with the SA method. If an optimum schedule of the service is found, the algorithm goes to the next one. If there is not a feasible schedule of the service with the current caregiver, the algorithm goes to the next available caregiver and the process is repeated. If all available caregivers have been tested, the service is assigned to a dummy caregiver (to allow the supervisors to easily know which services could not be properly scheduled). When the algorithm ends with a solution that includes n dummy caregivers it indicates to the company that it is not possible to find a feasible solution unless they hire n new caregivers.

[1]The available caregivers are the ones that do not surpass their daily maximum number of hours allowed.

1.1.3 Phase 3: the optimization

As it was previously explained, sometimes it is necessary to perform an optimization process to eliminate breaks and/or overlaps from the caregivers' schedules. To this aim, a modified version of the SA method, to fully adapt it to the problem under study, is designed. The idea of the SA method comes from the analogy between physical annealing of solids and combinatorial optimization problems.

The SA, presented in Algorithm 1.2, starts with an initial solution and then moves through the solution space in order to improve it. In each iteration it generates a set of neighbors, choosing the movements according to some probabilities, and selects the one with best objective function value. After that, if the selected neighbor improves the best solution found so far, the current and best solutions are updated with it. In any other case, a probabilistic acceptance criteria is used to update the current solution. The acceptance criteria is given by the probability $exp(-(f(x^*) - f(x))/T_k)$, where f is the objective function, x is the current solution, x^* is the new solution, k is the current iteration and T_k is the temperature. The cooling function, used to decrease the probability of acceptance at iteration k, is $T_k = T_0\beta^k$, where T_0 is the initial temperature. The objective function considered is the lexicographic order of:

a. The overlap[2] between services.

b. The breaks of the caregivers, with the exception of the largest one of the day if it lasts two or more hours, and the travel time between services.

The probability distribution used to select the movements is updated after each iteration. At the beginning the uniform discrete distribution is chosen and, in every iteration, the type of movement leading to the best neighbors is given a larger weight in the distribution.

Algorithm 1.2: Modified Simulated Annealing

```
   Data: the initial solution $x$, the movements $\Omega$ and the weights $\omega$
1  $x' \leftarrow x$
2  for $k \in K$ do
       //Generate a set of L neighbors
3      for $l \in L$ do
4          $movement \leftarrow \boldsymbol{chooseRandom}(\omega, \Omega)$
5          $\hat{x}_l \leftarrow \boldsymbol{movement}(x)$
       //Get the best neighbor
6      $x^* \leftarrow \{\hat{x}_k / \boldsymbol{f}(\hat{x}_k) = \min_{l \in L}\{\boldsymbol{f}(\hat{x}_l)\}\}$
       //Update the best solution
7      if $f(x^*) < f(x')$ then
8          $x' \leftarrow x^*$
9          $x \leftarrow x^*$
10     else if $\boldsymbol{acceptanceCriteria}(x^*, x)$ then
11         $x \leftarrow x^*$
12     $\omega \leftarrow \boldsymbol{updateWeights}(\omega)$
13 return $x'$
```

1.1.3.1 The movements

At each SA iteration, movements are used to retrieve the neighbors of the solution under consideration. These movements can be divided into two classes: the basic movements, which are

[2]The overlap indicates if the caregiver has been assigned to perform two services in the same interval of time, or if she is carrying out a service while also traveling to another user's home.

the simplest ones and tend to generate the best solutions, and the elaborate ones, which arise from merging and/or modifying basic movements, and are not usually used as much as the others but, sometimes, are essential to obtain better solutions.

The basic movements designed are the following:

First movement. This movement consists in modifying the schedule during which a service is performed by moving it forward or backward.

Step 1. One service and the number of minutes that it will be delayed or advanced is selected at random following a uniform distribution.

Step 2. The schedule of the selected service is delayed if the number of minutes chosen is positive or advanced if a negative number is selected.

Example 1.1.1. Illustration of the first movement.

Suppose that a caregiver conducts four services, as shown in Figure 1.2, and has a 70-minute break between service 1 and service 2. Using movement 1, this break can be removed, since delaying service 1 as much as possible (always in accordance with the traveling time to service 2) results in the elimination of the break from the caregiver's schedule.

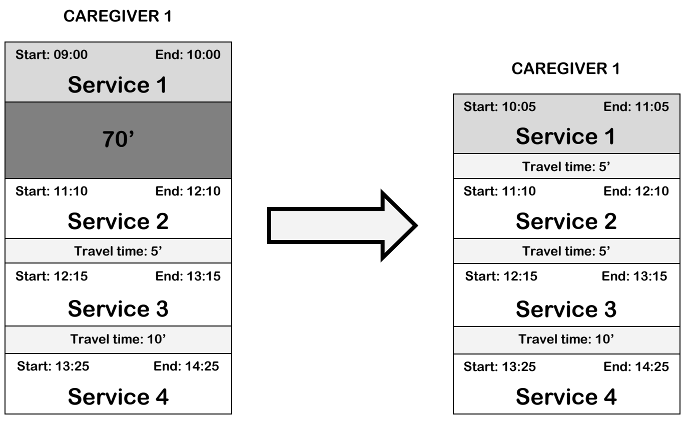


Figure 1.2: Example of the first movement.

Second movement. This movement consists in advancing or delaying a set of consecutive services.

Step 1. One service and the number of minutes that the services' schedules will be delayed or advanced is selected at random following a uniform distribution.

Step 2. The set of services that will be moved is selected, according to the number of minutes. If the quantity is positive then the followers of the service are selected, if it is negative then the predecessors are chosen. Note that, in this step, only a set of consecutive services, between which there is no break with a duration equal to or greater than two hours, are selected.

Step 3. The schedule of the services belonging to the set is modified, delaying or advancing them the given amount of minutes.

Example 1.1.2. Illustration of the second movement.

Suppose that a caregiver has four services with a 65-minute break (Figure 1.3). Applying the second movement, delaying services 1 and 2 the same number of minutes (always considering the travel time between service 2 and service 3), eliminates the break between services 2 and 3.

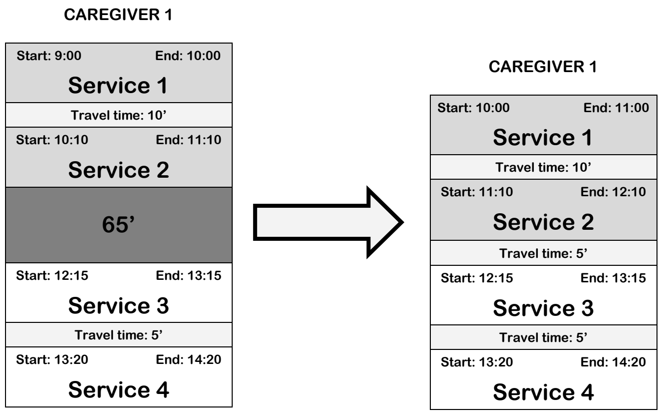


Figure 1.3: Example of the second movement.

Third movement. This movement comprises exchanging the starting times for two given services assigned to the same caregiver on a given day. In this movement, there is also the option of applying the first movement after having conducted the exchange.

Step 1. Two different services are chosen at random following a uniform distribution, such that they are both performed by a single caregiver during the same day.

Step 2. The starting times of the selected services are exchanged, and their ending times are adapted to their new starting times.

Step 3. The first movement will be randomly applied to the selected services.

Example 1.1.3. Illustration of the third movement.

Let us suppose a caregiver performs four services (see Figure 1.4) with a 35-minutes break between service 2 and service 3. If movement 3 is used, exchanging the starting times between services 2 and 4, the break is removed. The reason for this is that service 4 has a duration of 90 minutes, while service 2 lasts only 60 minutes; therefore, this extra duration of service 4 fills the break that the caregiver had in her schedule.

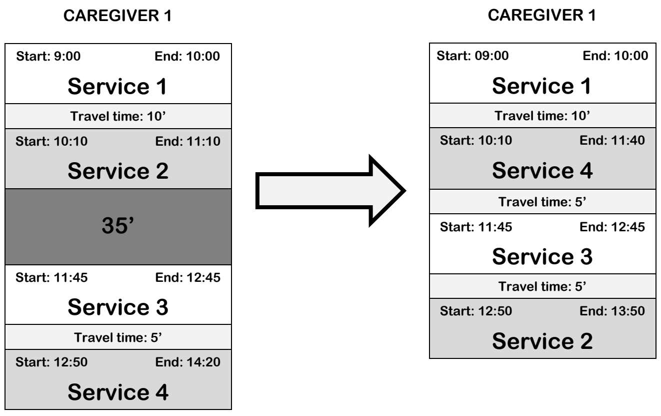


Figure 1.4: Example of the third movement.

Fourth movement. This movement consists in replacing the caregiver who conducts a given service.

Step 1. One service and one caregiver (other than the one who performs this service) are selected at random following a uniform distribution.

Step 2. The selected service is assigned to the new caregiver, maintaining the hours in which it must be conducted.

Example 1.1.4. Illustration of the fourth movement.

In this example, two caregivers (see Figure 1.5) are considered. In the initial schedule of caregiver 1, it can be seen that service 3 is isolated, because there are breaks before and after it. Using movement 4 in such a way that service 3 will now be conducted by caregiver 2, the break of caregiver 1 reaches a duration of 135 minutes (therefore, the company is no longer required to pay her for it). Therefore, the assignation of service 3 to caregiver 2 generates no breaks in her schedule.

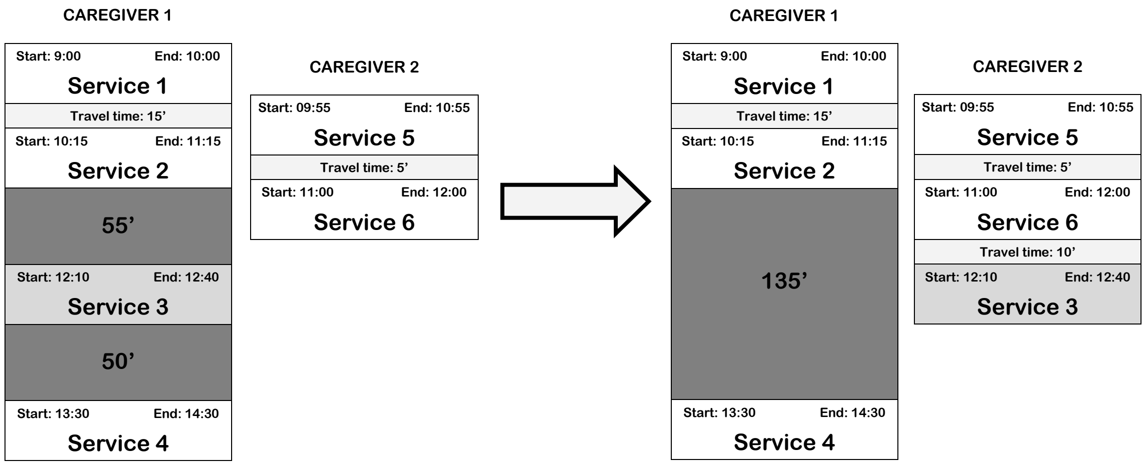


Figure 1.5: Example of the fourth movement.

The elaborated movements considered in the SA method are:

Fifth movement. This movement comprises exchanging the caregivers of two services that are performed by different caregivers on the same day.

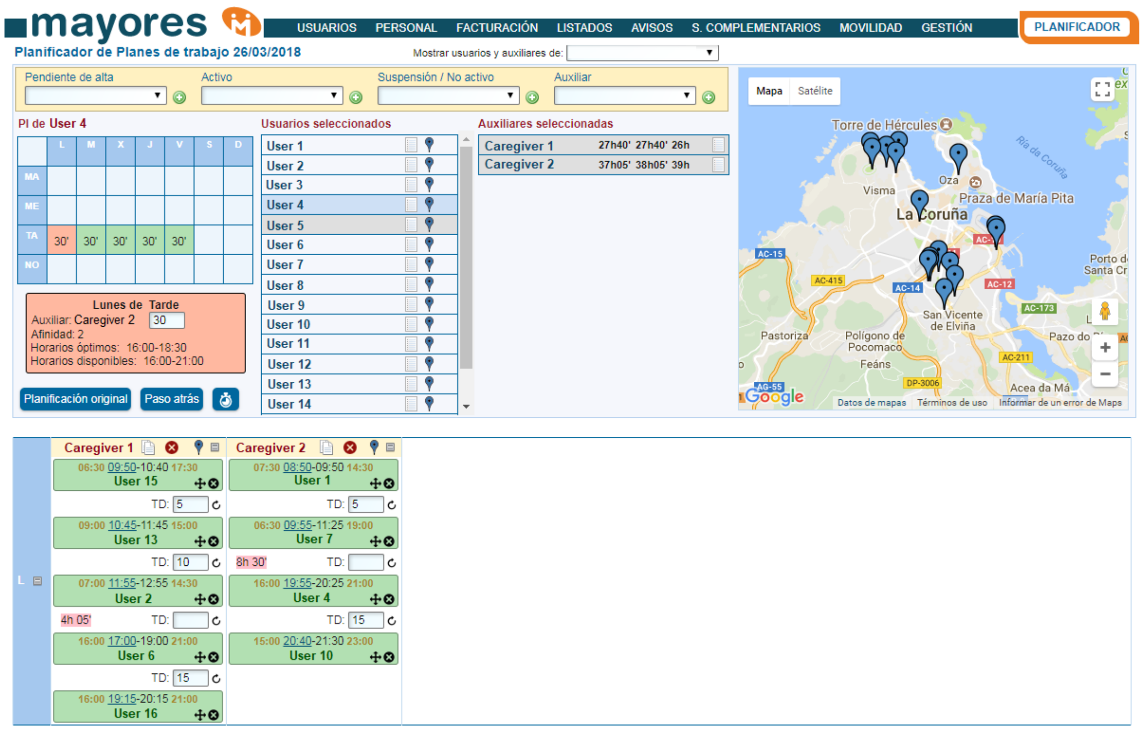
Sixth movement. This movement consists in applying twice the second movement for a service: for the service and the ones before it, and then for the services after it.

Seventh movement. Given two sets of services, this movement exchanges the caregivers who conduct them. To correctly apply this movement, all the services in each set must be performed by the same caregiver.

1.2 The implementation

The algorithm designed to tackle the problem was implemented in the company, providing a decision support tool that helps supervisors to automatically update caregivers' schedule when new incidents arise. In fact, it was integrated into a software application developed by researchers of Database Lab (LBD) of the University of A Coruña. The application allows supervisors to manage and visualize all the procedures related to the problem.

Figure 1.6 presents the schedule planner of the graphical user interface of the application, which shows a summary of the users and caregivers that have been selected by the supervisor as well as the current schedule of the caregivers.



Planificador de Planes de trabajo 26/03/2018

Mostrar usuarios y auxiliares de:

Pendiente de alta | Activo | Suspensión / No activo | Auxiliar

Pl de User 4

	L	M	X	J	V	S	D
MA							
ME							
TA	30'	30'	30'	30'	30'		
NO							

Lunes de Tarde
Auxiliar: Caregiver 2 30
Afinidad: 2
Horarios óptimos: 16:00-18:30
Horarios disponibles: 16:00-21:00

Usuarios seleccionados

User 1
User 2
User 3
User 4
User 5
User 6
User 7
User 8
User 9
User 10
User 11
User 12
User 13
User 14

Auxiliares seleccionadas

Caregiver 1	27h40'	27h40'	26h
Caregiver 2	37h05'	38h05'	39h

L

Caregiver 1	Caregiver 2
06:30 09:50-10:40 17:30 User 15	07:30 08:50-09:50 14:30 User 1
TD: 5	TD: 5
09:00 10:45-11:45 15:00 User 13	06:30 09:55-11:25 19:00 User 7
TD: 10	8h 30' TD:
07:00 11:55-12:55 14:30 User 2	16:00 19:55-20:25 21:00 User 4
4h 05' TD:	TD: 15
16:00 17:00-19:00 21:00 User 6	15:00 20:40-21:30 23:00 User 10
TD: 15	
16:00 19:15-20:15 21:00 User 16	

Figure 1.6: Schedule planner of the application.

The superior frame of Figure1.6 shows the information about the users and caregivers. The box on the left shows all the services required by a specific user. Below it, it is shown the caregiver assigned to a service, its duration, the affinity between the caregiver and the user, and the service's hard and soft time windows. Then, there is the list of selected users and the list of caregivers whose schedule has to be adjusted. Finally, the map on the right shows the location of the considered users.

The inferior frame of the figure is devoted to the schedule for the chosen caregivers, arranging the schedule of each caregiver into a column. Each of the services is displayed as a box, with the name of the user who requires it, the hard time window (marked in orange) and the corresponding start and end times (marked in blue). Between each pair of consecutive services the travel is shown (denoted as "TD") and, in cases where there is a break in the caregivers' schedule, the free time is marked in red.

1.3 Example: solving incidents

A supervisor of the company may have to manage up to, approximately, 40 caregivers and 200 users but, in their day-to-day work, the incidents to solve usually only affect to some of them. Because of this, the example described below is a good representation of how a supervisor would handle a real problem.

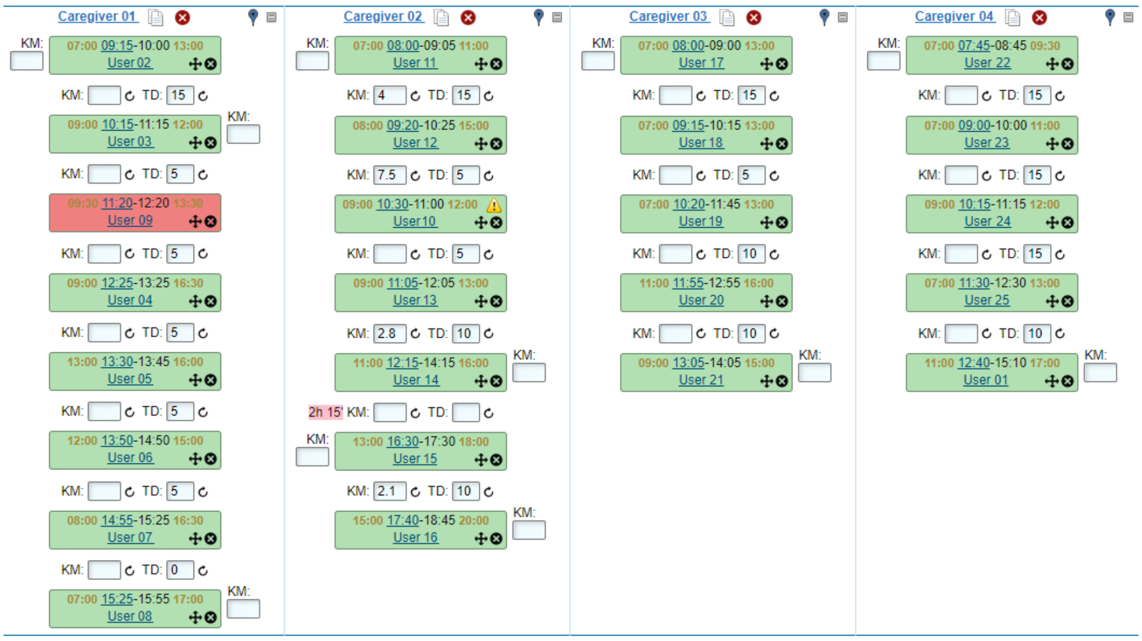
User	Incident	Duration	Hard Time Window
26	Add service	30	9:30 - 12:00
27	Add service	30	11:30 - 13:30
09	Remove service	–	–
10	Service new duration	45	9:00 - 12:00

Table 1.1: Incidents.

In this example the objective is to cancel a user, increase the duration of the services of another user and register two new users. Table 1.1 presents the characteristics of incidents to solve, that is, the user causing them, the incident type, the duration of the services and their hard time window.

Figure 1.7 presents the initial schedule of the example under study, with 4 caregivers (columns) and 25 users (with one service for each of them). Notice that, for simplicity, only the schedule of one day is shown, but the algorithm works with the whole week.

It can be seen that the service of User 09 is colored in red, what means that the user must be canceled. The service of User 10 has a warning signal, which indicates that its scheduled duration needs to be updated. Caregiver 02 has a break, between the services of User 14 and User 15, that has a duration of 2 hours and 15 minutes. Because of this, the break is not considered as working time and there is no need to reduce its duration. Finally, note that the services of User 26 and User 27 are not currently scheduled.


Caregiver 01
KM:
07:00 09:15-10:00 13:00
User 02
KM: TD: 15
09:00 10:15-11:15 12:00
User 03
KM:
KM: TD: 5
09:30 11:20-12:20 13:30
User 09
KM: TD: 5
09:00 12:25-13:25 16:30
User 04
KM: TD: 5
13:00 13:30-13:45 16:00
User 05
KM: TD: 5
12:00 13:50-14:50 15:00
User 06
KM: TD: 5
08:00 14:55-15:25 16:30
User 07
KM: TD: 0
07:00 15:25-15:55 17:00
User 08
KM:

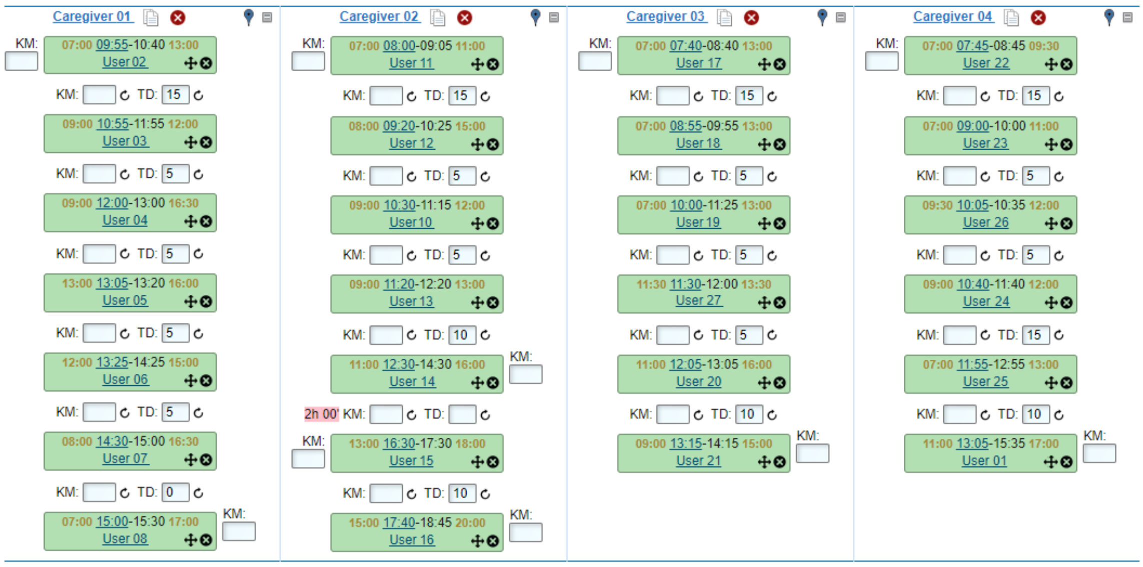
Caregiver 02
KM:
07:00 08:00-09:05 11:00
User 11
KM: 4 TD: 15
08:00 09:20-10:25 15:00
User 12
KM: 7.5 TD: 5
09:00 10:30-11:00 12:00 ⚠
User10
KM: TD: 5
09:00 11:05-12:05 13:00
User 13
KM: 2.8 TD: 10
11:00 12:15-14:15 16:00
User 14
KM:
2h 15' KM: TD:
KM:
13:00 16:30-17:30 18:00
User 15
KM: 2.1 TD: 10
15:00 17:40-18:45 20:00
User 16
KM:

Caregiver 03
KM:
07:00 08:00-09:00 13:00
User 17
KM: TD: 15
07:00 09:15-10:15 13:00
User 18
KM: TD: 5
07:00 10:20-11:45 13:00
User 19
KM: TD: 10
11:00 11:55-12:55 16:00
User 20
KM: TD: 10
09:00 13:05-14:05 15:00
User 21
KM:

Caregiver 04
KM:
07:00 07:45-08:45 09:30
User 22
KM: TD: 15
07:00 09:00-10:00 11:00
User 23
KM: TD: 15
09:00 10:15-11:15 12:00
User 24
KM: TD: 15
07:00 11:30-12:30 13:00
User 25
KM: TD: 10
11:00 12:40-15:10 17:00
User 01
KM:

Figure 1.7: Initial schedule.

Figure 1.8 presents the schedule obtained using the heuristic Algorithm 1.1. This solution was found in 1.63 minutes, executing the algorithm to solve the incidents for the whole week, not just for the day shown in the figures. The services of User 26 and User 27 have been correctly added to the schedule, assigned them to Caregiver 04 and Caregiver 03, respectively. The algorithm removed the service of User 09, from the plan of Caregiver 01, and updated the schedule of the other services to overcome the break created when removing the service. Finally, the service of User 10 was adapted to its new duration of 45 minutes and, to make space for it, some of its follower services had to be delayed.



Caregiver 01

KM:
07:00 09:55-10:40 13:00
User 02
KM: TD: 15
09:00 10:55-11:55 12:00
User 03
KM: TD: 5
09:00 12:00-13:00 16:30
User 04
KM: TD: 5
13:00 13:05-13:20 16:00
User 05
KM: TD: 5
12:00 13:25-14:25 15:00
User 06
KM: TD: 5
08:00 14:30-15:00 16:30
User 07
KM: TD: 0
07:00 15:00-15:30 17:00
User 08
KM:

Caregiver 02

KM:
07:00 08:00-09:05 11:00
User 11
KM: TD: 15
08:00 09:20-10:25 15:00
User 12
KM: TD: 5
09:00 10:30-11:15 12:00
User 10
KM: TD: 5
09:00 11:20-12:20 13:00
User 13
KM: TD: 10
11:00 12:30-14:30 16:00
User 14
KM:
2h 00' KM: TD:
KM:
13:00 16:30-17:30 18:00
User 15
KM: TD: 10
15:00 17:40-18:45 20:00
User 16
KM:

Caregiver 03

KM:
07:00 07:40-08:40 13:00
User 17
KM: TD: 15
07:00 08:55-09:55 13:00
User 18
KM: TD: 5
07:00 10:00-11:25 13:00
User 19
KM: TD: 5
11:30 11:30-12:00 13:30
User 27
KM: TD: 5
11:00 12:05-13:05 16:00
User 20
KM: TD: 10
09:00 13:15-14:15 15:00
User 21
KM:

Caregiver 04

KM:
07:00 07:45-08:45 09:30
User 22
KM: TD: 15
07:00 09:00-10:00 11:00
User 23
KM: TD: 5
09:30 10:05-10:35 12:00
User 26
KM: TD: 5
09:00 10:40-11:40 12:00
User 24
KM: TD: 15
07:00 11:55-12:55 13:00
User 25
KM: TD: 10
11:00 13:05-15:35 17:00
User 01
KM:

Figure 1.8: Final schedule.

The algorithm is very conservative because it maintains, as much as possible, the initial schedules of the caregivers. Therefore, the example illustrates how the algorithm is able to solve real-life incidents, providing updated schedules for the company.

Chapter 2

Mathematical formulation

After solving the original problem presented by the company, it was decided to study the HCSP from an optimization point of view, that is, to solve the problem in a more general manner. To this aim, new characteristics were added to the problem:

Soft time window. Apart from the hard time window, it could be interesting to allow users to specify a soft time window that states the times within which they would prefer to be attended. Although there is not need to uphold the soft time window, it would increase the satisfaction levels of the users.

Caregivers time window. Caregivers may also have a hard time window outside of which they cannot carry out any service. This feature increases the difficulty of the problem by restricting when caregivers can work.

Continuous time. In the original problem, each working day was divided in time slots. Now, to have more flexibility, the time is considered as a continuous variable.

Schedules from scratch. One important characteristic of the problem presented by the company is the fact that they wanted to update current schedules to solve new incidents introducing as few as possible modifications. Therefore, to study a more broad version of the problem, the schedules will be designed from scratch, that is, there is no need to have an initial solution. This also means that there is no need to maintain the previous schedules, which results in more freedom when finding the best possible schedules.

Objectives. Because of the increased flexibility resulting from solving the problem without an initial solution, the following two objectives that partly conflict can be considered: cost and welfare. The cost is related to the monetary implications of the solutions, that is, the salary of the caregivers. To reduce it, the travel times of the routes and the breaks should be minimized. The welfare is related to the preferences of the users, with respect to the caregivers and the soft time windows.

All these new characteristics result in a more difficult and rich problem. This chapter presents the mathematical formulation of the problem, an example to illustrate it and the different approaches considered to tackle it.

This chapter is organized as follows. Section 2.1 presents a thorough description of the mixed integer linear programming model of the problem. In Section 2.2 there is an illustrative example of the problem under study. Finally, Section 2.3 describes the three different approaches considered to tackle the problem.

2.1 The mixed integer programming model

In this section the Mixed Integer Linear Programming (MILP) model of the problem under study is thoroughly described. First, the parameters of the model and the main decision variables of the problem are introduced. After that, the objective functions and the constraints of the problem are described in detailed. The section concludes with a summary of the whole model.

2.1.1 Parameters

The *parameters* involved in the formulation of the model are as follows.

- $D = \{1, ..., 7\}$ is the set of days.
- $N = \{1, ..., n\}$ is the set of caregivers.
- $S = \{1, ..., s-1\}$ is the set of services to be completed and $S_{-k} = S \setminus \{k\}$.
- Two dummy services are also considered: 0 the initial dummy service and s the ending dummy service. Under these assumptions, the following sets are defined: $S^0 = S \cup \{0\}$, $S^1 = S \cup \{s\}$ and $S^{01} = S \cup \{0\} \cup \{s\}$. Given a service $k \in S$, two sets are introduced: $S^0_{-k} = S^0 \setminus \{k\}$ and $S^1_{-k} = S^1 \setminus \{k\}$.
- ρ^i_j is equal to 1 if caregiver $i \in N$ can perform service $j \in S$, and 0 otherwise.
- $\lambda^i_j \in \{0, 1, 2, 3, 4, 5\}$, is the affinity level between caregiver $i \in N$ and service $j \in S$. Note that $\rho^i_j = 0$ implies that $\lambda^i_j = 0$.
- η_j is the duration of service $j \in S$.
- $[\underline{\alpha}^d_j, \bar{\alpha}^d_j]$, $[\underline{\beta}^d_j, \bar{\beta}^d_j]$ are, respectively, the available and optimal time windows of service $j \in S$ on day $d \in D$. Note that, for the cases where service $j \in S$ does not belong to day $d \in D$ the time window is $\underline{\alpha}^d_j = \bar{\alpha}^d_j$.
- For every caregiver $i \in N$ on day $d \in D$, the time window $[\underline{\gamma}^{id}_j, \bar{\gamma}^{id}_j]$ indicates the availability of the caregiver.
- θ_{jk} is the travel time between services $j \in S$ and $k \in S^1$. For the dummy services the travel times are zero, that is, $\theta_{jk} = 0$ if $k = s$.
- ν^i_{con} is the agreed weekly working time of caregiver $i \in N$ and ν^{id} is the maximum time that caregiver $i \in N$ is allowed to work on day $d \in D$.
- π_{min} is the minimum length of time required for the largest break to be unpaid.
- $\omega_1, \omega_2, \omega_3, \omega_4 \in \mathbb{R}$ are the weights of each objective in the considered objective function.

2.1.2 Decision variables

The *main decision variables* are the following:

- For all $i \in N$, $d \in D$, $j \in S^0$, $k \in S^1$, with $j \neq k$,
$$x^{id}_{jk} = \begin{cases} 1, & \text{if caregiver } i \text{ goes directly from service } j \text{ to service } k \text{ on day } d; \\ 0, & \text{otherwise.} \end{cases}$$
- $t^{id}_j \in \mathbb{R}^+_0$ is the time caregiver $i \in N$ starts handling service $j \in S^{01}$ on day $d \in D$. Note that $t^{id}_j = 0$ in case caregiver $i \in N$ is not assigned to service $j \in S$ on day $d \in D$.

The *auxiliary decision variables* are the following:

- For all $i \in N$, $d \in D$, $j \in S$, $k \in$, with $j \neq k$,
$$y_{jk}^{id} = \begin{cases} 1, & \text{if caregiver } i \text{ has her largest break of day } d \text{ between services } j \text{ and } k; \\ 0, & \text{otherwise.} \end{cases}$$
- For all $i \in N$, $d \in D$,
$$\bar{y}^{id} = \begin{cases} 1, & \text{if there is no break for caregiver } i \text{ on day } d; \\ 0, & \text{otherwise.} \end{cases}$$
- $r^{id} \in \mathbb{R}_0^+$ is the greatest break of caregiver $i \in N$ on day $d \in D$.
- For all $i \in N$, $d \in D$,
$$u^{id} = \begin{cases} 1, & \text{if } r^{id} \geq \pi_{min}; \\ 0, & \text{otherwise.} \end{cases}$$
- For all $i \in N$, $d \in D$,
$$\hat{r}^{id} = \begin{cases} r^{id}, & \text{if } r^{id} \geq \pi_{min}; \\ 0, & \text{otherwise.} \end{cases}$$
- $z^i \in \mathbb{R}_0^+$ is the amount of overtime of caregiver $i \in N$.
- $v_j^{start} \in \mathbb{R}_0^+$ is the penalization for carrying out service $j \in S$ before its soft time window.
- $v_j^{end} \in \mathbb{R}_0^+$ is the penalization for carrying out service $j \in S$ after its soft time window.

Notice that, if service $j \in S$ is not carried out on day $d \in D$, the variable is $x_{jk}^{id} = 0$ for all $i \in N$, $k \in S^1$ and $t_j^{id} = 0$ for all $i \in N$.

2.1.3 Objective function

The objective function considers four different elements, which can be divided into two groups:

- Objective (2.1) accounts for the welfare of users, i.e., the total affinity between services and caregivers and the penalization of performing services outside their soft time windows.
$$f_1 = \min\ \omega_1 \sum_{i\in N}\sum_{d\in D}\sum_{j\in S}\sum_{k\in S^1} \lambda_j^i x_{jk}^{id} + \omega_2 \sum_{j\in S}(v_j^{start} + v_j^{end}) \tag{2.1}$$
- Objective (2.2) refers to schedule cost, which is composed by the the amount of overtime of the caregivers and their total worked time.
$$f_2 = \min\ \omega_3 \sum_{i\in N} z^i + \omega_4 \sum_{i\in N}\sum_{d\in D}(t_s^{id} - t_0^{id} - \hat{r}^{id}) \tag{2.2}$$

In order to set the weights ω_1 and ω_2, corresponding to objectives related to the welfare of users, the fact that $v_j^{start} \leq \max_{d\in D}\{\underline{\beta}_j^d - \underline{\alpha}_j^d\}$ and $v_j^{end} \leq \max_{d\in D}\{\bar{\alpha}_j^d - \bar{\beta}_j^d\}$ for all $j \in S$ is taken into account. Therefore, setting $\omega_1 = -\max\left\{1, \sum_{j\in S}\left[\max_{d\in D}\{\underline{\beta}_j^d - \underline{\alpha}_j^d\} + \max_{d\in D}\{\bar{\alpha}_j^d - \bar{\beta}_j^d\}\right]\right\}$ and $\omega_2 = 1$ guarantees that the affinity will be prioritized over soft time windows penalization[1]. For the cost of the schedule, since the units of this objective function coincide and there is no need to prioritize one over the other, the weights are $\omega_3 = \omega_4 = 1$.

[1]Note that, $\omega_1 < 0$ because the goal is to maximize the affinity levels in a minimization problem.

2.1.4 Constraints

Constraint (2.3) guarantees that every service is carried out by a single caregiver on a single day and that, after performing it, the caregiver goes directly to attend another service.

$$\sum_{i\in N}\sum_{d\in D}\sum_{k\in S^1_{-j}} x^{id}_{jk} = 1 \qquad \forall j \in S \tag{2.3}$$

$$x^{id}_{jk} \in \{0,1\} \qquad \forall i \in N, \forall d \in D, \forall j \in S^0, \forall k \in S^1, j \neq k$$

Constraint (2.4) guarantees that every service can only have one previous service.

$$\sum_{i\in N}\sum_{d\in D}\sum_{j\in S^0_{-k}} x^{id}_{jk} = 1 \qquad \forall k \in S \tag{2.4}$$

Constraint (2.5) assures that services can only be assigned to caregivers who can perform them.

$$\sum_{d\in D}\sum_{k\in S^1_{-j}} x^{id}_{jk} \leq \rho^i_j \qquad \forall i \in N, \forall j \in S \tag{2.5}$$

Constraint (2.6) guarantees that the routes of every caregiver on every day start at the initial dummy service.

$$\sum_{k\in S^1} x^{id}_{0k} = 1 \qquad \forall i \in N, \forall d \in D \tag{2.6}$$

Constraint (2.7) states that the routes of every caregiver on every day end at the ending dummy service.

$$\sum_{j\in S^0} x^{id}_{js} = 1 \qquad \forall i \in N, \forall d \in D \tag{2.7}$$

Constraint (2.8) avoids the segmentation of caregivers' routes.

$$\sum_{j\in S^0_{-h}} x^{id}_{jh} - \sum_{k\in S^1_{-h}} x^{id}_{hk} = 0 \qquad \forall i \in N, \forall d \in D, \forall h \in S \tag{2.8}$$

Constraint (2.9) focuses on the completion of the services in S.

$$\underline{\alpha}^d_j \sum_{k\in S^1_{-j}} x^{id}_{jk} \leq t^{id}_j \leq (\bar{\alpha}^d_j - \eta_j) \sum_{k\in S^1_{-j}} x^{id}_{jk} \qquad \forall i \in N, \forall d \in D, \forall j \in S \tag{2.9}$$

$$t^{id}_j \geq 0 \qquad \forall i \in N, \forall j \in S^{01}, \forall d \in D$$

Constraint (2.10) states that the starting times of two consecutive services allow the caregiver to complete the initial service and to travel to the following one.

$$t^{id}_j + (\eta_j + \theta_{jk})x^{id}_{jk} \leq t^{id}_k + \bar{\alpha}^d_j(1 - x^{id}_{jk}) \qquad \forall i \in N, \forall d \in D, \; \forall j \in S, \forall k \in S^1, j \neq k \tag{2.10}$$

Constraints (2.11) and (2.12) are needed to ensure that dummy services are performed within the caregivers' available time window.

$$t_0^{id} \geq \underline{\gamma}^{id} \qquad \forall i \in N, \forall d \in D \tag{2.11}$$

$$t_s^{id} \leq \bar{\gamma}^{id} \qquad \forall i \in N, \forall d \in D \tag{2.12}$$

Constraints (2.13) and (2.14) ensure that the initial dummy service will be carried out at the same time as the first real service of the caregiver.

$$t_0^{id} \leq t_k^{id} + \bar{\gamma}^{id}(1 - x_{0k}^{id}) \qquad \forall i \in N, \forall d \in D, \forall k \in S^1 \tag{2.13}$$

$$t_0^{id} \geq t_k^{id} - \bar{\gamma}^{id}(1 - x_{0k}^{id}) \qquad \forall i \in N, \forall d \in D, \forall k \in S^1 \tag{2.14}$$

Constraint (2.15) establishes that the ending dummy service will be performed right after its previous service ends. Notice that this constraint only involves services $j \in S$, because the case $j = 0$ was already considered in Constraint (2.14).

$$t_s^{id} \leq (t_j^{id} + \eta_j) + \bar{\gamma}^{id}(1 - x_{js}^{id}) \qquad \forall i \in N, \forall d \in D, \forall j \in S \tag{2.15}$$

Constraint (2.16) guarantees that the worked time of every caregiver on every day, excluding the largest break she has during the day if it lasts more than π_{min}, upholds the caregivers maximum daily working time.

$$t_s^{id} - t_0^{id} - \hat{r}^{id} \leq \nu^{id} \qquad \forall i \in N, \forall d \in D \tag{2.16}$$

To consider in the objective function the caregivers scheduled overtime, it is necessary to define the variable $z^i \in \mathbb{R}^+$ for all $i \in N$, which states the difference between the caregiver scheduled time and her working hours according to her contract. Constraint (2.17) computes the amount of overtime of the schedule of each caregiver[2].

$$z^i \geq \sum_{d \in D} \left(t_s^{id} - t_0^{id} - \hat{r}^{id} \right) - \nu_{con}^i \qquad \forall i \in N \tag{2.17}$$

$$z^i \geq 0 \qquad \forall i \in N$$

The variable, $r^{id} \in \mathbb{R}^+$, which is the greatest rest of the caregiver $i \in N$ on day $d \in D$, will play a key role in Constraint (2.18). It is important to mention that the time lost, for every caregiver $i \in N$ and day $d \in D$, between two services $j, k \in S^{01}$ is $t_k^{id} - (t_j^{id} + \eta_j + \theta_{jk})$. So, the lost time is the difference between:

a. The ending time of service j, $t_j^{id} + \eta_j$, plus the travel time between the services, θ_{jk}.

b. The starting time of service k, t_k^{id}.

Hence, Constraint (2.18) sets the minimum duration of the largest break of a caregiver on a day, considering every possible pair of consecutive services.

$$r^{id} \geq t_k^{id} - (t_j^{id} + \eta_j + \theta_{jk}) - \bar{\gamma}^{id}(1 - x_{jk}^{id}) \qquad \forall i \in N, \forall d \in D, \forall j, k \in S, j \neq k \tag{2.18}$$

$$r^{id} \geq 0 \qquad \forall i \in N, \forall d \in D$$

[2]Note that, when the right side of (2.17) is negative, because $z^i \geq 0$ and the term is minimized in the objective function, the variable will be 0.

This constraint leads to two possible scenarios:

a. If the caregiver carries out service j and k consecutively (in this order), which means that $x_{jk}^{id} = 1$, the value will be $r^{id} \geq t_k^{id} - (t_j^{id} + \eta_j + \theta_{jk})$.

b. If she does not perform both services j and k consecutively or if she is not assigned to one of them, that is $x_{jk}^{id} = 0$, then $r^{id} \geq t_k^{id} - (t_j^{id} + \eta_j + \theta_{jk}) - \bar{\gamma}^{id}$, where $t_k^{id} - (t_j^{id} + \eta_j + \theta_{jk}) - \bar{\gamma}^{id} \leq 0$. Note that using parameter $\bar{\gamma}^{id}$ in Constraint (2.18), guarantees that $t_k^{id} - (t_j^{id} + \eta_j + \theta_{jk}) - \bar{\gamma}^{id} \leq 0$.

The variable $y_{jk}^{id} \in \{0,1\}$ will allow us to select between which two consecutive services the caregiver has her largest break, that is, for all $i \in N$, $d \in D$, $j,k \in S$, with $j \neq k$,

$$y_{jk}^{id} = \begin{cases} 1, & \text{if caregiver } i \text{ has her largest break on day } d \text{ between services } j \text{ and } k; \\ 0, & \text{otherwise.} \end{cases}$$

Constraint (2.19) computes the maximum duration of the largest break of a caregiver on a day, considering every possible pair of consecutive services.

$$r^{id} \leq t_k^{id} - (t_j^{id} + \eta_j + \theta_{jk}) + \bar{\gamma}^{id}(1 - x_{jk}^{id}) + \bar{\gamma}^{id}(1 - y_{jk}^{id}) \qquad \forall i \in N, \forall d \in D, \forall j,k \in S, j \neq k \tag{2.19}$$

Two different options can arise in Constraint (2.19):

a. If the caregiver carries services j and k consecutively, that is $x_{jk}^{id} = 1$, then $r^{id} \leq t_k^{id} - (t_j^{id} + \eta_j + \theta_{jk}) + \bar{\gamma}^{id}(1 - y_{jk}^{id})$.

b. If the caregiver does not carry both services j and k in a consecutive way or she is not assigned to one of them, which means that $x_{jk}^{id} = 0$, then $r^{id} \leq t_k^{id} - (t_j^{id} + \eta_j + \theta_{jk}) + \bar{\gamma}^{id} + \bar{\gamma}^{id}(1 - y_{jk}^{id})$.

It is necessary to define a new auxiliary variable $\bar{y}^{id} \in \{0,1\}$ to determine the cases where there is no break between the services of a caregiver. For every $i \in N$ and $d \in D$, the variable is

$$\bar{y}^{id} = \begin{cases} 1, & \text{if there is no break for caregiver } i \text{ on day } d; \\ 0, & \text{otherwise.} \end{cases}$$

The interest of introducing this variable is due to the fact that Constraints (2.10) and (2.11) are not designed to involve dummy services. Thus, there are two cases when this new variable will be required:

- *The caregiver does not carry out any real service, that is, the caregiver goes directly from dummy service 0 to dummy service s. In this case, for caregiver i and day d, variable $\bar{y}^{id}$ is necessary because there is no variable y_{0s}^{id}.*
- *The caregiver only carries out one real service, that is, caregiver starts at dummy service 0, then goes to service j and finally ends at dummy service s. In this case, for caregiver i and day d, variable $\bar{y}^{id}$ is necessary because there are no variables y_{0j}^{id} and y_{js}^{id}.*

Note that this variable can also be activated if a caregiver's schedule has no break between any of her real services.

Constraint (2.20) imposes that the maximum duration of the largest break of a caregiver on a day must be zero if the caregiver does not carry out any real service.

$$r^{id} \leq \bar{\gamma}^{id}(1-\bar{y}^{id}) \qquad \forall i \in N, \forall d \in D \tag{2.20}$$

Constraint (2.21) selects one of these breaks as the largest one, for every caregiver and day.

$$\sum_{j\in S}\sum_{k\in S_{-j}} y_{jk}^{id} + \bar{y}^{id} = 1 \qquad \forall i \in N, \forall d \in D \tag{2.21}$$

$$y_{jk}^{id} \in \{0,1\} \qquad \forall i \in N, \forall d \in D, \forall j,k \in S, j \neq k$$

$$\bar{y}^{id} \in \{0,1\} \qquad \forall i \in N, \forall d \in D$$

Furthermore, Constraint (2.22) states that the largest break of a caregiver on a day has to be between two consecutive services.

$$y_{jk}^{id} \leq x_{jk}^{id} \qquad \forall i \in N, \forall d \in D, \forall j \in S, \forall k \in S, j \neq k \tag{2.22}$$

According to Constraints (2.18), (2.20) and (2.21), the maximum value of the variable r^{id} is set according to two cases:

a. *If the caregiver is not assigned to one of the services (j or k), or does not carry out these two services consecutively, $x_{jk}^{id} = 0$, $y_{jk}^{id} = 0$ and $r^{id} \leq t_k^{id} - (t_j^{id} + \eta_j + \theta_{jk}) + 2\bar{\gamma}^{id}$, where $t_k^{id} - (t_j^{id} + \eta_j + \theta_{jk}) + 2\bar{\gamma}^{id}$ is a positive number greater than any possible break.*

b. *If the caregiver carries out the two services j and k in a consecutive way, $x_{jk}^{id} = 1$, $r^{id} \leq t_k^{id} - (t_j^{id} + \eta_j + \theta_{jk}) + \bar{\gamma}^{id}(1 - y_{jk}^{id})$ and y_{jk}^{id} will select the largest one of them.*

To determine whether the largest break of a caregiver on a day is equal to or greater than π_{min}, the following auxiliary binary variable is introduced for all $i \in N$ and $d \in D$,

$$u^{id} = \begin{cases} 1, & \text{if } r^{id} \geq \pi_{min}; \\ 0, & \text{otherwise.} \end{cases}$$

Constraints (2.23) and (2.24) determine if the largest break of a caregiver on a day is equal to or greater than the maximum paid break allowed. To this aim, it is necessary to define $\varepsilon \in \mathbb{R}^+$ a constant close to 0.

$$r^{id} - \pi_{min} \geq \pi_{min}(u^{id} - 1) \qquad \forall i \in N, \forall d \in D \tag{2.23}$$

$$r^{id} - \pi_{min} + \varepsilon \leq (\bar{\gamma}^{id} - \underline{\gamma}^{id})u^{id} \qquad \forall i \in N, \forall d \in D \tag{2.24}$$

For these two constraints there are two possible scenarios:

a. *If the largest break of the caregiver on a day has a duration greater than π_{min}, that is $r^{id} > \pi_{min}$, it is necessary to set $u^{id} = 1$ so that $r^{id} - \pi_{min} \geq 0$ and $r^{id} - \pi_{min} + \varepsilon \leq (\bar{\gamma}^{id} - \underline{\gamma}^{id})$.*

b. *If the largest break of the caregiver on a day is equal to π_{min}, that is $r^{id} = \pi_{min}$, then $u^{id} = 1$ so that $r^{id} - \pi_{min} \geq 0$ and $r^{id} - \pi_{min} + \varepsilon \leq (\bar{\gamma}^{id} - \underline{\gamma}^{id})$. Note that this is the only case where it is necessary to use ε.*

c. *If the largest break of the caregiver on a day is shorter than π_{min}, that is $r^{id} < \pi_{min}$, then $u^{id} = 0$ so that $r^{id} - \pi_{min} \geq -\pi_{min}$ and $r^{id} - \pi_{min} + \varepsilon \leq 0$.*

The variable $\hat{r}^{id} \geq 0$ will take the value of the largest break for caregiver $i \in N$ on day $d \in D$ if it is equal or greater than π_{min}, that is

$$\hat{r}^{id} = \begin{cases} r^{id}, & \text{if } r^{id} \geq \pi_{min}; \\ 0, & \text{otherwise.} \end{cases}$$

Constraint (2.25) states that $\hat{r}^{id}$ will be null if $u^{id} = 0$, for every caregiver $i \in N$ and day $d \in D$.

$$\hat{r}^{id} \leq (\bar{\gamma}^{id} - \underline{\gamma}^{id})u^{id} \qquad \forall i \in N, \forall d \in D \tag{2.25}$$

$$\hat{r}^{id} \geq 0 \qquad \forall i \in N, \forall d \in D$$

$$u^{id} \in \{0,1\} \qquad \forall i \in N, \forall d \in D$$

Constraint (2.26) guarantees that $\hat{r}^{id}$ will not be greater than the largest break of caregiver $i \in N$ on day $d \in D$.

$$\hat{r}^{id} \leq r^{id} \qquad \forall i \in N, \forall d \in D \tag{2.26}$$

Constraint (2.27) states that $\hat{r}^{id}$ will be greater than or equal to the largest break of caregiver $i \in N$ on day $d \in D$ if the variable $u^{id} = 1$.

$$\hat{r}^{id} \geq r^{id} - (\bar{\gamma}^{id} - \underline{\gamma}^{id})(1 - u^{id}) \qquad \forall i \in N, \forall d \in D \tag{2.27}$$

The variable $v_j^{start} \in \mathbb{R}^+$ will be used to represent $\max\left\{0, \sum_{d \in D}\left(\underline{\beta}_j^d - \sum_{i \in N} t_j^{id}\right)\right\}$, whereas the variable $v_j^{end} \in \mathbb{R}^+$ will represent $\max\left\{0, \sum_{d \in D}\left(\sum_{i \in N} t_j^{id} + \eta_j - \bar{\beta}_j^d\right)\right\}$. Both variables penalize the services when they are carried out outside their optimal windows.

Constraint (2.28) states that the variable v_j^{start} will be greater than or equal to the difference between the start of the service's optimal time window and its starting time.

$$v_j^{start} \geq \sum_{d \in D}\left(\underline{\beta}_j^d \sum_{i \in N} \sum_{k \in S_{-j}^1} x_{jk}^{id} - \sum_{i \in N} t_j^{id}\right) \qquad \forall j \in S \tag{2.28}$$

$$v_j^{start} \geq 0 \qquad \forall j \in S$$

Notice that Constraint (2.28) works because the variable v_j^{start} is minimized in the objective function.

Three different cases are possible in Constraint 2.28:

a. *If the service starts before the beginning of its optimal time window, which means that* $\underline{\beta}_j^d > \sum_{i \in N} t_j^{id}$, *then* $v_j^{start} = \sum_{d \in D}\left(\underline{\beta}_j^d - \sum_{i \in N} t_j^{id}\right) > 0$.

b. *If the service starts at the beginning of its optimal time window, which means that* $\underline{\beta}_j^d = \sum_{i \in N} t_j^{id}$, *then* $v_j^{start} = 0$.

c. *If the service starts after the beginning of its optimal time window, which means that* $\underline{\beta}_j^d < \sum_{i \in N} t_j^{id}$, *then* $v_j^{start} = 0$.

Constraint (2.29) assures that the variable $v_j^{end} \in \mathbb{R}^+$ will be greater than or equal to the difference between the time at which the service is completed and the ending time of its optimal time window.

$$v_j^{end} \geq \sum_{d \in D} \left(\sum_{i \in N} t_j^{id} + (\eta_j - \bar{\beta}_j^d) \sum_{i \in N} \sum_{k \in S^1_{-j}} x_{jk}^{id} \right) \qquad \forall j \in S \tag{2.29}$$

$$v_j^{end} \geq 0 \qquad \forall j \in S$$

Notice that Constraint (2.29) works because the variable v_j^{end} is minimized in the objective function.

In the case of Constraint (2.29), there are three possibilities:

a. *If the service ends after the ending time of its optimal time window, which means that* $\sum_{i \in N} t_j^{id} + \eta_j > \bar{\beta}_j^d$, *then* $v_j^{end} = \sum_{d \in D} \left(\sum_{i \in N} t_j^{id} + \eta_j - \bar{\beta}_j^d \right) > 0$.

b. *If the service ends at the ending time of its optimal time window, which means that* $\sum_{i \in N} t_j^{id} + \eta_j = \bar{\beta}_j^d$, *then* $v_j^{end} = 0$.

c. *If the service ends before the ending time of its optimal time window, which means that* $\sum_{i \in N} t_j^{id} + \eta_j < \bar{\beta}_j^d$, *then* $v_j^{end} = 0$.

2.1.5 Summary of the MILP model

Next, the model that describes the HCSP under study is presented at a glance.

Sets	
$D = \{1, ..., 7\}$	Set of days.
$N = \{1, ..., n\}$	Set of caregivers.
$S = \{1, ..., s-1\}$	Set of services.
$S^0 = S \cup \{0\}$	Set of services and the initial dummy.
$S^1 = S \cup \{s\}$	Set of services and the ending dummy.
$S^{01} = S \cup \{0, s\}$	Set of services and the initial and ending dummies.
$S_{-k} = S \setminus \{k\}$	Set of services except $k \in S$.
	Analogously, $S^0_{-k} = S^0 \setminus \{k\}$ and $S^1_{-k} = S^1 \setminus \{k\}$.

Table 2.1: Sets involved in the problem.

Data	
ρ_j^i	It indicates if caregiver $i \in N$ can perform service $j \in S$.
λ_j^i	Affinity level between caregiver $i \in N$ and service $j \in S$.
η_j	Duration of service $j \in S$.
$[\underline{\alpha}_j^d, \bar{\alpha}_j^d]$	Hard time window of service $j \in S$ on day $d \in D$.
	Note that, if service $j \in S$ does not belong to day $d \in D$,
	the parameter is $\underline{\alpha}_j^d = \bar{\alpha}_j^d$.
$[\underline{\beta}_j^d, \bar{\beta}_j^d]$	Soft time window of service $j \in S$ on day $d \in D$.
$[\underline{\gamma}^{id}, \bar{\gamma}^{id}]$	Availability time period of caregiver $i \in N$ on day $d \in D$.
θ_{jk}	Travel time between services $j \in S$ and $k \in S^1$.
	Note that, if $k = s$, then $\theta_{js} = 0$.
ν^i	Agreed weekly working time of caregiver $i \in N$.
ν^{id}	Maximum time caregiver $i \in N$ is allowed to work on day $d \in D$.
π_{min}	Minimum length of time required for the largest break to be unpaid.

Table 2.2: Parameters and variables involved in the problem.

Variables	
$x_{jk}^{id} \in \{0,1\}$	It indicates if caregiver $i \in N$ goes from service $j \in S^0$. to service $k \in S^1$ on day $d \in D$.
$t_j^{id} \in \mathbb{R}_0^+$	For caregiver $i \in N$, it represents the starting time of service $j \in S^{01}$ on day $d \in D$.
$y_{jk}^{id} \in \{0,1\}$	For caregiver $i \in N$, it indicates if the break between services $j \in S$ and $k \in S$ has been selected to be discounted from the working day $d \in D$.
$\bar{y}^{id} \in \{0,1\}$	It states if there is no break for caregiver $i \in N$ on day $d \in D$.
$r^{id} \in \mathbb{R}_0^+$	Greatest break of caregiver $i \in N$ on day $d \in D$.
$u^{id} \in \{0,1\}$	It indicates if the largest break of caregiver $i \in N$ on day $d \in D$ is greater than or equal to π_{min}.
$\hat{r}^{id} \in \mathbb{R}_0^+$	Greatest break of caregiver $i \in N$ on day $d \in D$ if it is greater than or equal to π_{min}. Otherwise, it will be 0.
$z^i \in \mathbb{R}_0^+$	Amount of overtime of caregiver $i \in N$.
$v_j^{start} \in \mathbb{R}_0^+$	Penalization for carrying out service $j \in S$ before its soft time window.
$v_j^{end} \in \mathbb{R}_0^+$	Penalization for carrying out service $j \in S$ after its soft time window.

Table 2.3: Variables involved in the problem.

The formulation of the problem is:

$$f_1 = \min\ \omega_1 \sum_{i\in N}\sum_{d\in D}\sum_{j\in S}\sum_{k\in S^1} \lambda_j^i x_{jk}^{id} + \omega_2 \sum_{j\in S}(v_j^{start} + v_j^{end}) \tag{2.1}$$

$$f_2 = \min\ \omega_3 \sum_{i\in N} z^i + \omega_4 \sum_{i\in N}\sum_{d\in D}(t_s^{id} - t_0^{id} - \hat{r}^{id}) \tag{2.2}$$

Subject to

$$\sum_{i\in N}\sum_{d\in D}\sum_{k\in S^1_{-j}} x_{jk}^{id} = 1 \qquad \forall j \in S \tag{2.3}$$

$$\sum_{i\in N}\sum_{d\in D}\sum_{j\in S^0_{-k}} x_{jk}^{id} = 1 \qquad \forall k \in S \tag{2.4}$$

$$\sum_{d\in D}\sum_{k\in S^1_{-j}} x_{jk}^{id} \le \rho_j^i \qquad \forall i \in N, \forall j \in S \tag{2.5}$$

$$\sum_{k\in S^1} x_{0k}^{id} = 1 \qquad \forall i \in N, \forall d \in D \tag{2.6}$$

$$\sum_{j\in S^0} x_{js}^{id} = 1 \qquad \forall i \in N, \forall d \in D \tag{2.7}$$

$$\sum_{j\in S^0_{-h}} x_{jh}^{id} - \sum_{k\in S^1_{-h}} x_{hk}^{id} = 0 \qquad \forall i \in N, \forall d \in D, \forall h \in S \tag{2.8}$$

$$\underline{\alpha}_j^d \sum_{k\in S^1_{-j}} x_{jk}^{id} \le t_j^{id} \le (\bar{\alpha}_j^d - \eta_j) \sum_{k\in S^1_{-j}} x_{jk}^{id} \qquad \forall i \in N, \forall d \in D, \forall j \in S \tag{2.9}$$

$$t_j^{id} + (\eta_j + \theta_{jk}) x_{jk}^{id} \le t_k^{id} + \bar{\alpha}_j^d (1 - x_{jk}^{id}) \qquad \forall i \in N, \forall d \in D, \forall j \in S, \forall k \in S^1, j \ne k \tag{2.10}$$

$$t_0^{id} \ge \underline{\gamma}^{id} \qquad \forall i \in N, \forall d \in D \tag{2.11}$$

$$t_s^{id} \le \bar{\gamma}^{id} \qquad \forall i \in N, \forall d \in D \tag{2.12}$$

$$t_0^{id} \le t_k^{id} + \bar{\gamma}^{id}(1 - x_{0k}^{id}) \qquad \forall i \in N, \forall d \in D, \forall k \in S^1 \tag{2.13}$$

$$t_0^{id} \ge t_k^{id} - \bar{\gamma}^{id}(1 - x_{0k}^{id}) \qquad \forall i \in N, \forall d \in D, \forall k \in S^1 \tag{2.14}$$

$$t_s^{id} \le (t_j^{id} + \eta_j) + \bar{\gamma}^{id}(1 - x_{js}^{id}) \qquad \forall i \in N, \forall d \in D, \forall j \in S \tag{2.15}$$

$$t_s^{id} - t_0^{id} - \hat{r}^{id} \leq \nu^{id} \qquad \forall i \in N, \forall d \in D \tag{2.16}$$

$$z^i \geq \sum_{d \in D} \left(t_s^{id} - t_0^{id} - \hat{r}^{id} \right) - \nu^i \qquad \forall i \in N \tag{2.17}$$

$$r^{id} \geq t_k^{id} - (t_j^{id} + \eta_j + \theta_{jk}) - \bar{\gamma}^{id}(1 - x_{jk}^{id}) \qquad \forall i \in N, \forall d \in D, \forall j \in S, \forall k \in S, j \neq k \tag{2.18}$$

$$r^{id} \leq t_k^{id} - (t_j^{id} + \eta_j + \theta_{jk}) + \bar{\gamma}^{id}(1 - x_{jk}^{id}) + \bar{\gamma}^{id}(1 - y_{jk}^{id}) \qquad \forall i \in N, \forall d \in D, \forall j \in S, \forall k \in S, j \neq k \tag{2.19}$$

$$r^{id} \leq \bar{\gamma}^{id}(1 - \bar{y}^{id}) \qquad \forall i \in N, \forall d \in D \tag{2.20}$$

$$\sum_{j \in S} \sum_{k \in S_{-j}} y_{jk}^{id} + \bar{y}^{id} = 1 \qquad \forall i \in N, \forall d \in D \tag{2.21}$$

$$y_{jk}^{id} \leq x_{jk}^{id} \qquad \forall i \in N, \forall d \in D, \forall j \in S, \forall k \in S, j \neq k \tag{2.22}$$

$$r^{id} - \pi_{min} \geq \pi_{min}(u^{id} - 1) \qquad \forall i \in N, \forall d \in D \tag{2.23}$$

$$r^{id} - \pi_{min} + \varepsilon \leq (\bar{\gamma}^{id} - \underline{\gamma}^{id})u^{id} \qquad \forall i \in N, \forall d \in D \tag{2.24}$$

$$\hat{r}^{id} \leq (\bar{\gamma}^{id} - \underline{\gamma}^{id})u^{id} \qquad \forall i \in N, \forall d \in D \tag{2.25}$$

$$\hat{r}^{id} \leq r^{id} \qquad \forall i \in N, \forall d \in D \tag{2.26}$$

$$\hat{r}^{id} \geq r^{id} - (\bar{\gamma}^{id} - \underline{\gamma}^{id})(1 - u^{id}) \qquad \forall i \in N, \forall d \in D \tag{2.27}$$

$$v_j^{start} \geq \sum_{d \in D} \left(\underline{\beta}_j^d \sum_{i \in N} \sum_{k \in S_{-j}^1} x_{jk}^{id} - \sum_{i \in N} t_j^{id} \right) \qquad \forall j \in S \tag{2.28}$$

$$v_j^{end} \geq \sum_{d \in D} \left(\sum_{i \in N} t_j^{id} + (\eta_j - \bar{\beta}_j^d) \sum_{i \in N} \sum_{k \in S_{-j}^1} x_{jk}^{id} \right) \qquad \forall j \in S \tag{2.29}$$

$$x_{jk}^{id} \in \{0, 1\} \qquad \forall i \in N, \forall d \in D, \forall j \in S^0, \forall k \in S^1, j \neq k$$

$$t_j^{id} \geq 0 \qquad \forall i \in N, \forall j \in S^{01}, \forall d \in D$$

$$y_{jk}^{id} \in \{0, 1\} \qquad \forall i \in N, \forall d \in D, \forall j \in S, \forall k \in S, j \neq k$$

$$\bar{y}^{id}, u^{id} \in \{0, 1\};\ r^{id}, \hat{r}^{id} \geq 0 \qquad \forall i \in N, \forall d \in D$$

$$z^i \geq 0 \qquad \forall i \in N$$

$$v_j^{start}, v_j^{end} \geq 0 \qquad \forall j \in S$$

2.2 Illustrative example of the problem

In this section an illustrative example of the problem under study is presented. It consists of 54 services that two caregivers have to carry out over the course of a week.

First, Table 2.4 describes the characteristics of the services including: the set of tasks assigned to each service, the user demanding the service, the day on which the service should be performed, the time windows and the duration. Then, Table 2.5 shows a description of the tasks to be performed during the services. Finally, the affinity levels between users and the available caregivers are presented in Table 2.6.

Service	Tasks	User	Day	Hard TW	Soft TW	Duration
1	1, 2, 3, 4, 5, 6, 8	1	Mon	08:00 - 11:00	08:00 - 10:00	90
2	3, 8, 9, 10, 11, 12	9	Mon	09.30 - 12:30	09:30 - 11:30	120
3	4, 5, 13	10	Mon	11:30 - 14:00	12:00 - 14:00	60
4	7, 8, 9, 11	11	Mon	12:00 - 14:30	13:30 - 14:30	60
5	8, 9, 13, 14, 15, 16	1	Mon	18:00 - 21:00	18:00 - 20:00	120
6	3, 8, 11	2	Tue	08:00 - 09:00	08:00 - 09:00	60
7	1, 2, 3, 4, 5, 6, 8	1	Tue	08:00 - 11:00	08:00 - 10:00	90
8	4, 7, 10	12	Tue	10:30 - 14:00	10:30 - 11:30	60
9	10, 13	4	Tue	12:00 - 13:30	12:00 - 13:00	45
10	7, 8, 9, 11	11	Tue	12:00 - 14:30	13:30 - 14:30	60
11	8, 9, 13, 14, 15, 16	1	Tue	18:00 - 21:00	18:00 - 20:00	120
12	1, 2, 3, 4, 5, 6, 8	1	Wed	08:00 - 11:00	08:00 - 10:00	90
13	2, 11	3	Wed	09:30 - 10:30	09:30 - 10:30	30
14	3, 8, 9, 10, 11, 12	9	Wed	09.30 - 12:30	09:30 - 11:30	120
15	7, 8, 9, 11	11	Wed	12:00 - 14:30	13:30 - 14:30	60
16	8, 9, 13, 14, 15, 16	1	Wed	18:00 - 21:00	18:00 - 20:00	120
17	3, 8, 11	2	Tue	08:00 - 09:00	08:00 - 09:00	60
18	1, 2, 3, 4, 5, 6, 8	1	Thu	08:00 - 11:00	08:00 - 10:00	90
19	4, 7, 10	12	Thu	10:30 - 14:00	10:30 - 11:30	60
20	10, 13	4	Thu	12:00 - 13:30	12:00 - 13:00	45
21	7, 8, 9, 11	11	Thu	12:00 - 14:30	13:30 - 14:30	60
22	8, 9, 13, 14, 15, 16	1	Thu	18:00 - 21:00	18:00 - 20:00	120
23	1, 2, 3, 4, 5, 6, 8	1	Fri	08:00 - 11:00	08:00 - 10:00	90
24	3, 8, 9, 10, 11, 12	9	Fri	09.30 - 12:30	09:30 - 11:30	120
25	4, 5, 13	10	Fri	11:30 - 14:00	12:00 - 14:00	60
26	7, 8, 9, 11	11	Fri	12:00 - 14:30	13:30 - 14:30	60
27	8, 9, 13, 14, 15, 16	1	Fri	18:00 - 21:00	18:00 - 20:00	120
28	1, 2, 3, 8, 9	13	Mon	06:30 - 08:00	06:30 - 07:30	60
29	3, 8, 9, 10, 12, 13, 14	5	Mon	07:00 - 12:00	09:00 - 11:00	120
30	7, 8	14	Mon	12:00 - 14:00	12:00 - 13:00	30
31	4, 5, 6, 12, 13	6	Mon	16:00 - 20:00	16:00 - 20:00	120
32	4, 5, 6, 10	15	Mon	17:00 - 20:00	17:30 - 19:30	60
33	1, 2, 3, 8, 9	6	Tue	07:00 - 10:00	09:00 - 10:00	60
34	1, 2, 3, 8, 9	13	Tue	08:30 - 10:00	09:00 - 10:00	60
35	4, 5, 6, 11, 12, 14	7	Tue	10:00 - 13:00	10:00 - 12:00	90
36	5, 11	8	Tue	11:30 - 12:30	11:30 - 12:30	30
37	7, 8, 9, 14	16	Tue	11:30 - 14:00	11:30 - 13:30	60
38	7, 8, 9, 12, 11, 14	17	Tue	12:30 - 15:00	11:30 - 13:30	75
39	1, 2, 3, 8, 9	13	Wed	06:30 - 08:00	06:30 - 07:30	60
40	3, 8, 9, 10, 12, 13, 14	5	Wed	07:00 - 12:00	09:00 - 11:00	120
41	7, 8	14	Wed	12:00 - 14:00	12:00 - 13:00	30
42	4, 5, 6, 12, 13	6	Wed	16:00 - 20:00	16:00 - 20:00	120
43	4, 5, 6, 10	15	Wed	17:00 - 20:00	17:30 - 19:30	60
44	1, 2, 3, 8, 9	6	Thu	07:00 - 10:00	09:00 - 10:00	60
45	1, 2, 3, 8, 9	13	Thu	08:30 - 10:00	09:00 - 10:00	60
46	4, 5, 6, 11, 12, 14	7	Thu	10:00 - 13:00	10:00 - 12:00	90
47	5, 11	8	Thu	11:30 - 12:30	11:30 - 12:30	30
48	7, 8, 9, 14	16	Thu	11:30 - 14:00	11:30 - 13:30	60
49	7, 8, 9, 11, 12, 14	17	Tue	12:30 - 15:00	11:30 - 13:30	75
50	1, 2, 3, 8, 9	13	Fri	06:30 - 08:00	06:30 - 07:30	60
51	3, 8, 9, 10, 12, 13, 14	8	Fri	07:00 - 12:00	09:00 - 11:00	120
52	7, 8	14	Fri	12:00 - 14:00	12:00 - 13:00	30
53	4, 5, 6, 12, 13	6	Fri	16:00 - 20:00	16:00 - 20:00	120
54	4, 5, 6, 10	15	Fri	17:00 - 20:00	17:30 - 19:30	60

Table 2.4: List of services.

Task	Description	Task	Description
1	Get out of bed	9	Do the dishes
2	Make beds	10	Do the laundry
3	Cook breakfast	11	Cleaning
4	Bathe	12	Change diaper
5	Wash hair	13	Go for a walk
6	Dress/undress	14	Brushing teeth/dentures
7	Cook lunch	15	Cook dinner
8	Feed	16	Put to bed

Table 2.5: List of tasks.

User	Affinity Caregiver 1	Affinity Caregiver 2
1	5	2
2	2	2
3	5	2
4	4	1
5	4	4
6	2	5
7	2	4
8	4	2
9	4	2
10	5	4
11	4	2
12	2	4
13	4	5
14	1	5
15	2	5
16	2	5
17	2	2

Table 2.6: List of affinities.

The weekly schedules of the caregivers are shown in Figures 2.1 and 2.2. Each column represents the day schedule, the travel times and the breaks (the light ones are considered as working time, whereas the dark ones are unpaid breaks). Each box includes the following information: hard time window (top left), soft time window (top right), service and user (middle) and the scheduled start and end time (bottom).

Analyzing the cost of the schedule for Caregiver 1, it can be seen that the schedule of Monday is repeated on Friday[3]. The same happens with Tuesday and Thursday. Since every day there is only one break of more than two hours, they will be subtracted from their corresponding working hours. In this way, the fifteen minute break in Tuesday and Thursday belongs to the working day. In terms of Caregiver 2, the schedule of Monday is the same as the ones on Wednesday and Friday. During those days, Caregiver 2 has two breaks that last more than two hours, which means that only the largest one of them will not be paid. The schedule of Tuesday is repeated on Thursday and, in this case, Caregiver 2 has no breaks.

According to the affinity levels, all services are being carried out by their preferred caregivers except the ones of Users 12 and 8. These users are currently assigned to a caregiver with whom they have an affinity level of 2, but their affinity with the caregiver not attending them is 4. In terms of the soft time window penalization, some services are carried out within their soft time

[3]Recurrent tasks are carried out at the same time.

window (e.g. Services 6, 17, 35, 46 and 13), while others start before (e.g. Services 3, 4, 15, 25 and 26) or end after (e.g. Services 7, 8, 18 and 19) their soft time window.

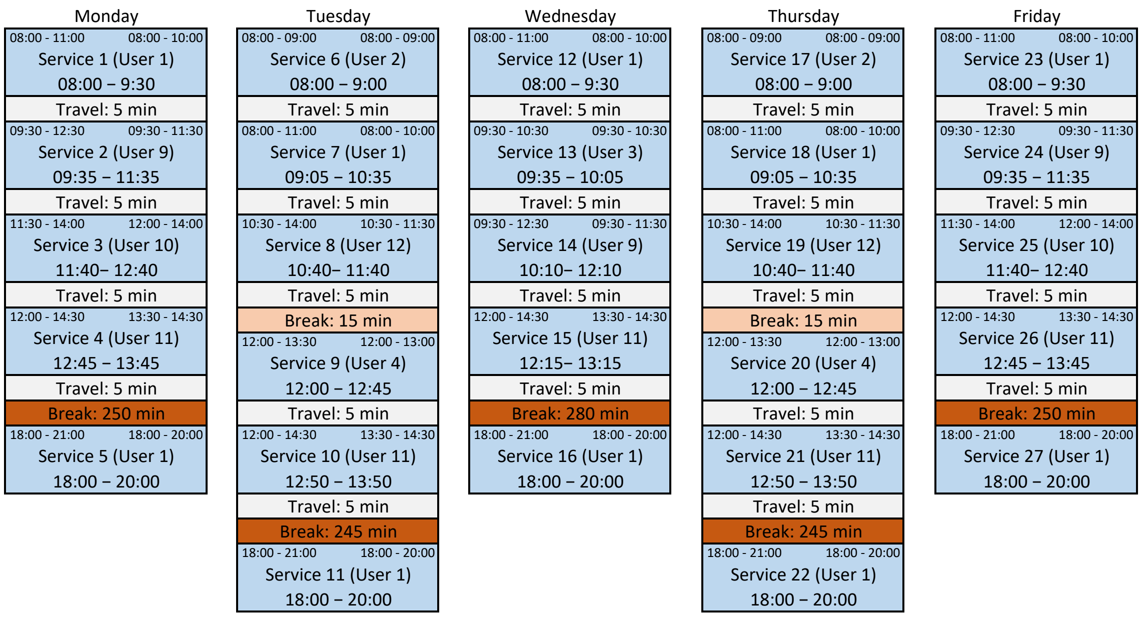


Figure 2.1: Schedules of Caregiver 1.

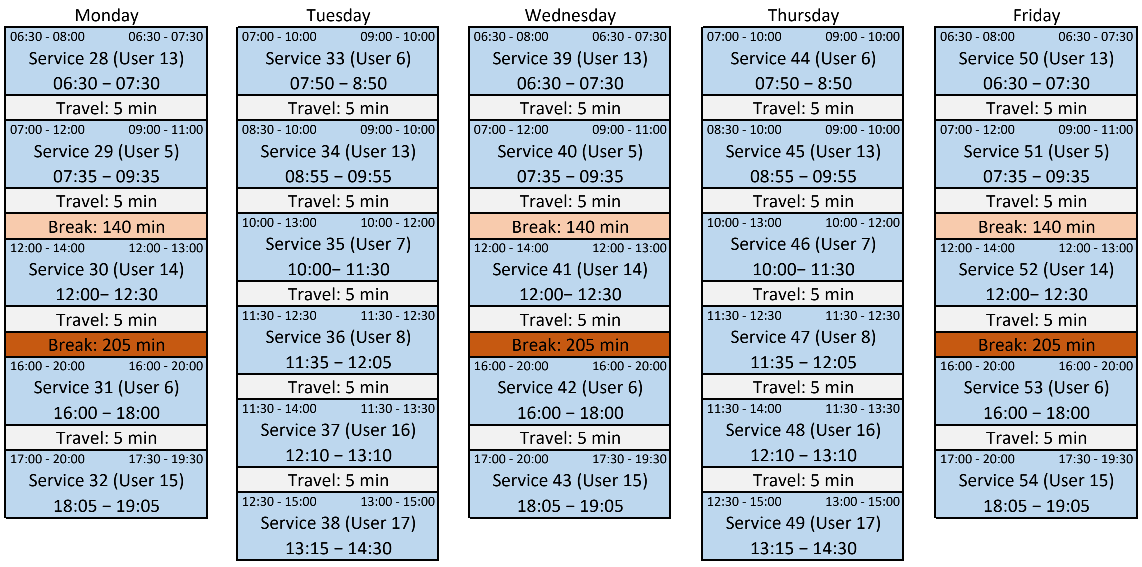


Figure 2.2: Schedules of Caregiver 2.

The objective function values, in minutes, are shown in Table 2.7. Both schedules are similar in terms of affinity, soft time penalization and working time, even though Caregiver 2 has 50 minutes of overtime (if the agreed weekly working time is 40 hours).

	Affinity	STW penalization	Overtime	Worked time
Caregiver 1	113	425	0	2330
Caregiver 2	118	405	50	2450
Total	231	830	50	4780

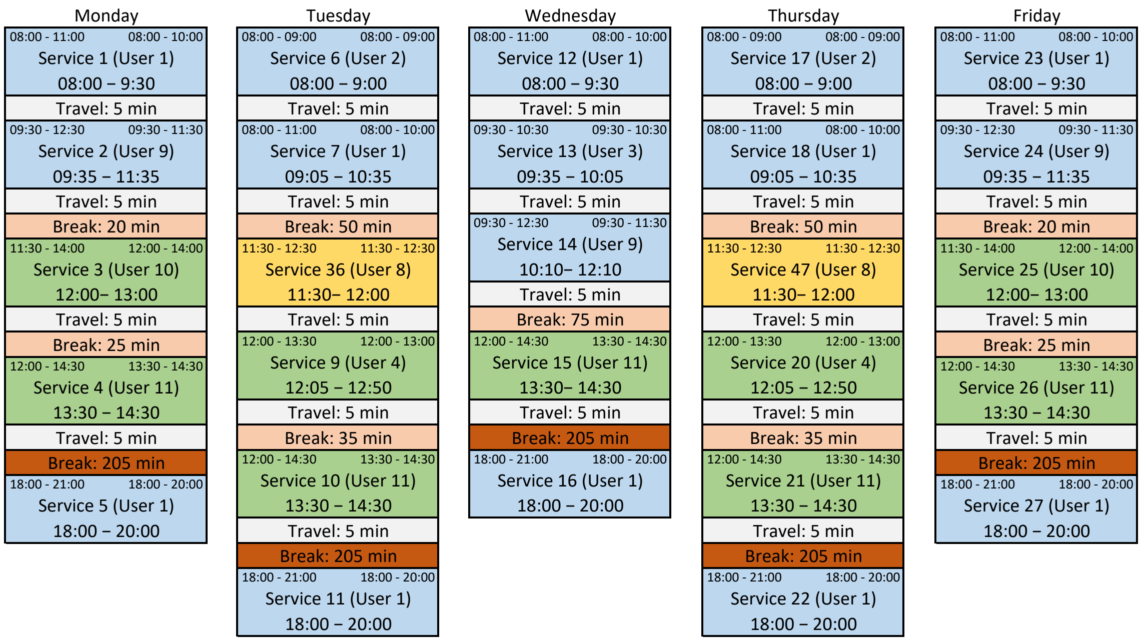
Table 2.7: Objective function values.

The schedule can be modified in order to improve the affinity and soft time window penalization. Figures 2.3 and 2.4 present the new schedules and Table 2.8 their objective function values.

	Affinity	STW penalization	Overtime	Worked time
Caregiver 1	117	120	175	2575
Caregiver 2	122	140	50	2440
Total	239	260	225	5015

Table 2.8: Improved objective function values.

The overall affinity is improved by interchanging Services 8, 19, 36 and 47 between both caregivers (see yellow boxes). As far as the soft time window penalization is concerned, its value is greatly reduced by modifying the schedule of the services carried out outside their soft time window (see green boxes). However, it will possibly imply an increment of the duration of some breaks. For instance, the delay of Services 3 and 4 on Monday, creates two new breaks of 20 and 25 minutes for Caregiver 1, which are not deducted from the journey. This type of modifications in the schedule can increase the overtime, as it happens in this case for Caregiver 1, resulting now in 175 minutes of overtime.



Monday	Tuesday	Wednesday	Thursday	Friday
08:00 - 11:00 08:00 - 10:00 Service 1 (User 1) 08:00 – 9:30	08:00 - 09:00 08:00 - 09:00 Service 6 (User 2) 08:00 – 9:00	08:00 - 11:00 08:00 - 10:00 Service 12 (User 1) 08:00 – 9:30	08:00 - 09:00 08:00 - 09:00 Service 17 (User 2) 08:00 – 9:00	08:00 - 11:00 08:00 - 10:00 Service 23 (User 1) 08:00 – 9:30
Travel: 5 min	Travel: 5 min	Travel: 5 min	Travel: 5 min	Travel: 5 min
09:30 - 12:30 09:30 - 11:30 Service 2 (User 9) 09:35 – 11:35	08:00 - 11:00 08:00 - 10:00 Service 7 (User 1) 09:05 – 10:35	09:30 - 10:30 09:30 - 10:30 Service 13 (User 3) 09:35 – 10:05	08:00 - 11:00 08:00 - 10:00 Service 18 (User 1) 09:05 – 10:35	09:30 - 12:30 09:30 - 11:30 Service 24 (User 9) 09:35 – 11:35
Travel: 5 min	Travel: 5 min	Travel: 5 min	Travel: 5 min	Travel: 5 min
Break: 20 min	Break: 50 min	09:30 - 12:30 09:30 - 11:30 Service 14 (User 9) 10:10– 12:10	Break: 50 min	Break: 20 min
11:30 - 14:00 12:00 - 14:00 Service 3 (User 10) 12:00– 13:00	11:30 - 12:30 11:30 - 12:30 Service 36 (User 8) 11:30– 12:00	Travel: 5 min	11:30 - 12:30 11:30 - 12:30 Service 47 (User 8) 11:30– 12:00	11:30 - 14:00 12:00 - 14:00 Service 25 (User 10) 12:00– 13:00
Travel: 5 min	Travel: 5 min	Break: 75 min	Travel: 5 min	Travel: 5 min
Break: 25 min	12:00 - 13:30 12:00 - 13:00 Service 9 (User 4) 12:05 – 12:50	12:00 - 14:30 13:30 - 14:30 Service 15 (User 11) 13:30– 14:30	12:00 - 13:30 12:00 - 13:00 Service 20 (User 4) 12:05 – 12:50	Break: 25 min
12:00 - 14:30 13:30 - 14:30 Service 4 (User 11) 13:30 – 14:30	Travel: 5 min	Travel: 5 min	Travel: 5 min	12:00 - 14:30 13:30 - 14:30 Service 26 (User 11) 13:30 – 14:30
Travel: 5 min	Break: 35 min	Break: 205 min	Break: 35 min	Travel: 5 min
Break: 205 min	12:00 - 14:30 13:30 - 14:30 Service 10 (User 11) 13:30 – 14:30	18:00 - 21:00 18:00 - 20:00 Service 16 (User 1) 18:00 – 20:00	12:00 - 14:30 13:30 - 14:30 Service 21 (User 11) 13:30 – 14:30	Break: 205 min
18:00 - 21:00 18:00 - 20:00 Service 5 (User 1) 18:00 – 20:00	Travel: 5 min		Travel: 5 min	18:00 - 21:00 18:00 - 20:00 Service 27 (User 1) 18:00 – 20:00
	Break: 205 min		Break: 205 min	
	18:00 - 21:00 18:00 - 20:00 Service 11 (User 1) 18:00 – 20:00		18:00 - 21:00 18:00 - 20:00 Service 22 (User 1) 18:00 – 20:00	

Figure 2.3: Improved schedules of Caregiver 1.

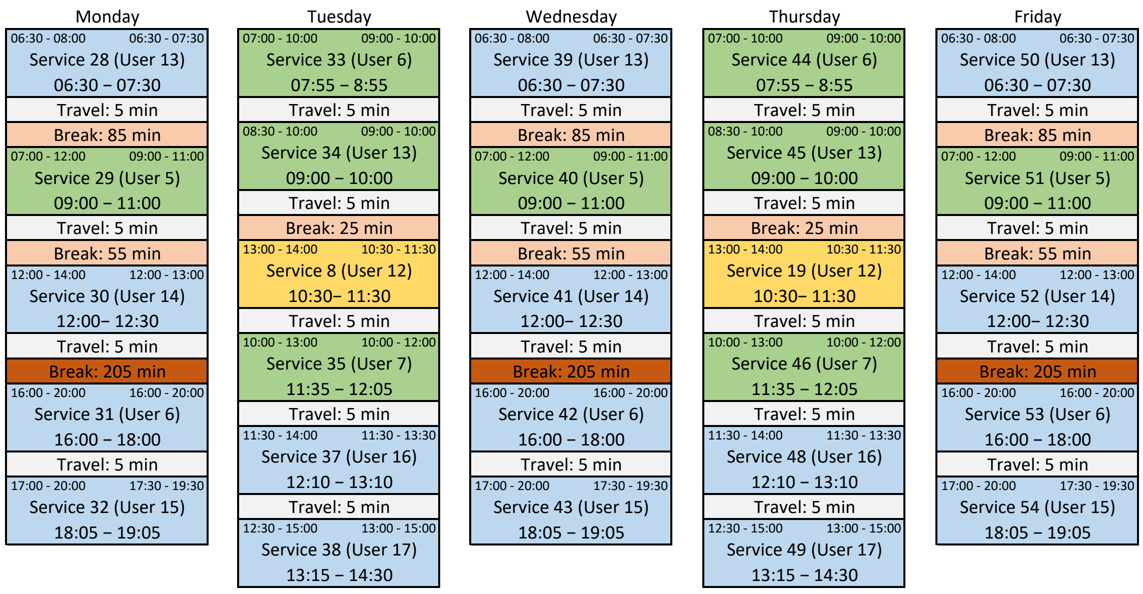


Figure 2.4: Improved schedules of Caregiver 2.

2.3 Resolution approaches

Since the MILP formulation of the HCSP problem has two different objective functions, three different approaches can be considered to solve it.

First approach. In the first approach, the welfare of users is prioritized over the cost of the schedule. This is the option that the company would use in the areas where they are already operating, since the priority is the satisfaction of users. Furthermore, if the company were conservative with the initial schedules, the soft time windows of the services could be adjusted to the previous schedules.

Second approach. The second approach consists in prioritizing the cost of the schedule over users' welfare. This option would be useful for the company when they want to start working on a new area, which means that there are no previous schedules and the users preferences are not a priority. In fact, the goal of the company would be to obtain the cheapest and most efficient schedules, so they would not need to hire too many caregivers.

Third approach. Finally, it seems natural to study the biobjective version of the problem. Studying the two objectives at the same time results in a set of different solutions, in such a way that supervisors can choose a solution to work with by analyzing the trade-off between cost and welfare.

Chapter 3

The metaheuristic algorithm

The HCSP is a generalization of the Vehicle Routing Problem (VRP), which is NP-hard (see Lenstra & Kan (1981)). Therefore, the MILP formulation can only be solved in small instances with a state of the art optimization solver. For this reason, a tailored algorithm was designed, based on the Adaptive Large Neighborhood Search (ALNS) method proposed by Ropke & Pisinger (2006). This methodology, which is based on using destroy and repair operators, has been proved to be more successful for solving different types of vehicle routing problems than other traditional methods (for more information see Pisinger & Ropke (2007)). A complete description of the ALNS is presented in this chapter, providing examples of how the considered operators work.

This chapter is organized as follows. Section 3.1 describes the ALNS methodology. After that, the procedure followed to obtain the initial solution is presented in Section 3.2. The removal and insertion operators are introduced in Sections 3.3 and 3.4, respectively. Finally, the problem of obtaining the schedule of a route is addressed in Section 3.5, focusing on tackling the problem with Constraint Programming.

3.1 Adaptive Large Neighborhood Search method

In this section, the ALNS is introduced, which is a metaheuristic algorithm based on destroy and repair neighborhoods.

Let us consider a minimization problem such as

$$\begin{aligned} \min_{\boldsymbol{\omega}} \quad & f(\boldsymbol{\omega}) \\ \text{s.t.} \quad & \boldsymbol{\omega} \in \Omega \end{aligned} \tag{3.1}$$

where $f : \Omega \longrightarrow \mathbb{R}$ is the objective function to be optimized (minimized, in this case) and Ω is the set of feasible solutions.

The pseudocode of the ALNS method is described in Algorithm 3.1. The input data of the algorithm are: the initial solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$), the objective function (f), the set of removal operators (Σ_{rem}) and the set of insertion operators (Σ_{ins}). Note that a solution $\boldsymbol{\omega}$ is composed by $\boldsymbol{x}$ (where $\boldsymbol{x} = (x_{jk}^{id}), \forall i \in N, \forall d \in D, \forall j \in S^0, \forall k \in S^1, j \neq k$), that defines the routes, and $\boldsymbol{t}$ (where $\boldsymbol{t} = (t_j^{id}), \forall i \in N, \forall j \in S^{01}, \forall d \in D$), that specifies the starting times of the services.

The method starts by initializing the weights of the removal (σ_{rem}) and insertion (σ_{ins}) operators. Additionally, the best solution ($\boldsymbol{\omega}'$) is set as the initial one (line 1). Then, until a stopping criteria is not met, the solution is destroyed and repaired in order to improve the objective function value.

First, the removal ($\varsigma_{rem} \in \Sigma_{rem}$) and insertion ($\varsigma_{ins} \in \Sigma_{ins}$) operators are selected at random according to the weights σ_{rem} and σ_{rem}, respectively (lines 3 - 4). After that, using the operators previously selected, the current solution $\boldsymbol{\omega}$ is destroyed, obtaining $\hat{\boldsymbol{\omega}}$, and repaired creating a new solution $\boldsymbol{\omega}^*$ (lines 5 - 6). The best solution ($\boldsymbol{\omega}'$) is updated if the new one ($\boldsymbol{\omega}^*$) improves its objective function value (lines 7 - 10). The current solution is updated choosing between the new ($\boldsymbol{\omega}^*$) and the best ($\boldsymbol{\omega}'$) one according to an acceptance criteria that allows to select worse solutions in order to diversify the search (line 11). Finally, the weights of the removal (σ_{rem}) and insertion (σ_{ins}) operators are updated, in order to choose more frequently the ones that result in better solutions (line 12).

Algorithm 3.1: ALNS - Adaptive Large Neighborhood Search

```
   Data: the initial solution ($\boldsymbol{\omega} = (\boldsymbol{x},\boldsymbol{t})$), the objective function ($f$), the removal operators
         ($\Sigma_{rem}$) and the insertion operators ($\Sigma_{ins}$)
 1 $\sigma_{rem} \leftarrow (1, ..., 1)$, $\sigma_{ins} \leftarrow (1, ..., 1)$, $\boldsymbol{\omega}' \leftarrow \boldsymbol{\omega}$
   //Improve the solution
 2 while stopping criteria not met do
       //Get removal and insertion operators
 3     $\varsigma_{rem} \leftarrow$ chooseRandom($\sigma_{rem}, \Sigma_{rem}$)
 4     $\varsigma_{ins} \leftarrow$ chooseRandom($\sigma_{ins}, \Sigma_{ins}$)
       //Obtain new solution
 5     $\hat{\boldsymbol{\omega}} \leftarrow$ destroySolution($\boldsymbol{\omega}$, $\varsigma_{rem}$, $f$)
 6     $\boldsymbol{\omega}^* \leftarrow$ repairSolution($\hat{\boldsymbol{\omega}}$, $\varsigma_{ins}$, $f$)
       //Update best solution
 7     if $f(\boldsymbol{\omega}^*) < f(\boldsymbol{\omega}')$ then
 8         $\boldsymbol{\omega}' \leftarrow \boldsymbol{\omega}^*$, $\tau \leftarrow$ true
 9     else
10         $\tau \leftarrow$ false
       //Update current solution
11     $\boldsymbol{\omega} \leftarrow$ acceptanceCriteria($\boldsymbol{\omega}^*$, $\boldsymbol{\omega}'$)
       //Update the weights of the operators
12     $\sigma_{rem} \leftarrow$ updateWeights($\sigma_{rem}$, $\tau$)
13     $\sigma_{ins} \leftarrow$ updateWeights($\sigma_{ins}$, $\tau$)
14 return $\boldsymbol{\omega}$
```

The criteria to update the current solution is based on the probability $prob = exp(-(f(\boldsymbol{\omega}^*) - f(\boldsymbol{\omega}'))/T_i)$, where T_i is a temperature that decreases after each iteration i according to: a cooling parameter $0 < \beta < 1$, an initial temperature $T_0 > 0$ and the formula $T_i = \beta^i T_0$. Therefore, the new solution will be the chosen one with probability *prob*. The weights of the selected operators are increased if the new solution improves the best one so far.

3.2 Initial solution

The initial solution is obtained by iteratively using one of the insertion operators, which will be introduced in Section 3.4, to add services to the solution (described in Algorithm 3.2). First, an empty solution, with no services scheduled, is generated. Then, the services are added according to the insertion operators. The method stops when either all services have been scheduled, or no more services can be added to the solution.

Algorithm 3.2: Initial solution

```
Data: the insertion operator ($\varsigma_{ins}$)
1 $\boldsymbol{\omega} \leftarrow$ emptySolution()
2 while stopping criteria not met do
      //Add a service to the solution
3     $\boldsymbol{\omega} \leftarrow$ repairSolution($\boldsymbol{\omega}$, $\varsigma_{ins}$, $f$)
4 return $\omega$
```

3.3 Removal Operators

The removal operators are used to destroy part of the solution by removing a percentage p of services from the caregivers routes. This section describes the removal operators used to destroy the solution:

- Random removal
- Related removal
- Cost removal
- 1-route removal
- 2-route removal

3.3.1 Random removal

The random removal operator, described in Algorithm 3.3, selects the services that will be deleted from the solution at random following a uniform distribution. The method ends when a percentage of services has been removed from the solution.

Algorithm 3.3: Random removal operator

```
Data: the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$), the percentage of services to be removed ($p$) and the set of
      services ($S$)
  //Get proportion of not scheduled services
1 $S_u \leftarrow$ unscheduledServices($\boldsymbol{\omega}, S$) //Get unscheduled services
2 $S_s \leftarrow$ scheduledServices($\boldsymbol{\omega}, S$)//Get scheduled services
3 $p' \leftarrow |S_u|/|S|$ //Get proportion of unscheduled services
  //Destroy the solution
4 while $p' < p$ do
5     $j \leftarrow$ chooseRandomService($S_s$) //Get a random service
6     $d \leftarrow$ day($j$), $i \leftarrow$ caregiver($j$) //Get caregiver and day of the service
7     $x_{lj}^{id} \leftarrow 0$, $x_{jl}^{id} \leftarrow 0$, $\forall l \in S^{01}$ //Remove the service from the route
8     $t_j^{id} \leftarrow 0$ //Unschedule the service
      //Update the proportion of not scheduled services
9     $S_u \leftarrow$ unscheduledServices($\boldsymbol{\omega}, S$) //Get unscheduled services
10    $p' \leftarrow |S_u|/|S|$
11 return $\boldsymbol{\omega}$
```

Example 3.3.1. Illustration of random removal operator.

Let us consider the route presented in Figure 3.1. The scheduled services have a duration of 90 minutes[1] and the travel times between them are: $\theta_{1,2} = 5$, $\theta_{1,3} = 15$, $\theta_{1,4} = 30$, $\theta_{2,3} = 10$, $\theta_{2,4} = 25$ and $\theta_{3,4} = 15$[2] (the grey area after the services in Figure 3.1). Furthermore, the percentage of services to be removed from the solution is 25% ($p = 0.25$).

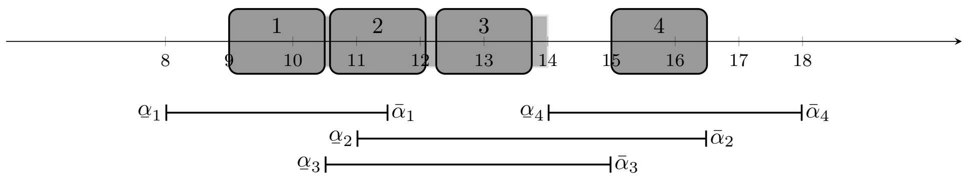


Figure 3.1: Route to destroy.

In this case, $S_u = \emptyset$, $S_s = \{1, 2, 3, 4\}$ and $p' = |S_u|/|S| = 0$. Service $j = 2$ is chosen at random and it is removed from the route (see Figure 3.2). After that, $S_u = \{2\}$ and $p' = |S_u|/|S| = 0.25 \geq p$, so there is no need to remove more services from the solution.

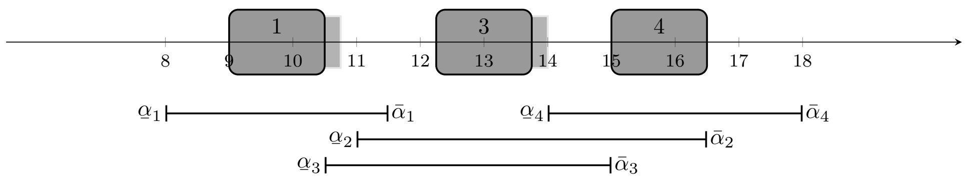


Figure 3.2: Remove service 2.

3.3.2 Related removal

The related removal operator, described in Algorithm 3.4, deletes from the route the services that are most related between them. The level of relation between two services is measured by the overlapping of their time windows and the day they belong to.

The first service to be removed is randomly selected, following a uniform distribution (lines 1 - 2). Then, the algorithm chooses an unscheduled service at random (line 5) and iterates through the scheduled services to obtain their relation level (lines 6 - 14). Finally, the service with the best related value is removed from the schedule. The algorithm ends when the predefined percentage of solution has been destroyed.

[1]For simplicity, the example is presented in minutes. Therefore, the 8:00 a.m. will correspond to minute 0. Note that, 8:00 a.m. is time 0 because it is when the earliest time window of the services starts.

[2]Note that, even though in the definition of the parameters the notation used is θ_{jk}, to avoid possible confusions, the notation $\theta_{j,k}$ will be considered in the example.

Algorithm 3.4: Related removal operator

```
Data: the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$), the percentage of services to be removed ($p$) and the set of
      services ($S$)
1  if $S_u = \emptyset$ then
       //Remove service at random
2      $j \leftarrow$ chooseRandomService($S_s$), $\boldsymbol{\omega} \leftarrow$ removeService($j, \boldsymbol{\omega}$)
   //Get proportion of not scheduled services
3  $S_u \leftarrow$ unscheduledServices($\boldsymbol{\omega}, S$), $S_s \leftarrow$ scheduledServices($\boldsymbol{\omega}, S$), $p' \leftarrow |S_u|/|S|$
   //Destroy the solution
4  while $p' < p$ do
       //Get an unscheduled service at random and its day
5      $k \leftarrow$ chooseRandomService($S_u$), $d_k \leftarrow$ day($k$)
       //Get relation with first scheduled service, $j$
6      $j \leftarrow$ firstService($S_s$), $d_j \leftarrow$ day($j$), $rel_j \leftarrow \theta_{j,k} + |\underline{\alpha}_j^{d_j} - \underline{\alpha}_k^{d_k}| + |\bar{\alpha}_j^{d_j} - \bar{\alpha}_k^{d_k}|$
7      if $d_j \neq d_k$ then
8          $rel_j \leftarrow rel_j + 1$ //Increase the relation if the services belong to the same day
       //Get most related service
9      for $l \in S_s \setminus \{j\}$ do
           //Get relation with the other services
10         $d_l \leftarrow$ day($l$), $rel_l \leftarrow \theta_{l,k} + |\underline{\alpha}_l^{d_l} - \underline{\alpha}_k^{d_k}| + |\bar{\alpha}_l^{d_l} - \bar{\alpha}_k^{d_k}|$
11         if $d_l \neq d_k$ then
12             $rel_l \leftarrow rel_l + 1$
           //Update most related service
13         if $rel_l < rel_j$ then
14             $j \leftarrow l$, $rel_j \leftarrow rel_l$
       //Remove the most related service from the schedule
15     $d \leftarrow$ day($j$), $i \leftarrow$ caregiver($j$) //Get caregiver and day of the service
16     $x_{l,j}^{i,d} \leftarrow 0$, $x_{j,l}^{i,d} \leftarrow 0$, $\forall l \in S^{01}$ //Remove the service from the route
17     $t_j^{i,d} \leftarrow 0$ //Unschedule the service
       //Update the proportion of not scheduled services
18     $S_u \leftarrow$ unscheduledServices($\boldsymbol{\omega}, S$)
19     $p' \leftarrow |S_u|/|S|$
20 return $\boldsymbol{\omega}$
```

Example 3.3.2. Illustration of related removal operator.

Let us consider the route presented in Figure 3.1. The hard time windows of the services are: $\underline{\alpha}_1^d = 0$, $\bar{\alpha}_1^d = 210$, $\underline{\alpha}_2^d = 180$, $\bar{\alpha}_2^d = 510$, $\underline{\alpha}_3^d = 150$, $\bar{\alpha}_3^d = 420$, $\underline{\alpha}_4^d = 360$ and $\bar{\alpha}_4^d = 600$. Let us suppose that the percentage of services to be removed from the solution is 50% ($p = 0.5$).

Now, $S_u = \emptyset$, $S_s = \{1, 2, 3, 4\}$ and $p' = |S_u|/|S| = 0$. Service $j = 3$ is chosen at random and it is removed from the route (see Figure 3.3).

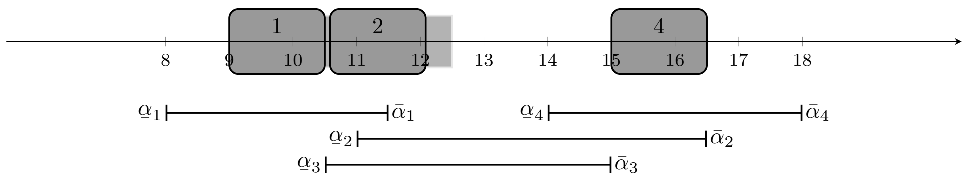


Figure 3.3: Remove Service 3 at random.

So, $S_u = \{3\}$ and $p' = |S_u|/|S| = 0.25 < p$. Therefore, another service needs to be removed from the solution. To choose the service to be removed, it is necessary to obtain the relation level between all scheduled services (1, 2 and 4) and the unscheduled one (3):

Service 1. $rel_1 = \theta_{1,3} + |\underline{\alpha}_1^{d_1} - \underline{\alpha}_3^{d_3}| + |\bar{\alpha}_1^{d_1} - \bar{\alpha}_3^{d_3}| = 15 + |0 - 150| + |210 - 420| = 375.$

Service 2. $rel_2 = \theta_{2,3} + |\underline{\alpha}_2^{d_2} - \underline{\alpha}_3^{d_3}| + |\bar{\alpha}_2^{d_2} - \bar{\alpha}_3^{d_3}| = 10 + |180 - 150| + |510 - 420| = 130.$

Service 4. $rel_4 = \theta_{4,3} + |\underline{\alpha}_4^{d_4} - \underline{\alpha}_3^{d_3}| + |\bar{\alpha}_4^{d_4} - \bar{\alpha}_3^{d_3}| = 15 + |360 - 150| + |600 - 420| = 405.$

The service most related to Service 3 is Service 2. Therefore it is removed from the schedule (see Figure 3.4).

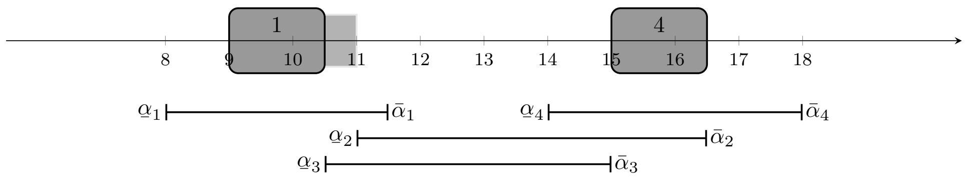


Figure 3.4: Route obtained after removing Service 2.

3.3.3 Cost removal

The cost removal operator, described in Algorithm 3.5, iteratively deletes from the routes the service that contributes the most to the objective function value.

The algorithm iterates through the scheduled services, removes them from the solution and obtains the cost of the solution without the service (lines 5 - 9). After that, the removed service giving the minimum cost, is the one that will be permanently deleted from the solution (line 10). The algorithm ends when a certain percentage of solution has been destroyed.

Algorithm 3.5: Cost removal operator

```
Data: the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$), the percentage of services to be removed ($p$), the set of
      services ($S$) and the objective function ($f$)
   //Get the proportion of not scheduled services
1  $S_u \leftarrow$ unscheduledServices($\boldsymbol{\omega}, S$)
2  $S_s \leftarrow$ scheduledServices($\boldsymbol{\omega}, S$)
3  $p' \leftarrow |S_u|/|S|$
   //Destroy the solution
4  while $p' < p$ do
       //Get the service that contributes the most to $f$
5      for $l \in S_s$ do
6          $d \leftarrow$ day($l$), $i \leftarrow$ caregiver($l$) //Get the caregiver and day of the service
7          $\bar{\boldsymbol{x}} \leftarrow \boldsymbol{x}$, $\bar{x}_{kl}^{id} \leftarrow 0$, $\bar{x}_{lk}^{id} \leftarrow 0$, $\forall k \in S^{01}$ //Remove the service from the route
8          $\bar{\boldsymbol{t}} \leftarrow \boldsymbol{t}$, $\bar{t}_l^{id} \leftarrow 0$ //Unschedule the service
9          $\Omega \leftarrow \Omega \cup \{(\bar{\boldsymbol{x}}, \bar{\boldsymbol{t}})\}$ //Solution without the service
10     $\boldsymbol{\omega} \leftarrow \{\hat{\boldsymbol{\omega}} / \boldsymbol{f}(\hat{\boldsymbol{\omega}}) = \min_{\bar{\boldsymbol{\omega}} \in \Omega}\{\boldsymbol{f}(\bar{\boldsymbol{\omega}})\}\}$ //Solution with the best value of $f$
       //Update the proportion of not scheduled services
11     $S_u \leftarrow$ unscheduledServices($\boldsymbol{\omega}, S$)
12     $p' \leftarrow |S_u|/|S|$
13 return $\boldsymbol{\omega}$
```

Example 3.3.3. Illustration of cost removal operator.

Let us consider the route presented in Figure 3.1. The starting times of the services are: $t_1 = 60$, $t_2 = 155$, $t_3 = 255$ and $t_4 = 420$. Furthermore, the proportion of solution to destroy is 25% and the objective function considered is the working time of the caregiver.

The cost of the schedule when removing Service 1 (see Figure 3.5) is $t_4+\eta_4-t_2 = 420+90-155 = 355$.

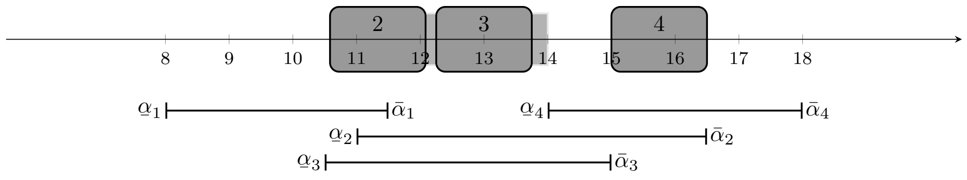


Figure 3.5: Route obtained removing Service 1.

The cost of the schedule when removing Service 2 (see Figure 3.6) is $t_4+\eta_4-t_1 = 420+90-60 = 450$.

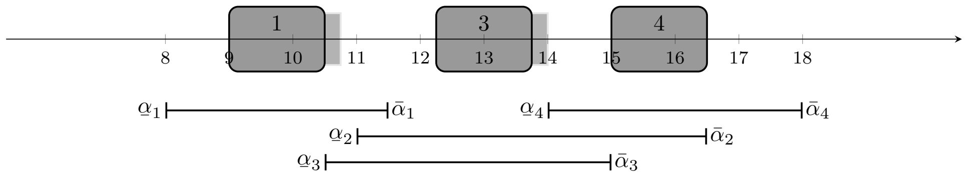


Figure 3.6: Route obtained removing Service 2.

The cost of the schedule when removing Service 3 (see Figure 3.7) is $t_4 + \eta_4 - t_1 - (t_4 - (t_2 + \eta_2 + \theta_{2,4})) = 420 + 90 - 60 - (420 - (155 + 90 + 25)) = 300$ because, in this case, removing Service 3 results in a break that lasts more than 2 hours.

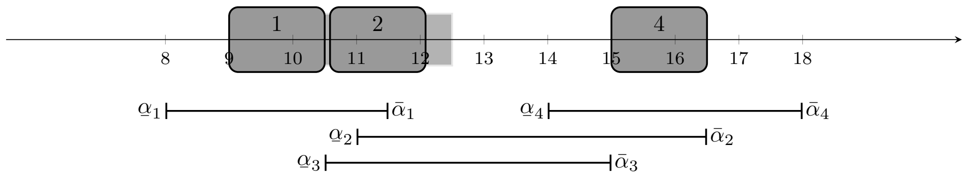


Figure 3.7: Route obtained removing Service 3.

The cost of the schedule when removing Service 4 (see Figure 3.8) is $t_3+\eta_3-t_1 = 255+90-60 = 285$.

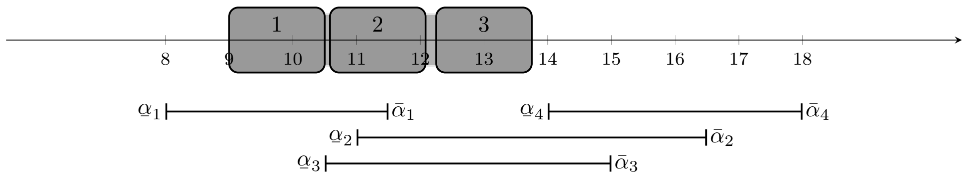


Figure 3.8: Route obtained removing Service 4.

Therefore, the service to be removed is Service 4, because is the one that results in the bigger reduction of cost.

3.3.4 1-route removal

The 1-route removal operator, described in Algorithm 3.6, deletes complete routes from the solution until at least the required number of services have been removed.

The algorithm randomly selects a route (caregiver and day), according to a uniform distribution, and removes its services from the solution (lines 5 - 9). If the required proportion of services has been removed then the algorithm stops. Otherwise, the method removes another route.

Algorithm 3.6: 1-route removal operator

```
Data: the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$), the percentage of services to be removed ($p$) and the set of
      services ($S$)
//Get the proportion of not scheduled services
1  $S_u \leftarrow$ unscheduledServices($\boldsymbol{\omega}, S$)
2  $S_s \leftarrow$ scheduledServices($\boldsymbol{\omega}, S$)
3  $p' \leftarrow |S_u|/|S|$
//Destroy the solution
4  while $p' < p$ do
       //Remove a route from the solution
5      $i \leftarrow$ getCaregiver($\boldsymbol{\omega}$), $d \leftarrow$ getDay($\boldsymbol{\omega}$) //Get a random caregiver and a day
6      $S_i \leftarrow$ getServices($\boldsymbol{\omega}, S_s, i, d$) //Get the scheduled services assigned to the caregiver
7      for $l \in S_i$ do
8          $x^{id}_{kl} \leftarrow 0$, $x^{id}_{lk}, \leftarrow 0 \; \forall k \in S^{01}$ //Remove the service from the route
9          $t^{id}_{l} \leftarrow 0$ //Unschedule the service
       //Update the proportion of not scheduled services
10     $S_u \leftarrow$ unscheduledServices($\boldsymbol{\omega}, S$)
11     $p' \leftarrow |S_u|/|S|$
12 return $\boldsymbol{\omega}$
```

3.3.5 2-route removal

The 2-route removal operator, described in Algorithm 3.7, deletes two complete routes selected at random, following a uniform distribution, from the solution.

Algorithm 3.7: 2-route removal operator

```
Data: the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$) and the set of services ($S$)
//Get two random routes (caregiver and day)
1  $i_1 \leftarrow$ getCaregiver($\boldsymbol{\omega}$), $d_1 \leftarrow$ getDay($\boldsymbol{\omega}$)
2  $i_2 \leftarrow$ getCaregiver($\boldsymbol{\omega}$), $d_2 \leftarrow$ getDay($\boldsymbol{\omega}$)
//Get the scheduled services assigned to the caregivers
3  $S_{i_1} \leftarrow$ getServices($\boldsymbol{\omega}, S_s, i_1, d_1$), $S_{i_2} \leftarrow$ getServices($\boldsymbol{\omega}, S_s, i_2, d_2$)
//Remove the services
4  for $l \in S_{i_1}$ do
5      $x^{i_1 d_1}_{kl} \leftarrow 0$, $x^{i_1 d_1}_{lk}, \leftarrow 0 \; \forall k \in S^{01}$ //Remove the service from the route
6      $t^{i_1 d_1}_{l} \leftarrow 0$ //Unschedule the service
7  for $l \in S_{i_2}$ do
8      $x^{i_2 d_2}_{k,l} \leftarrow 0$, $x^{i_2 d_2}_{lk}, \leftarrow 0 \; \forall k \in S^{01}$ //Remove the service from the route
9      $t^{i_2 d_2}_{l} \leftarrow 0$ //Unschedule the service
10 return $\boldsymbol{\omega}$
```

3.4 Insertion Operators

The insertion operators are used to restore the solution by introducing the services back into the routes. The considered insertion operators are described below.

- Basic greedy
- Random greedy
- Different caregiver basic greedy
- Different caregiver random greedy

3.4.1 Basic greedy

The basic greedy operator, described in Algorithm 3.8, adds to the schedule the services that, when they are inserted at their best position, it results in the least objective function increase.

Algorithm 3.8: Basic greedy operator

```
   Data: Solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$)
1  $S_u \leftarrow$ unscheduledServices($\boldsymbol{\omega}$)
2  while $|S_u| > 0$ do
       //Schedule every service at every position of the route
3      $\Omega \leftarrow \emptyset$ //Set of solutions
4      for $l \in S_u$ do
5          $d \leftarrow$ day($l$) //Day of service $l$
6          for $i \in N$ do
7              $R_i \leftarrow$ route($i, d, \boldsymbol{\omega}$) //Route of caregiver $i$
               //Schedule the service at every position of the route
8              for $j \in R_i = \{0, 1, ..., r, s\}$ do
9                  if $j = 0$ then
10                     $\bar{R} \leftarrow \{l, 1, ..., r\}$
11                 else if $j = r$ then
12                     $\bar{R} \leftarrow \{1, ..., r, l\}$
13                 else if $j \neq s$ then
14                     $\bar{R} \leftarrow \{1, ..., j, l, j+1, ..., r\}$
                   //If service $l$ can be added to $R$
15                 if feasible($\bar{R}, i, d, l$) then
16                     $\bar{\boldsymbol{t}} \leftarrow \boldsymbol{t}$, $\bar{t}_{j' \in \bar{R}} \leftarrow$ schedule($\bar{R}, f$) //Get the new schedule of $\bar{R}$
17                     $\bar{\boldsymbol{x}} \leftarrow \boldsymbol{x}$, $\bar{x}^{id}_{jl} \leftarrow 1$, $\bar{x}^{id}_{lj+1} \leftarrow 1$ //Update the route
18                     $\Omega \leftarrow \Omega \cup \{(\bar{\boldsymbol{x}}, \bar{\boldsymbol{t}})\}$ //Add the new solution
19     $\boldsymbol{\omega} \leftarrow \{\tilde{\boldsymbol{\omega}} / \boldsymbol{f}(\tilde{\boldsymbol{\omega}}) = \min_{\hat{\boldsymbol{\omega}} \in \Omega}\{\boldsymbol{f}(\hat{\boldsymbol{\omega}})\}\}$ //Choose the solution with best cost
       //Check if a service has been added to the schedule
20     $\hat{S}_u \leftarrow$ unscheduledServices($\boldsymbol{\omega}$) //Get the new non scheduled services
21     if $|S_u| = |\hat{S}_u|$ then
22         break loop //No service has been added to the schedule
23     else
24         $S_u \leftarrow \hat{S}_u$ //The service has been added to the schedule
25 return $\boldsymbol{\omega}$
```

The algorithm, while there are unscheduled services, iterates through them to select the next one that needs to be added to the solution. Each service is assigned to all available caregivers and scheduled in each feasible position of the routes (lines 4 - 18), which results in multiple new solutions. Then, from all the possible solutions, the one with minimal cost is chosen (line 19), which means that its corresponding new service is the one added to the schedule. Finally, the set of unscheduled services is updated (line 20). The process finishes when no service is added to the solution[3] (lines 21 - 22).

Example 3.4.1. Illustration of basic greedy operator.

Let us consider that three services must be scheduled in a single route. The services have a duration of 90 minutes[4] and the travel times between them are: $\theta_{1,2} = 10$, $\theta_{1,3} = 5$ and $\theta_{2,3} = 15$. The hard time windows of the services are: $\underline{\alpha}_1^d = 0$, $\bar{\alpha}_1^d = 450$, $\underline{\alpha}_2^d = 60$, $\bar{\alpha}_2^d = 510$, $\underline{\alpha}_3^d = 540$ and $\bar{\alpha}_3^d = 690$. For simplicity, the objective function considered will be the worked time of the caregiver.

The first service to be scheduled is chosen at random, following a uniform distribution, since all of them have equal duration. Let us assume that the one selected is Service 1 (see Figure 3.9).

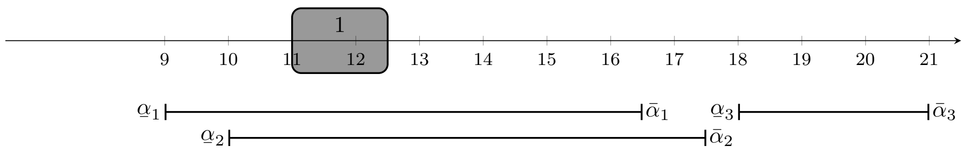


Figure 3.9: Route to be repaired using the basic greedy operator.

The next service to be added to the route is the one that will result in the least increase of cost. In order to do this, the services are inserted in every feasible position of the route and the best one will be selected. First, the different positions where Service 2 can be inserted in the route are evaluated:

- Scheduling Service 2 in the first position of the route (see Figure 3.10) results in a cost of $t_1 + \eta_1 - t_2 = 160 + 90 - 60 = 190$ minutes.

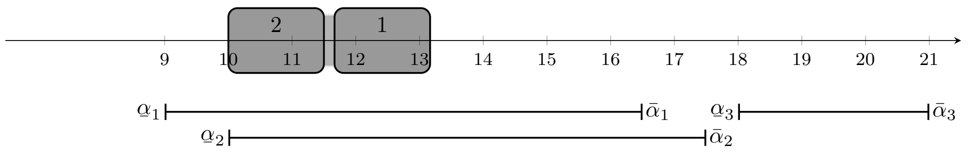


Figure 3.10: Schedule with Service 2 in first position (basic greedy).

- Scheduling Service 2 in the second position of the route (see Figure 3.11) results in a cost of $t_2 + \eta_2 - t_1 = 100 + 90 - 0 = 190$ minutes.

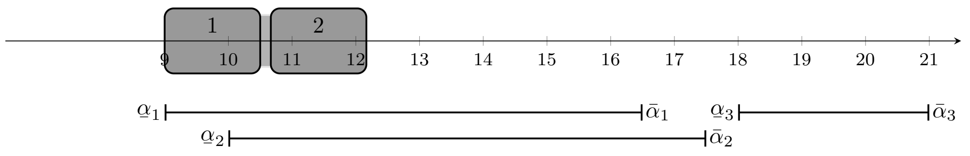


Figure 3.11: Schedule with Service 2 in second position (basic greedy).

[3]If no new service has been added to the solution, it means that it is not feasible to assign the unscheduled services to the available caregivers.

[4]For simplicity, the example is presented in minutes and therefore, the time 9:00 corresponds to the minute 0.

Now, the different positions where Service 3 can be introduced in the route are evaluated:

- Scheduling Service 3 in the first position of the route (see Figure 3.12) results in a non feasible solution, because the hard time window of Service 1 ends before the one of 3 begins.

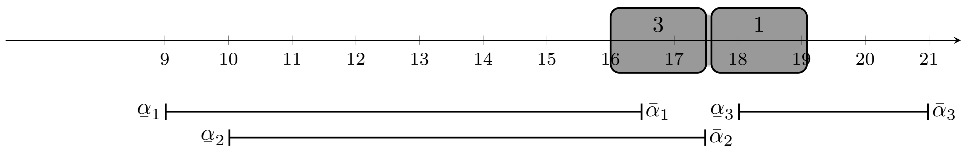


Figure 3.12: Schedule with Service 3 in first position (basic greedy).

- Scheduling Service 3 in the second position of the route (see Figure 3.13) results in a cost of $t_3 + \eta_3 - t_1 - (t_3 - (t_1 + \eta_1 + \theta_{1,3})) = 540 + 90 - 0 - (540 - (0 + 90 + 5)) = 185$ minutes. Since this is the option giving the least cost, Service 3 will be scheduled in the second position of the route.

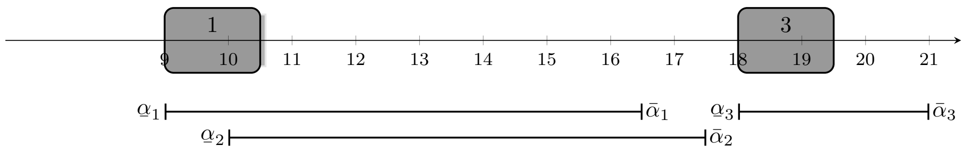


Figure 3.13: Schedule with Service 3 in second position(basic greedy).

Finally, the best position in the schedule for Service 2 is decided:

- Scheduling Service 2 in the first position of the route (see Figure 3.14) results in a cost of $t_3 + \eta_3 - t_2 - (t_3 - (t_1 + \eta_1 + \theta_{1,3})) = 540 + 90 - 60 - (540 - (160 + 90 + 5)) = 285$ minutes.

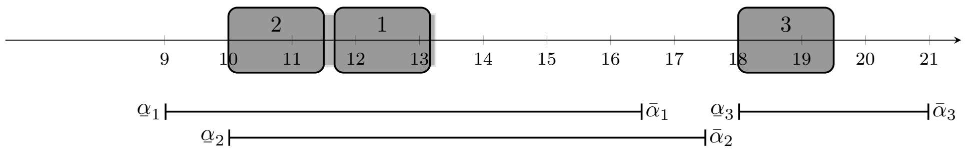


Figure 3.14: Schedule with Service 2 in first position (basic greedy).

- Scheduling Service 2 in the second position of the route (see Figure 3.15) results in a cost of $t_3 + \eta_3 - t_1 - (t_3 - (t_2 + \eta_2 + \theta_{2,3})) = 540 + 90 - 0 - (540 - (100 + 90 + 15)) = 295$ minutes.

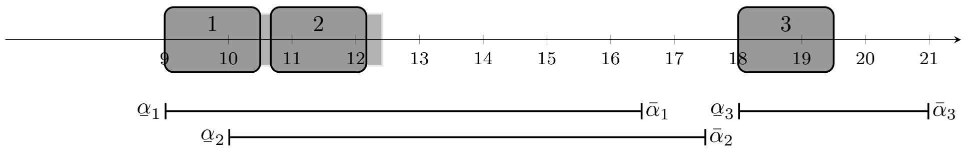


Figure 3.15: Schedule with Service 2 in second position (basic greedy).

- It is not feasible to schedule Service 2 in the last position of the route, because the hard time window of 2 ends before the one of 3 starts.

Thus, the best schedule is the one of Figure 3.14, because it is the one with smallest cost.

3.4.2 Random greedy

The random greedy operator, described in Algorithm 3.9, iteratively chooses a service at random and, then, schedules it at the best position according to the objective function value.

The algorithm, while the stopping criteria is not met, selects a service at random following a uniform distribution (line 3), assigns it to all available caregivers and schedules it in each feasible position of the routes (lines 4 - 17), which results in multiple new solutions. Then, from all the possible solutions, the one with the smallest cost is chosen (line 18). The algorithm stops when all services have been added to the solution or when a certain number of iterations without adding a new service to the solution is reached (lines 20 - 23).

Algorithm 3.9: Random greedy operator

```
Data: the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$) and the number of iterations without improvement ($m$)
1  $S_u \leftarrow$ unscheduledService($\boldsymbol{\omega}$)
2  while $|S_u| > 0$ and cont $< m$ do
3      $l \leftarrow$ randomService($S_u$), $d \leftarrow$ day($l$) //Get a service (and its day) at random
       //Schedule the service at every position of the route
4      $\Omega \leftarrow \emptyset$
5      for $i \in N$ do
6          $R_i \leftarrow$ route($i, d, \boldsymbol{\omega}$) //Route of caregiver $i$
           //Schedule the service at every position of the route
7          for $j \in R_i = \{0, 1, ..., r, s\}$ do
8              if $j = 0$ then
9                  $\bar{R} \leftarrow \{l, 1, ..., r\}$
10             else if $j = r$ then
11                 $\bar{R} \leftarrow \{1, ..., r, l\}$
12             else if $j \neq s$ then
13                 $\bar{R} \leftarrow \{1, ..., j, l, j+1, ..., r\}$
               //If service $l$ can be added to $R$
14             if feasible($\bar{R}, i, d, l$) then
15                 $\bar{\boldsymbol{t}} \leftarrow \boldsymbol{t}$, $\bar{t}_{j' \in \bar{R}} \leftarrow$ schedule($\bar{R}, f$) //Get the new schedule of $\bar{R}$
16                 $\bar{\boldsymbol{x}} \leftarrow \boldsymbol{x}$, $\bar{x}^{id}_{jl} \leftarrow 1$, $\bar{x}^{id}_{lj+1} \leftarrow 1$ //Update the route
17                 $\Omega \leftarrow \Omega \cup \{(\bar{\boldsymbol{x}}, \bar{\boldsymbol{t}})\}$ //Add the new solution
18     $\boldsymbol{\omega} \leftarrow \{\tilde{\boldsymbol{\omega}} / \boldsymbol{f}(\tilde{\boldsymbol{\omega}}) = \min_{\hat{\boldsymbol{\omega}} \in \Omega}\{\boldsymbol{f}(\hat{\boldsymbol{\omega}})\}\}$ //Choose the solution with best cost
       //Check if a service has been added to the schedule
19     $\hat{S}_u \leftarrow$ unscheduledServices($\boldsymbol{\omega}$)
20     if $|S_u| = |\hat{S}_u|$ then
21         cont $\leftarrow$ cont $+ 1$ //No service has been added to the schedule
22     else
23         $S_u \leftarrow \hat{S}_u$, cont $\leftarrow 0$
24 return $\boldsymbol{\omega}$
```

Example 3.4.2. Illustration of random greedy operator.

Let us consider the example in Figure 3.9. The first service to be scheduled is chosen at random, according to a uniform distribution, because all of them have the same duration. In this case, service 1 is selected (see Figure 3.16).

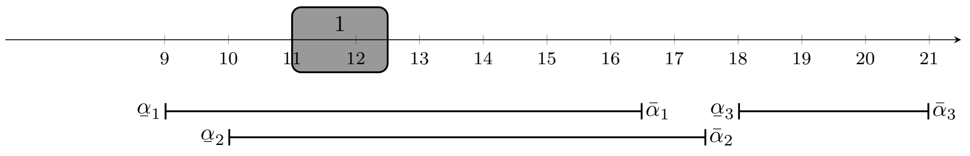


Figure 3.16: Route to be repaired using the random greedy operator.

The next service to be added to the route is chosen at random, Service 2, and it will be inserted at the position resulting in the least increase of cost.

- Scheduling Service 2 in the first position of the route (see Figure 3.17) results in a cost of $t_1 + \eta_1 - t_2 = 160 + 90 - 60 = 190$ minutes.

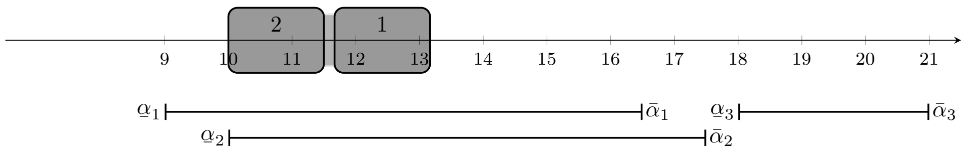


Figure 3.17: Schedule Service 2 in the first position (random greedy).

- Scheduling Service 2 in the second position of the route (see Figure 3.18) results in a cost of $t_2 + \eta_2 - t_1 = 100 + 90 - 0 = 190$ minutes.

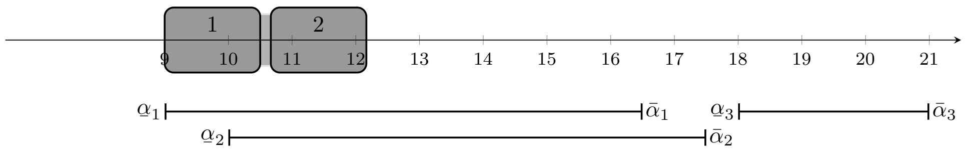


Figure 3.18: Schedule Service 2 in the second position (random greedy).

Both options result in the same cost, so the service is randomly scheduled in the second position. Finally, the best position to schedule Service 3 must be found.

- Scheduling Service 3 in the first or second position of the route results in a no feasible solution, because the hard time windows of services 1 and 2 end before the one of 3 begins. This means that the only feasible option is to schedule Service 3 in the last position of the route (see Figure 3.19), which results in a cost of $t_3 + \eta_3 - t_1 - (t_3 - (t_2 + \eta_2 + \theta_{2,3})) = 540 + 90 - 0 - (540 - (100 + 90 + 15)) = 295$ minutes.

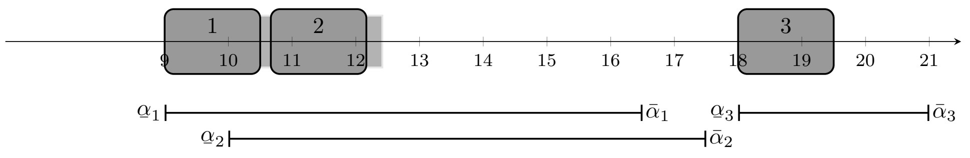


Figure 3.19: Schedule Service 3 in the third position (random greedy).

Algorithm 3.10: Different caregiver basic greedy operator

```
Data: the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$) and the previous solution ($\hat{\boldsymbol{\omega}}$)
1  $S_u \leftarrow \textbf{\textit{unscheduledServices}}(\boldsymbol{\omega})$
2  while $|S_u| > 0$ do
       //Schedule every service at every position of the route
3      $\Omega \leftarrow \emptyset$
4      for $l \in S_u$ do
5          $d \leftarrow \textbf{\textit{day}}(l)$, $i_l \leftarrow \textbf{\textit{caregiver}}(l, \hat{\boldsymbol{\omega}})$
6          for $i \in N,\ i \neq i_l$ do
7              $R_i \leftarrow \textbf{\textit{route}}(i, d, \boldsymbol{\omega})$
               //Schedule the service at every position of the route
8              for $j \in R_i = \{0, 1, ..., r, s\}$ do
9                  if $j = 0$ then
10                     $\bar{R} \leftarrow \{l, 1, ..., r\}$
11                 else if $j = r$ then
12                     $\bar{R} \leftarrow \{1, ..., r, l\}$
13                 else if $j \neq s$ then
14                     $\bar{R} \leftarrow \{1, ..., j, l, j+1, ..., r\}$
                   //Check if service $l$ can be added to $R$
15                 if $\textbf{\textit{feasible}}(\bar{R}, i, d, l)$ then
16                     $\bar{\boldsymbol{t}} \leftarrow \boldsymbol{t}$, $\bar{t}_{j' \in \bar{R}} \leftarrow \textbf{\textit{schedule}}(\bar{R}, f)$ //Get the new schedule of $\bar{R}$
17                     $\bar{\boldsymbol{x}} \leftarrow \boldsymbol{x}$, $\bar{x}^{id}_{jl} \leftarrow 1$, $\bar{x}^{id}_{lj+1} \leftarrow 1$ //Update the route
18                     $\Omega \leftarrow \Omega \cup \{(\bar{\boldsymbol{x}}, \bar{\boldsymbol{t}})\}$ //Add the new solution
19         if $\Omega = \emptyset$ then // If service $l$ was not assigned to any other caregiver
               //Assign the service to the caregiver attending it before
20             $i \leftarrow i_l$, $R_i \leftarrow \textbf{\textit{route}}(i, d, \boldsymbol{\omega})$
               //Schedule the service at every position of the route
21             for $j \in R_i = \{0, 1, ..., r, s\}$ do
22                 if $j = 0$ then
23                     $\bar{R} \leftarrow \{l, 1, ..., r\}$
24                 else if $j = r$ then
25                     $\bar{R} \leftarrow \{1, ..., r, l\}$
26                 else if $j \neq s$ then
27                     $\bar{R} \leftarrow \{1, ..., j, l, j+1, ..., r\}$
                   //Check if service $l$ can be added to $R$
28                 if $\textbf{\textit{feasible}}(\bar{R}, i, d, l)$ then
29                     $\bar{\boldsymbol{t}} \leftarrow \boldsymbol{t}$, $\bar{t}_{j' \in \bar{R}} \leftarrow \textbf{\textit{schedule}}(\bar{R}, f)$ //Get the new schedule of $\bar{R}$
30                     $\bar{\boldsymbol{x}} \leftarrow \boldsymbol{x}$, $\bar{x}^{id}_{jl} \leftarrow 1$, $\bar{x}^{id}_{lj+1} \leftarrow 1$ //Update the route
31                     $\Omega \leftarrow \Omega \cup \{(\bar{\boldsymbol{x}}, \bar{\boldsymbol{t}})\}$ //Add the new solution
32     $\boldsymbol{\omega} \leftarrow \{\tilde{\boldsymbol{\omega}} / \boldsymbol{f}(\tilde{\boldsymbol{\omega}}) = \min_{\hat{\boldsymbol{\omega}} \in \Omega}\{\boldsymbol{f}(\hat{\boldsymbol{\omega}})\}\}$ //Best solution
33     $\hat{S}_u \leftarrow \textbf{\textit{unscheduledServices}}(\boldsymbol{\omega})$
       //Check if a service has been added to the schedule
34     if $|S_u| = |\hat{S}_u|$ then
35         break //No service has been added to the schedule
36     else
37         $S_u \leftarrow \hat{S}_u$
38 return $\boldsymbol{\omega}$
```

Algorithm 3.11: Different caregiver random greedy operator

```
Data: the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$), the previous solution ($\hat{\boldsymbol{\omega}}$), iterations without improvement
      ($m$) and the objective function ($f$)
1  $S_u \leftarrow$ unscheduledServices($\boldsymbol{\omega}$)
2  while $|S_u| > 0$ and $cont < m$ do
3      $l \leftarrow$ randomServices($S_u$), $d \leftarrow$ day($l$), $i_l \leftarrow$ caregiver($l, \hat{\boldsymbol{\omega}}$), $\Omega \leftarrow \emptyset$
       // Assign the service to every caregiver
4      for $i \in N,\ i \neq i_l$ do
5          $R_i \leftarrow$ route($i, d, \boldsymbol{\omega}$)
           // Schedule the service at every position of the route
6          for $j \in R_i = \{0, 1, ..., r, s\}$ do
7              if $j = 0$ then
8                  $\bar{R} \leftarrow \{l, 1, ..., r\}$
9              else if $j = r$ then
10                 $\bar{R} \leftarrow \{1, ..., r, l\}$
11             else if $j \neq s$ then
12                 $\bar{R} \leftarrow \{1, ..., j, l, j+1, ..., r\}$
               // If service $l$ can be added to $R$
13             if feasible($\bar{R}, i, d, l$) then
14                 $\bar{\boldsymbol{t}} \leftarrow \boldsymbol{t}$, $\bar{t}_{j' \in \bar{R}} \leftarrow$ schedule($\bar{R}, f$) // Get the new schedule of $\bar{R}$
15                 $\bar{\boldsymbol{x}} \leftarrow \boldsymbol{x}$, $\bar{x}^{id}_{jl} \leftarrow 1$, $\bar{x}^{id}_{lj+1} \leftarrow 1$ // Update the route
16                 $\Omega \leftarrow \Omega \cup \{(\bar{\boldsymbol{x}}, \bar{\boldsymbol{t}})\}$ // Add the new solution
17     if $\Omega = \emptyset$ then // If service $l$ was not assigned to any other caregiver
           // Assign the service to the caregiver attending it before
18         $i \leftarrow i_l$, $R_i \leftarrow$ route($i, d, \boldsymbol{\omega}$)
           // Schedule the service at every position of the route
19         for $j \in R_i = \{0, 1, ..., r, s\}$ do
20             if $j = 0$ then
21                 $\bar{R} \leftarrow \{l, 1, ..., r\}$
22             else if $j = r$ then
23                 $\bar{R} \leftarrow \{1, ..., r, l\}$
24             else if $j \neq s$ then
25                 $\bar{R} \leftarrow \{1, ..., j, l, j+1, ..., r\}$
               // If service $l$ can be added to $R$
26             if feasible($\bar{R}, i, d, l$) then
27                 $\bar{\boldsymbol{t}} \leftarrow \boldsymbol{t}$, $\bar{t}_{j' \in \bar{R}} \leftarrow$ schedule($\bar{R}, f$) // Get the new schedule of $\bar{R}$
28                 $\bar{\boldsymbol{x}} \leftarrow \boldsymbol{x}$, $\bar{x}^{id}_{jl} \leftarrow 1$, $\bar{x}^{id}_{lj+1} \leftarrow 1$ // Update the route
29                 $\Omega \leftarrow \Omega \cup \{(\bar{\boldsymbol{x}}, \bar{\boldsymbol{t}})\}$ // Add the new solution
30     $\boldsymbol{\omega} \leftarrow \{\tilde{\boldsymbol{\omega}} / \boldsymbol{f}(\tilde{\boldsymbol{\omega}}) = \min_{\hat{\boldsymbol{\omega}} \in \Omega}\{\boldsymbol{f}(\hat{\boldsymbol{\omega}})\}\}$ // Best solution
31     $\hat{S}_u \leftarrow$ unscheduledServices($\boldsymbol{\omega}$)
       // Check if a service has been added to the schedule
32     if $|S_u| = |\hat{S}_u|$ then
33         $cont \leftarrow cont + 1$ // No service has been added to the schedule
34     else
35         $S_u \leftarrow \hat{S}_u$, $cont \leftarrow 0$
36 return $\boldsymbol{\omega}$
```

3.4.3 Different caregiver basic greedy

The different caregiver basic greedy operator, described in Algorithm 3.10, uses the basic greedy to add the services but, in this case, trying to guarantee that the services will be assigned to a different caregiver.

The algorithm tries to assign the services to a different caregiver than the one who was attending them before the destruction phase (lines 6 - 18). If it is not possible, the service is assigned to the caregiver attending it before (lines 19 - 31). Then, from all the possible solutions, the one with minimal cost is chosen (line 32), which means that its corresponding new service is the one added to the schedule. Finally, the set of unscheduled services is updated (line 33) and, if no service has been added to the solution, the loop terminates (lines 34 - 35).

3.4.4 Different caregiver random greedy

The different caregiver random greedy operator, described in Algorithm 3.11, is similar to the random greedy but trying to assign the services to different caregivers than the ones attending them before the destruction phase.

The algorithm, while the stopping criteria is not met, selects a service at random following a uniform distribution (line 3). Then, it tries to assign the services to a different caregiver from the one who was attending them before the destruction phase. Then, it is scheduled in each feasible position of the routes (lines 4 - 16). In case that is not possible, the service is assigned to the caregiver attending it before (lines 17 - 29). After that, from all the possible solutions, the one with minimal cost is chosen (line 30). The algorithm stops when all services have been added to the solution or when a certain number of iterations without adding a new service to the solution is reached (lines 32 - 35).

3.5 Obtaining the schedule of a route

Since a routing and scheduling problem is being considered, to evaluate the insertion operators it is not only necessary to know the routes of each caregiver, but also the starting times of every service. Because of this, after obtaining a new route using any of the insertion operators, it is necessary to define its schedule. This is done inside the insertion operators, using the function ***schedule***(R, f), where R is the route and f is the objective function.

One of the most important aspects when obtaining the schedule of a route is the objective function to optimize. Note that, once the route is fixed, there is no need to pay attention to the affinity and the overtime. The affinity cannot be changed because the services are already assigned to the caregiver. The overtime involves the weekly working time of the caregivers, which means that the best way to minimize the overtime is by minimizing the working time of the route. On one hand, prioritizing the welfare over the cost implies that the schedule of the route should minimize first the soft time window penalization and, after that, the working time of the caregiver. On the other hand, prioritizing the cost over the welfare requires that the schedule minimizes the working time and, then, the soft time window penalization.

Three different methods to obtain the schedule of a route have been developed:

First heuristic approach. A heuristic method that, given a route, obtains the schedule prioritizing the welfare over the cost. It is described in Chapter 4.

Second heuristic approach. A heuristic method that obtains the schedule of a route so it prioritizes the cost over the welfare. It is described in Chapter 5.

Constraint programming. The optimal starting times of a route can be obtained by modeling the resulting scheduling problem using constraint programming and optimizing the required objective function. This method, described in Section 3.5.1, will be used to evaluate the other two heuristic approaches.

3.5.1 Constraint programming

Constraint Programming (CP), originated in the field of artificial intelligence and computer science, consists in formulating and solving problems in terms of the set of constraints that define the problem, rather than focusing on the variables or the objective function (for more information see Rossi *et al.* (2006)). These type of problems, defined by a set of constraints, are called Constraint Satisfaction Problems (CSP) and the goal can be to obtain one, some, or even all feasible solutions. In addition to this, there are areas of CP where an objective function has to be optimized (mainly planning, sequencing and scheduling). Associating a CSP with an objective function results in a Constraint Optimization Problem (COP). When solving a COP the goal is to find the best solution with respect to the considered objective. Once a feasible solution is found, this can be done by adding a new constraint requiring that for subsequent solutions the objective value should be better. Ideally, when solving a COP, there should be guarantees that the solution found is the optimal one, which could be done using linearization techniques. A short review on mathematical programming techniques in CP can be found in Focacci *et al.* (2002).

The advantage of CP, that makes it efficient when solving scheduling problems, is that it has a rich set of operators and variable types, allowing a more intuitive way of programming. In terms of variables, CP allows variable indexing, which means that a variable can be used as an index for other one, which reduces the number of variables involved in the problem. In terms of constraints, CP uses special constraints such as inequality constraints (e.g. $x \neq y$), logical constraints (e.g. $x > y$ or $y > x$), maximum/minimum constraints (e.g. $x = \max\{y, z, t\}$), implication constraints (e.g. if $x > y$ then $x > t$) or global constraints (e.g. $allDifferent(x)$, that is, a constraint guaranteeing that all the values of variable x are different), among others.

CP has been used to solve multiple kinds of scheduling problems: staff scheduling in health care (see Weil *et al.* (1995) , Bourdais *et al.* (2003) and Trilling *et al.* (2006)), precast production scheduling (see Chan & Hu (2002)), home care scheduling (see Bertels & Fahle (2006)) or drone scheduling (see Ham (2018) and Montemanni & Dell'Amico (2023)), among others. A review on different techniques used in the literature to solve scheduling problems can be found in Fazel Zarandi *et al.* (2020)

3.5.1.1 CP model

In the problem under study, the goal is to use CP to obtain the schedule of a given route $R = \{1, ..., l\}$, in such a way that it optimizes a certain objective function[5]. The variables involved in the CP problem are:

- t_j and e_j, which specify the starting and ending time of service $j \in R$. Variable t_j is bounded by the hard time window of service j. Meanwhile, variable e_j is bounded by $\underline{\alpha}_j$ and $\bar{\alpha}+\theta_{j,j+1}$, that is, the travel time is considered as part of the duration of the service[6].

- $interv_j$, which represents the interval variable associated to service $j \in R$ and it must meet the following conditions: starts at variable t_j, ends at variable e_j and has a duration of $\eta_j + \theta_{j,j+1}$, where $j+1$ is the next service of the route. Therefore, the variable will ensure that $t_j + \eta_j + \theta_{j,j+1} = e_j$ (there is no need to add this as a constraint).

[5]Note that, since a fixed route is being scheduled, the values of i (caregiver) and d (day) are omitted.

[6]Notice that, in case $j = l$, there is not service $j+1$. So $\theta_{j,j+1} = 0$.

- r, which is is the variable related to the maximum break between the services of the route.
- $\hat{r}$, which coincides with r if $r \geq \pi_{min}$. Otherwise, it takes value 0. Note that it represents the break that will not be considered as working time.
- v_j^{start} and v_j^{end}, which indicate the penalization of carrying out service j before, or after, its soft time window.

The objective function of the problem is:

$$\min \omega_2 \sum_{j \in R} (v_j^{start} + v_j^{end}) + \omega_4 (e_l - t_1 - \hat{r}) \tag{3.2}$$

where the first element of the objective is the soft time window penalization and the second is the working time associated to the schedule. The weights ω_2 and ω_4 must be set properly to establish the element of the objective function that will be prioritized when solving the problem.

The following constraints are necessary in order to make sure that the schedule of the route will follow the guidelines of the original problem.

Constraint (3.3) ensures that the order of the services of the route will be maintained in the schedule.

$$e_j \leq t_{j+1}, \forall j \in \{1, ..., l-1\} \tag{3.3}$$

Constraint (3.4) guarantees that there will be no overlap between the services (and their travel time).

$$\text{NoOverlap}(interv_j, \forall j \in R) \tag{3.4}$$

Constraint (3.5) states that each service must start, and end, within its hard time window.

$$\underline{\alpha}_j \leq t_j \leq \bar{\alpha} - \eta_j, \forall j \in R \tag{3.5}$$

Constraint (3.6) guarantees that the first service of the route will start after the beginning of the hard time window.

$$\underline{\gamma} \leq t_1 \tag{3.6}$$

Constraint (3.7) ensures that the last service of the route ends before the end of the hard time window of the caregiver.

$$e_l \leq \bar{\gamma} \tag{3.7}$$

The largest break between consecutive services of the route is obtained using Constraint (3.8). Note that the travel time between services is not considered as part of the break.

$$r = \max_{j=1,...,l-1} \{t_{j+1} - e_j\} \tag{3.8}$$

Constraint (3.9) is used to obtain the largest break to be discounted from the caregivers working time (if it is greater than or equal to π_{min}).

$$\text{If } r \geq \pi_{min} \text{ then } \hat{r} = r \text{ else } \hat{r} = 0 \tag{3.9}$$

Constraint (3.10) guarantees that the schedule does not surpass the maximum allowed daily working time of the caregiver.

$$e_l - t_1 - \hat{r} \leq \nu \tag{3.10}$$

The penalization for starting the services before their soft time windows is obtained in Constraint (3.11).

$$v_j^{start} = \max\{0, \underline{\beta}_j - t_j\}, \forall j \in R \tag{3.11}$$

Finally, Constraint (3.12) computes the penalization for ending the services after their soft time windows.

$$v_j^{end} = \max\{0, t_j + \eta_j - \bar{\beta}_j\}, \forall j \in R \tag{3.12}$$

3.5.1.2 Resolution

The scheduling problem under study has an objective function to optimize, but CP was developed to mainly address feasibility problems. This is overcome by using Google OR Tools (see Perron & Furnon (2022)), in particular its CP solver known as CPSAT, which uses SAT (satisfiability) methods to solve CP problems. CPSAT has been extended to optimize objective functions, so it can be used to find optimal solutions or, in case the optimality is not guaranteed, to know the gap with the objective bound. To solve a problem, CPSAT uses different techniques like the simplex algorithm, branch and bound or linear relaxations on a SAT solver (which aims to solve the SAT version of the problem).

CPSAT has been successfully used in the literature to solve a variety of problems: sports scheduling (see Dimitsas *et al.* (2022)), resource-constrained project scheduling (see Teichteil-Königsbuch *et al.* (2023)), cyclic hoist scheduling (see Efthymiou & Yorke-Smith (2023)), scheduling automated guided vehicles in production (see Schweitzer *et al.* (2023)), electric bus recharging scheduling (see Dhingra *et al.* (2021)) or drone delivery scheduling (see Montemanni & Dell'Amico (2023))

Therefore, a version of the ALNS method that obtains the schedules of the routes with CPSAT will be tested, denoted as ALNS_CPSAT.

Appendix 3.A Auxiliary functions

Some auxiliary functions are needed to correctly use the insertion operators during the reconstruction phase of the ALNS.

3.A.1 Check if a route is feasible

In order to add a new service to a given route, it is necessary to check if the resulting route is feasible,as Algorithm 3.12 shows. The method consists in checking whether the hard time windows of caregivers and services allow the new service to be added to the route.

Algorithm 3.12: feasible - Check if a route is feasible

```
Data: the route ($R = \{1, ..., r\}$), the caregiver ($i$), the day ($d$) and the service ($j$)
$t^e, t^l \leftarrow$ getEarliestLatest($R \backslash \{j\}, i, d$) // Earliest and latest staring times of $R \backslash \{j\}$
if $j = 1$ then
    $a \leftarrow \max\{\underline{\alpha}_j^d, \underline{\gamma}^{id}\}$ // Earliest starting time of $j$
    $b \leftarrow \min\{t_{j+1}^l - \theta_{j,j+1}, \bar{\alpha}_j^d\}$ // Latest ending time of $j$
    if $b - a \geq \eta_j$ then
        return $True$
else if $j = r$ then
    $a \leftarrow \max\{t_{j-1}^e + \eta_{j-1} + \theta_{j-1,j}, \underline{\alpha}_j^d\}$ // Earliest starting time of $j$
    $b \leftarrow \min\{\bar{\alpha}_j^d, \bar{\gamma}^{id}\}$ // Latest ending time of $j$
    if $b - a \geq \eta_j$ then
        return $True$
else
    $a \leftarrow \max\{t_{j-1}^e + \eta_{j-1} + \theta_{j-1,j}, \underline{\alpha}_j^d\}$ // Earliest starting time of $j$
    $b \leftarrow \min\{t_{j+1}^l - \theta_{j,j+1}, \bar{\alpha}_j^d\}$ // Latest ending time of $j$
    if $b - a \geq \eta_j$ then
        return $True$
```

3.A.2 Earliest and latest starting times

The earliest and latest starting times of the services, according to hard time windows, are obtained using Algorithm 3.13.

Algorithm 3.13: getEarliestLatest - Get earliest and latest staring times

```
Data: the route ($R$), the caregiver ($i$), the day ($d$)
//Get earliest start for the services, according to hard time windows
for $j \in R$ do
    if $j = 1$ then
        $t_j^e \leftarrow \max\{\underline{\alpha}_j^d, \underline{\gamma}^{id}\}$
    else
        $t_j^e \leftarrow \max\{\underline{\alpha}_j^d, t_{j-1}^e + \eta_{j-1} + \theta_{j-1,j}\}$
//Get latest start for the service, according to hard time windows
for $j \in reversed(R)$ do
    if $j = r$ then
        $t_j^l \leftarrow \min\{\bar{\alpha}_j^d - \eta_j, \bar{\gamma}^{id} - \eta_j\}$
    else
        $t_j^l \leftarrow \min\{\bar{\alpha}_j^d - \eta_j, t_{j+1}^l - \theta_{j,j+1} - \eta_j\}$
return $t^e, t^l$
```

Chapter 4

Hierarchical approach: welfare over cost

As it was explained in Chapter 3, when using the ALNS methodology to solve the problem, it is necessary to establish the schedule of the routes. In Section 3.5.1 an exact method for obtaining the schedules is proposed. However, in this chapter a heuristic algorithm is introduced. In particular, this new algorithm will be used to obtain the schedule of a route, in such a way that the welfare of the users will be prioritized over the cost of the schedule (which is the first approach considered to tackle the HCSP, as explained in Section 2.3). The combination of the ALNS with the scheduling algorithm described in this chapter is denoted by ALNS_WC.

First, the most important features of the algorithm are described in Section 4.1, explaining the general behavior of the method. After that, some auxiliary functions are presented in Appendix 4.A, which may be necessary for a deeper understanding of the algorithm.

4.1 Algorithm to schedule a route prioritizing welfare over cost

Figure 4.1 shows a simple diagram to explain the general scheme of the algorithm developed to obtain the schedule of a route, in such a way that the welfare is prioritized over the cost.

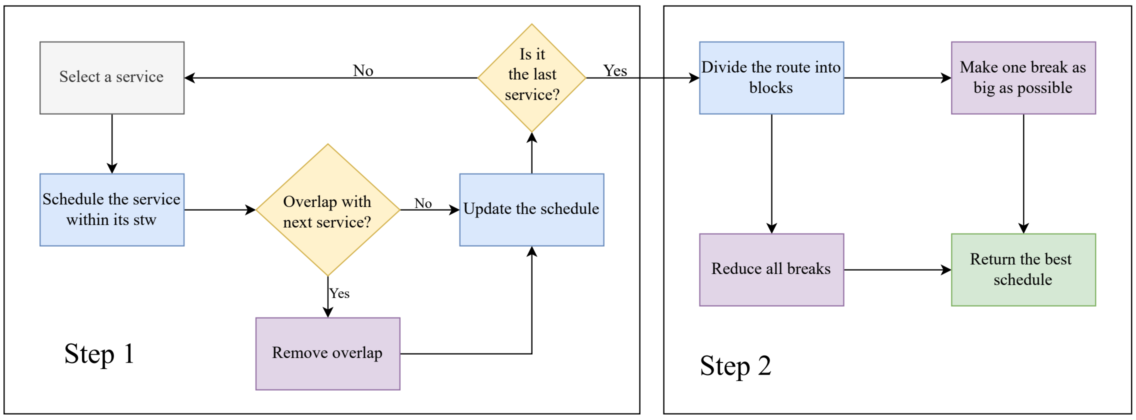


Figure 4.1: Scheme of algorithm ALNS_WC.

The algorithm is divided into two steps. In the first one, the schedule with best penalization value is found. This is done by trying to schedule the services within their soft time window and, if there are overlaps between them, removing them by working with the blocks of conflicting services. The second step modifies the schedule in order to improve the cost. To do this the route is divided into blocks of consecutive services and, after that, the cost is improved by reducing all breaks between the blocks or by making one of them as big as possible.

Algorithm 4.1 describes the method developed to obtain the schedule for a route ($R = \{1, ..., r\}$), that is, the schedule for each caregiver at each day[1]. The resulting schedule will have the best value for the penalization of preferred time windows. Once this is achieved, the algorithm focuses on minimizing the cost. To do this, it is necessary to obtain the earliest starting times of the services, as well as the blocks of services overlapping the soft time windows (line 1). Then, a solution with the optimal value for the penalization of soft time windows is obtained (line 2). Finally, this schedule is modified in order to reduce its cost (line 3).

Algorithm 4.1: ALNS_WC - Schedule a route prioritizing welfare over cost

```
  Data: the route ($R$)
  //Get earliest and latest times for the services and the blocks
1 $t^e, t^l, b^e, b^l, \Delta \leftarrow$ getInfo($R$)
  //Get the schedule with best penalization value
2 $t \leftarrow$ getSchedulePenalization($t^e, t^l, b^e, b^l, \Delta, R$)
  //Get the schedule with best penalization value
3 $\bar{t} \leftarrow$ getScheduleCost($t^e, t^l, t, \Delta, R$)
4 return $\bar{t}$
```

Now the three key functions outlined in the algorithm (***getInfo***, ***getSchedulePenalization*** and ***getScheduleCost***) will be carefully explained.

4.1.1 Obtain information of the route: *getInfo*

The function shown in Algorithm 4.2 obtains, for the services, two types of earliest and latest starting times: according to hard or soft time windows. After that, the route is divided into blocks, using function ***getBlocksSTW***, where two services (and the ones between them) belong to the same block if there are incompatibilities between their soft time windows.

[1]For simplicity, from now on indices i and d will be omitted.

Algorithm 4.2: getInfo - Get earliest times and blocks of services

```
   Data: the route ($R$)
   //Get earliest start for the services, according to hard time windows
 1 for $j \in R$ do
 2     if $j = 1$ then
 3         $t^e_j \leftarrow \max\{\underline{\alpha}_j, \underline{\gamma}\}$
 4     else
 5         $t^e_j \leftarrow \max\{\underline{\alpha}_j, t^e_{j-1} + \eta_{j-1} + \theta_{j-1,j}\}$
   //Get latest start for the services, according to hard time windows
 6 for $j \in reversed(R)$ do
 7     if $j = r$ then
 8         $t^l_j \leftarrow \min\{\bar{\alpha}_j - \eta_j, \bar{\gamma} - \eta_j\}$
 9     else
10         $t^l_j \leftarrow \min\{\bar{\alpha}_j - \eta_j, t^l_{j+1} - \theta_{j,j+1} - \eta_j\}$
   //Get earliest start for the services, according to soft time windows
11 for $j \in R$ do
12     $b^e_j \leftarrow \min\{\max\{\underline{\beta}_j, t^e_j\}, t^l_j\}$
   //Get latest start for the services, according to soft time windows
13 for $j \in R$ do
14     $b^l_j \leftarrow \max\{\min\{\bar{\beta}_j - \eta_j, t^l_j\}, t^e_j\}$
   //Get blocks of services with overlapping soft time windows
15 $\Delta \leftarrow$ getBlocksSTW$(R, t^e)$
16 return $t^e, t^l, b^e, b^l, \Delta$
```

Example 4.1.1. Illustration of Algorithm 4.2.

The algorithm is illustrated considering a route composed of 6 services, whose time windows are shown in Figure 4.2 and detailed in Table 4.1. For simplicity, all the services will have a duration of 1 hour and the travel time between them will be 5 minutes. Further, 6:00 is time 0 of the planning horizon. The available working times for the caregiver are $\underline{\gamma} = 0$ and $\bar{\gamma} = 960$.

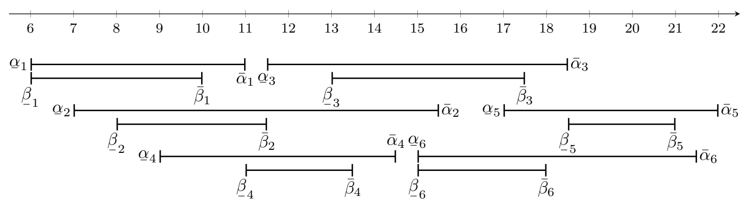


Figure 4.2: Hard and soft time windows of the services (ALNS_WC).

j	1	2	3	4	5	6
$\underline{\alpha}_j$	0	60	330	180	660	540
$\bar{\alpha}_j$	300	570	750	510	960	930
$\underline{\beta}_j$	0	120	420	300	750	540
$\bar{\beta}_j$	240	330	690	450	900	720

Table 4.1: Hard and soft time windows of the services (ALNS_WC).

The earliest starting times, according to hard time windows, are:

$j = 1$. $t^e_1 = \max\{\underline{\alpha}_1, \underline{\gamma}\} = \max\{0, 0\} = 0$.

$j = 2.\ t^e_2 = \max\{\underline{\alpha}_2, t^e_1 + \eta_1 + \theta_{1,2}\} = \max\{60, 0 + 60 + 5\} = 65.$

$j = 3.\ t^e_3 = \max\{\underline{\alpha}_3, t^e_2 + \eta_2 + \theta_{2,3}\} = \max\{330, 65 + 60 + 5\} = 330.$

$j = 4.\ t^e_4 = \max\{\underline{\alpha}_4, t^e_3 + \eta_3 + \theta_{3,4}\} = \max\{180, 330 + 60 + 5\} = 395.$

$j = 5.\ t^e_5 = \max\{\underline{\alpha}_5, t^e_4 + \eta_4 + \theta_{4,5}\} = \max\{660, 395 + 60 + 5\} = 660.$

$j = 6.\ t^e_6 = \max\{\underline{\alpha}_6, t^e_5 + \eta_5 + \theta_{5,6}\} = \max\{540, 660 + 60 + 5\} = 725.$

The latest starting times, according to hard time windows, are:

$j = 6.\ t^l_6 = \min\{\bar{\alpha}_6 - \eta_6, \bar{\gamma} - \eta_6\} = \min\{930 - 60, 960 - 60\} = 870.$

$j = 5.\ t^l_5 = \min\{\bar{\alpha}_5 - \eta_5, t^l_6 - \theta_{5,6} - \eta_5\} = \min\{960 - 60, 870 - 5 - 60\} = 805.$

$j = 4.\ t^l_4 = \min\{\bar{\alpha}_4 - \eta_4, t^l_5 - \theta_{4,5} - \eta_4\} = \min\{510 - 60, 805 - 5 - 60\} = 450.$

$j = 3.\ t^l_3 = \min\{\bar{\alpha}_3 - \eta_3, t^l_4 - \theta_{3,4} - \eta_3\} = \min\{750 - 60, 450 - 5 - 60\} = 385.$

$j = 2.\ t^l_2 = \min\{\bar{\alpha}_2 - \eta_2, t^l_3 - \theta_{2,3} - \eta_2\} = \min\{570 - 60, 385 - 5 - 60\} = 320.$

$j = 1.\ t^l_1 = \min\{\bar{\alpha}_1 - \eta_1, t^l_2 - \theta_{1,2} - \eta_1\} = \min\{300 - 60, 320 - 5 - 60\} = 240.$

The earliest starting times, according to soft time windows, are:

$j = 1.\ b^e_1 = \min\{\max\{\underline{\beta}_1, t^e_1\}, t^l_1\} = \min\{\max\{0, 0\}, 240\} = 0.$

$j = 2.\ b^e_2 = \min\{\max\{\underline{\beta}_2, t^e_2\}, t^l_2\} = \min\{\max\{120, 65\}, 320\} = 120.$

$j = 3.\ b^e_3 = \min\{\max\{\underline{\beta}_3, t^e_3\}, t^l_3\} = \min\{\max\{420, 330\}, 385\} = 385.$

$j = 4.\ b^e_4 = \min\{\max\{\underline{\beta}_4, t^e_4\}, t^l_4\} = \min\{\max\{300, 395\}, 450\} = 395.$

$j = 5.\ b^e_5 = \min\{\max\{\underline{\beta}_5, t^e_5\}, t^l_5\} = \min\{\max\{750, 660\}, 805\} = 750.$

$j = 6.\ b^e_6 = \min\{\max\{\underline{\beta}_6, t^e_6\}, t^l_6\} = \min\{\max\{540, 725\}, 870\} = 725.$

The latest starting times, according to soft time windows, are:

$j = 1.\ b^l_1 = \max\{\min\{\bar{\beta}_1 - \eta_1, t^l_1\}, t^e_1\} = \max\{\min\{240 - 60, 240\}, 0\} = 180.$

$j = 2.\ b^l_2 = \max\{\min\{\bar{\beta}_2 - \eta_2, t^l_2\}, t^e_2\} = \max\{\min\{330 - 60, 320\}, 65\} = 270.$

$j = 3.\ b^l_3 = \max\{\min\{\bar{\beta}_3 - \eta_3, t^l_3\}, t^e_3\} = \max\{\min\{690 - 60, 385\}, 330\} = 385.$

$j = 4.\ b^l_4 = \max\{\min\{\bar{\beta}_4 - \eta_4, t^l_4\}, t^e_4\} = \max\{\min\{450 - 60, 450\}, 395\} = 395.$

$j = 5.\ b^l_5 = \max\{\min\{\bar{\beta}_5 - \eta_5, t^l_5\}, t^e_5\} = \max\{\min\{900 - 60, 805\}, 660\} = 805.$

$j = 6.\ b^l_6 = \max\{\min\{\bar{\beta}_6 - \eta_6, t^l_6\}, t^e_6\} = \max\{\min\{720 - 60, 870\}, 725\} = 725.$

The earliest and latest starting times of the services, according to hard and soft time windows, are presented in Figure 4.3. Notice that the hard and soft time windows of the services are presented in gray.

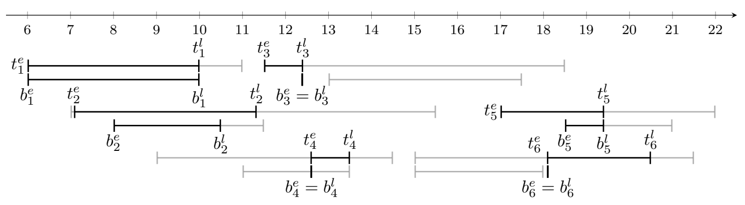


Figure 4.3: Earliest and latest starting times.

The function ***getBlocksSTW*** returns the list of blocks $\Delta = \{\{1,2\},\{3,4\},\{5,6\}\}$ (for more information see Example 4.A.1).

4.1.2 Obtain a schedule with best penalization value: *getSchedulePenalization*

Algorithm 4.3 describes the procedure used to obtain a schedule with a minimal value for the penalization of the soft time windows.

Algorithm 4.3: getSchedulePenalization - Get the schedule with best penalization value

```
Data: the earliest starting times htw ($t^e$), the latest starting times htw ($t^l$), the earliest
      starting times stw ($b^e$), the latest starting times stw ($b^l$), the schedule ($t$), the blocks
      ($\Delta$) and the route ($R$)
//Get the schedule with best penalization value
1  $\bar{R} \leftarrow \emptyset$
2  for $j \in R$ do
3      if $j \neq r$ and $j \notin \bar{R}$ then
           //Get overlap with follower
4          $\varpi \leftarrow b_j^e + \eta_j + \theta_{j,j+1} - b_{j+1}^l$
5          if $\varpi > 0$ then
               //Get block of services
6              $\bar{\delta} \leftarrow \emptyset$
7              for $\delta \in \Delta$ do
8                  if $j \in \delta$ or $j+1 \in \delta$ then
9                      $\bar{\delta} \leftarrow \bar{\delta} + \delta$
               //Schedule $\bar{\delta} = \{\delta_1, ..., \delta_r\}$ so the services are as close as possible
10             $t \leftarrow$ getScheduleBlock($t^e, t^l, t, \delta$)
               //Delay the block in order to reduce its penalization
11             $\bar{\delta}$, $t \leftarrow$ delayBlock($t^e, t^l, t, \delta, \Delta$)
12             $\bar{R} \leftarrow \bar{R} \cup \{\bar{\delta}\}$
13         else
               //Set time for $j$
14             $t_j \leftarrow b_j^e$, $\bar{R} \leftarrow \bar{R} \cup \{j\}$
               //Update earliest start, according to soft time window, of $j+1$
15             $b_{j+1}^e \leftarrow \max\{t_j + \eta_j + \theta_{j,j+1}, b_{j+1}^e\}$
16     else if $j = r$ and $j \notin \bar{R}$ then
17         $t_j \leftarrow b_j^e$
18 return $t = t_j \ \forall j \in R$
```

This function iterates through the services of the route and tries to schedule them within their soft time windows (lines 13 - 15). In case there are some services of a block that cannot be scheduled within their soft time window, all the services of the block will be taken into account when minimizing the penalization (lines 5 - 12).

Example 4.1.2. Illustration of Algorithm 4.3.

Continuing with the output obtained in Example 4.1.1, the goal now is to obtain the schedule of the route with the best soft time window penalization. First, the set of scheduled services is initialized, $\bar{R} = \emptyset$. So, iterating through the services the following schedule is found:

$j = 1$. The overlap that will be generated scheduling Service 1 at its earliest starting time, according to soft time windows, and its follower at the latest starting time is

$$\varpi = b_1^e + \eta_1 + \theta_{1,2} - b_2^l = 0 + 60 + 5 - 270 = -205.$$

This is illustrated in Figure 4.4. As $\varpi \leq 0$ (line 13), the schedule of the service is $t_1 = b_1^e = 0$ and the scheduled services are updated $\bar{R} = \bar{R} \cup \{1\} = \{1\}$ (line 14). Finally, the earliest starting time of 2 is updated, according to soft time windows, in order to not generate overlaps between these two services in future iterations, $b_2^e = \max\{t_1 + \eta_1 + \theta_{1,2}, b_2^e\} = \max\{0 + 60 + 5, 120\} = 120$ (line 15).

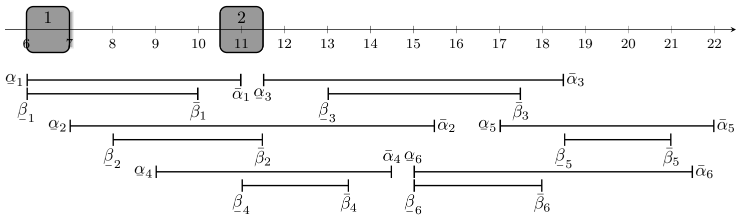


Figure 4.4: Schedule for services 1 and 2.

$j = 2$. The overlap that will be generated scheduling Service 2 at its earliest starting time, according to soft time windows, and its follower at the latest starting time is

$$\varpi = b_2^e + \eta_2 + \theta_{2,3} - b_3^l = 120 + 60 + 5 - 385 = -200.$$

This is illustrated in Figure 4.5. As $\varpi \leq 0$ (line 13), the schedule of the service is $t_2 = b_2^e = 120$ and the scheduled services are updated, $\bar{R} = \bar{R} \cup \{2\} = \{1, 2\}$ (line 14). Finally, the earliest starting time of 3 is updated, according to soft time windows, in order to not generate overlaps between these two services in future iterations, $b_3^e = \max\{t_2 + \eta_2 + \theta_{2,3}, b_3^e\} = \max\{120 + 60 + 5, 385\} = 385$ (line 15).

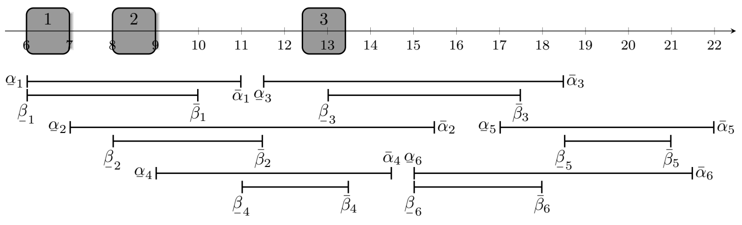


Figure 4.5: Schedule for services 1, 2 and 3.

$j = 3$. The overlap that will be generated if Service 3 is scheduled at its earliest starting time, according to soft time windows, and its follower at the latest starting time is

$$\varpi = b_3^e + \eta_3 + \theta_{3,4} - b_4^l = 385 + 60 + 5 - 395 = 55.$$

This is illustrated in Figure 4.6.

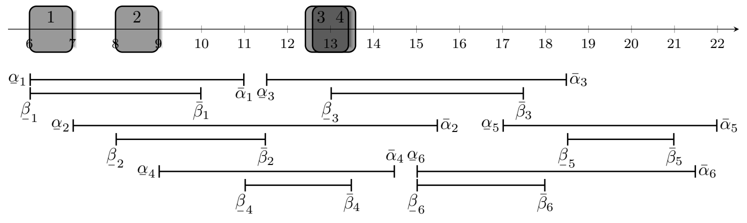


Figure 4.6: Schedule for services 1, 2, 3 and 4.

As $\varpi > 0$ (line 5), the block of Δ that contains the services 3 and 4 is found, which is $\delta = \{3, 4\}$. The services are scheduled with function ***getScheduleBlock*** (for more details see Example 4.A.3), $t_3 = 330$ and $t_4 = 395$. After that, function ***delayBlock*** (for more details see Example 4.A.5) improves the soft time window penalization of the block. The obtained elements are $\bar{\delta} = \{3, 4\}$ and the schedule (see Figure 4.7) is $t_1 = 0$, $t_2 = 120$, $t_3 = 330$ and $t_4 = 395$. Finally, the list of scheduled services is updated: $\bar{R} = \bar{R} \cup \bar{\delta} = \{1, 2\} \cup \{3, 4\} = \{1, 2, 3, 4\}$.

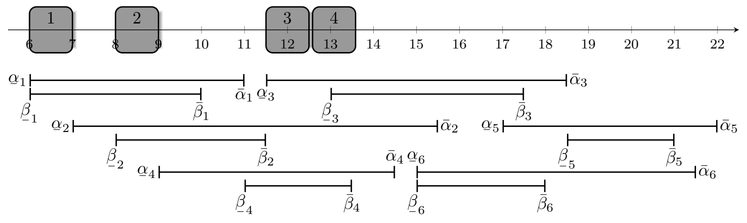


Figure 4.7: Schedule for services 1, 2, 3 and 4.

$j = 4$. This service is already scheduled since $4 \in \bar{R}$.

$j = 5$. The overlap that will be generated if Service 5 is scheduled at its earliest starting time, according to soft time windows, and its follower at the latest starting time is

$$\varpi = b_5^e + \eta_5 + \theta_{5,6} - b_6^l = 750 + 60 + 5 - 725 = 90.$$

This is illustrated in Figure 4.8.

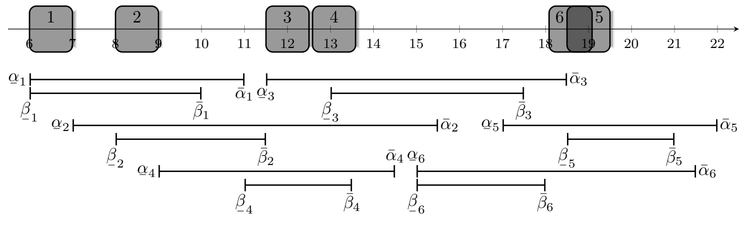


Figure 4.8: Schedule for services 1, 2, 3, 5 and 6.

As $\varpi > 0$ (line 5), the block of Δ that contains the services 5 and 6 is found, which is $\delta = \{5, 6\}$. The services are scheduled with function ***getScheduleBlock*** (see Example 4.A.4 for more details), obtaining $t_5 = 660$ and $t_6 = 725$. After that, function ***delayBlock*** (for more details see Example 4.A.6) improves the soft time window penalization of the block. The obtained elements are $\bar{\delta} = \{5, 6\}$ and the schedule, (see Figure 4.9), is $t_1 = 0$, $t_2 = 120$, $t_3 = 330$, $t_4 = 395$, $t_5 = 660$ and $t_6 = 725$. Finally, the list of scheduled services is updated: $\bar{R} = \bar{R} \cup \bar{\delta} = \{1, 2, 3, 4\} \cup \{5, 6\} = \{1, 2, 3, 4, 5, 6\}$.

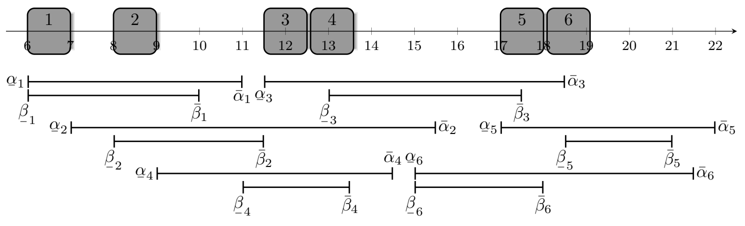


Figure 4.9: Schedule for services 1, 2, 3, 5 and 6.

$j = 6$. This service is already scheduled since $\{6\} \in \bar{R}$.

The function returns the schedule shown in Figure 4.10: $t_1 = 0$, $t_2 = 120$, $t_3 = 330$, $t_4 = 395$, $t_5 = 660$ and $t_6 = 725$. The soft time window penalization is

$$(\underline{\beta}_3 - t_3) + (\underline{\beta}_5 - t_5) + (t_4 + \eta_4 - \bar{\beta}_4) + (t_6 + \eta_6 - \bar{\beta}_5) =$$

$$= (420 - 330) + (750 - 660) + (395 + 60 - 450) + (725 + 60 - 720) = 90 + 90 + 5 + 65 = 250.$$

This is represented in Figure 4.10 as a dotted line next to the soft time window of the services with penalization.

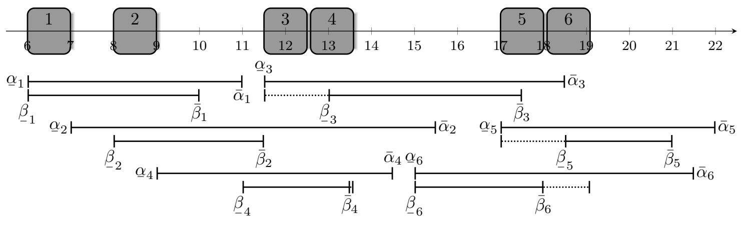


Figure 4.10: Schedule for the route.

4.1.3 Reduce the cost of the schedule: *getScheduleCost*

This section presents Algorithm 4.4, the procedure used to update the schedule in order to reduce its cost while maintaining the soft time window penalization.

This function consists in, first, dividing the route into blocks separated by the breaks. The schedule cost will be improved by making all breaks as small as possible (lines 6 - 8) or making one of them as big as possible (lines 10 - 15).

Algorithm 4.4: getScheduleCost - Get the schedule that optimizes the cost

```
Data: the earliest start ($t^e$), the latest start ($t^l$), the schedule ($t$), the blocks ($\Delta$) and the
      route ($R$)
//Get block of consecutive services
1  $\Delta \leftarrow$ getBlocksConsecutiveServices($R, t$)
//Get time window for each block so the penalization is maintained
2  $a^e, a^l \leftarrow$ getBlocksEarliestLatestStart($t^e, t^l, \Delta, t$)
//Improve the cost of the schedule
3  if there is only one block then
4      $\hat{t} \leftarrow t$
5  else
       //Make all breaks between blocks as small as possible
6      $\hat{t}_h \leftarrow a^l_h \ \forall h \in first\ block$
7      $j \leftarrow$ first service of second block
8      $\hat{t}_h \leftarrow \max\{a^e_h, \hat{t}_{h-1} + \eta_{h-1} + \theta_{h-1,h}\} \ \forall h \in [j, .., r]$
9      while $\delta \in \Delta$ is not the last one do
           //Make break after $\delta$ as big as possible
10         $\bar{t}_h \leftarrow a^e_h \ \forall h \in \delta$
11         $\bar{t}_h \leftarrow a^l_h \ \forall h \in \hat{\delta} = \delta + 1$
12         $j \leftarrow$ first service of $\delta$, $k \leftarrow$ last service of $\delta + 1 = \hat{\delta}$
13         $\bar{t}_h \leftarrow \min\{a^l_h, \bar{t}_{h+1} - \theta_{h,h+1} - \eta_h\} \ \forall h \in [j-1, ..., 1]$
14         $\bar{t}_h \leftarrow \max\{a^e_h, \bar{t}_{h-1} + \eta_{h-1} + \theta_{h-1,h}\} \ \forall h \in [k+1, ..., r]$
15         $\hat{t} \leftarrow$ chooseBestCostSchedules($\bar{t}$, $\hat{t}$) //Choose the schedule with best cost
16 return $\hat{t}$
```

Example 4.1.3. Illustration of Algorithm 4.4.

Let us start from the final schedule obtained in Example 4.1.2, depicted in Figure 4.10. First, function ***getBlocksConsecutiveServices*** divides the route into blocks separated by the breaks, $\Delta = \{\{1\}, \{2\}, \{3, 4\}, \{5, 6\}\}$ (for more details see Example 4.A.9). Then, function ***getBlocksEarliestLatestStart*** provides the earliest ($a^e_1 = 0$, $a^e_2 = 120$, $a^e_3 = 330$, $a^e_4 = 395$, $a^e_5 = 660$ and $a^e_6 = 725$) and latest ($a^l_1 = 180$, $a^l_2 = 270$, $a^l_3 = 385$, $a^l_4 = 450$, $a^l_5 = 750$ and $a^l_6 = 815$) starting times of the services that maintain the penalization of the blocks (for more details see Example 4.A.10).

To reduce the cost of the schedule it is necessary, on one hand, to make all the breaks between blocks as small as possible (lines 6 - 8):

$h = 1$. $\hat{t}_1 = a^l_1 = 180$.

$h = 2$. $\hat{t}_2 = \max\{a^e_2, \hat{t}_1 + \eta_1 + \theta_{1,2}\} = \max\{120, 180 + 60 + 5\} = 245$.

$h = 3$. $\hat{t}_3 = \max\{a^e_3, \hat{t}_2 + \eta_2 + \theta_{2,3}\} = \max\{330, 245 + 60 + 5\} = 330$.

$h = 4$. $\hat{t}_4 = \max\{a^e_4, \hat{t}_3 + \eta_3 + \theta_{3,4}\} = \max\{395, 330 + 60 + 5\} = 395$.

$h = 5$. $\hat{t}_5 = \max\{a^e_5, \hat{t}_4 + \eta_4 + \theta_{4,5}\} = \max\{660, 395 + 60 + 5\} = 660$.

$h = 6$. $\hat{t}_6 = \max\{a^e_6, \hat{t}_5 + \eta_5 + \theta_{5,6}\} = \max\{725, 660 + 60 + 5\} = 725$.

The largest break in this schedule (see Figure 4.11) is the one between services 4 and 5, which has a duration of $\hat{t}_5 - (\hat{t}_4 + \eta_4 + \theta_{4,5}) = 660 - (395 + 60 + 5) = 660 - 460 = 200$. The cost of the schedule is $\hat{t}_6 + \eta_6 - \hat{t}_1 - break = 725 + 60 - 180 - 200 = 405$.

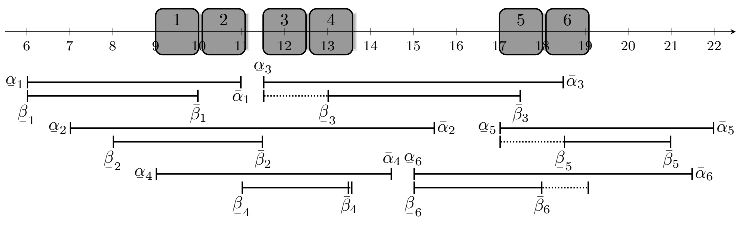


Figure 4.11: Schedule with the smallest breaks.

The other option is to make one of the breaks between the blocks as big as possible, while reducing the other ones (lines 9 - 15):

$\delta = \{1\}$. The schedule that makes the break after the block as big as possible is:

$h = 1$. $\bar{t}_1 = a_1^e = 0$.

$h = 2$. $\bar{t}_2 = a_2^l = 270$.

$h = 3$. $\bar{t}_3 = \max\{a_3^e, \bar{t}_2 + \eta_2 + \theta_{2,3}\} = \max\{330, 270 + 60 + 5\} = 335$.

$h = 4$. $\bar{t}_4 = \max\{a_4^e, \bar{t}_3 + \eta_3 + \theta_{3,4}\} = \max\{395, 335 + 60 + 5\} = 400$.

$h = 5$. $\bar{t}_5 = \max\{a_5^e, \bar{t}_4 + \eta_4 + \theta_{4,5}\} = \max\{660, 400 + 60 + 5\} = 660$.

$h = 6$. $\bar{t}_6 = \max\{a_6^e, \bar{t}_5 + \eta_5 + \theta_{5,6}\} = \max\{725, 660 + 60 + 5\} = 725$.

The largest break in this schedule (see Figure 4.12) is the one between services 1 and 2, which has a duration of $\bar{t}_2 - (\bar{t}_1 + \eta_1 + \theta_{1,2}) = 270 - (0 + 60 + 5) = 270 - 65 = 205$. The cost of the schedule is $\bar{t}_6 + \eta_6 - \bar{t}_1 - break = 725 + 60 - 0 - 205 = 580$.

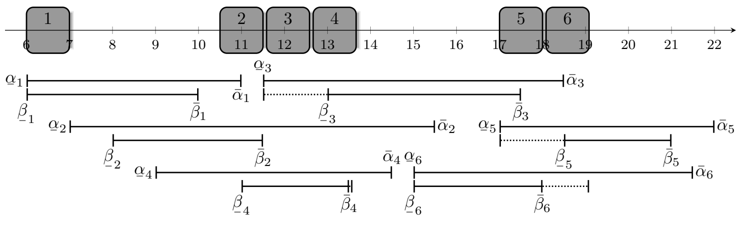


Figure 4.12: Schedule with the largest break between services 1 and 2.

$\delta = \{2\}$. The schedule that makes the break after the block as big as possible is:

$h = 2$. $\bar{t}_2 = a_2^e = 120$.

$h = 3$. $\bar{t}_3 = a_3^l = 385$.

$h = 4$. $\bar{t}_4 = a_4^l = 450$.

$h = 1$. $\bar{t}_1 = \min\{a_1^l, \bar{t}_2 - \theta_{1,2} - \eta_2\} = \min\{180, 120 - 60 - 5\} = 55$.

$h = 5$. $\bar{t}_5 = \max\{a_5^e, \bar{t}_4 + \eta_4 + \theta_{4,5}\} = \max\{660, 400 + 60 + 5\} = 660$.

$h = 6$. $\bar{t}_6 = \max\{a_6^e, \bar{t}_5 + \eta_5 + \theta_{5,6}\} = \max\{725, 660 + 60 + 5\} = 725$.

The largest break in this schedule (see Figure 4.13) is the one between services 2 and 3, which has a duration of $\bar{t}_3 - (\bar{t}_2 + \eta_2 + \theta_{2,3}) = 385 - (120 + 60 + 5) = 385 - 185 = 200$. The cost of the schedule is $\bar{t}_6 + \eta_6 - \bar{t}_1 - break = 725 + 60 - 55 - 200 = 530$.

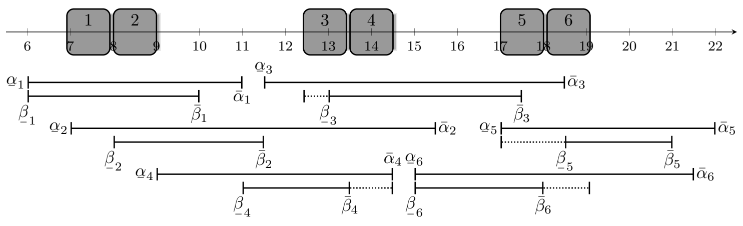


Figure 4.13: Schedule with the largest break between services 2 and 3.

$\delta = \{3, 4\}$. The schedule that makes the break after the block as big as possible is:

$h = 3$. $\bar{t}_3 = a^e_3 = 330$.

$h = 4$. $\bar{t}_4 = a^e_4 = 395$.

$h = 5$. $\bar{t}_5 = a^l_5 = 750$.

$h = 6$. $\bar{t}_6 = a^l_6 = 815$.

$h = 2$. $\bar{t}_2 = \min\{a^l_2, \bar{t}_3 - \theta_{2,3} - \eta_2\} = \min\{270, 330 - 60 - 5\} = 265$.

$h = 1$. $\bar{t}_1 = \min\{a^l_1, \bar{t}_2 - \theta_{1,2} - \eta_2\} = \min\{180, 265 - 60 - 5\} = 180$.

The largest break in this schedule (see Figure 4.14) is the one between services 4 and 5, which has a duration of $\bar{t}_5 - (\bar{t}_4 + \eta_4 + \theta_{4,5}) = 750 - (395 + 60 + 5) = 750 - 460 = 290$. The cost of the schedule is $\bar{t}_6 + \eta_6 - \bar{t}_1 - \mathit{break} = 815 + 60 - 180 - 290 = 405$.

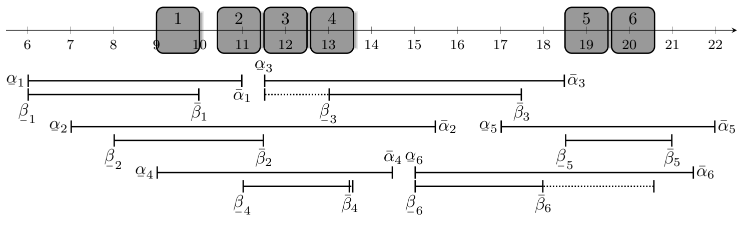


Figure 4.14: Schedule with the largest break between services 4 and 5.

The best cost value is 405 and there are two different schedules that reach this value. The first option is the schedule presented in Figure 4.11, which was obtained by making the breaks between the blocks as small as possible. The second option is the schedule of Figure 4.14, that was achieved by maximizing the duration of the break between blocks $\{3, 4\}$ and $\{5, 6\}$.

Appendix 4.A Auxiliary functions

The functions presented in this appendix are used to completely describe the method used to obtain the schedule of a route in order to prioritize its welfare over its cost.

4.A.1 Obtain blocks: *getBlocksSTW*

Algorithm 4.5 describes the method used to obtain the blocks of services that have overlapping soft time windows.

To reach this goal, the algorithm iterates through the services of the route and checks if their soft time window overlaps with the one of the other services, in which case the services are added to the same block (lines 2 - 14). Finally, the blocks are separated if the earliest starting times of the services are not consecutive (lines 15 - 23).

Algorithm 4.5: GetBlocksSTW - Divide the route into blocks

```
Data: the route ($R$), the earliest starting times ($t^e$)
//Get blocks of services that have overlapping soft time windows
1  $\Delta \leftarrow \emptyset$, $\hat{R} \leftarrow R$
2  for $j \in R$ do
3      if $j = r$ then
4          $\Delta \leftarrow \Delta \cup \{j\}$, $\hat{R} \leftarrow \hat{R} \backslash \{j\}$
5      else if $j \in \hat{R}$ then
           //Get services that have overlapping soft time window
6          $\delta \leftarrow \emptyset$
7          for $k \in F(j, \hat{R})$ do
8              if $k = j + 1$ and $\bar{\beta}_j + \theta_{j,k} > \underline{\beta}_k$ then
9                  $\delta \leftarrow \delta \cup \{k\}$
10             else if $\bar{\beta}_j \geq \underline{\beta}_k$ then
11                 $\delta \leftarrow \delta \cup \{k\}$
           //Add more services to the block
12         $\Delta$, $\hat{R} \leftarrow$ updateBlock($\Delta$, $\delta$, $R$, $\hat{R}$, $j$)
13         if $\hat{R} = \emptyset$ then
14             break
//Separate the block if the earliest times are not consecutive
15 $\hat{\Delta} \leftarrow \emptyset$
16 for $\delta = \{\delta_1, ..., \delta_r\} \in \Delta$ do
17     $j \leftarrow \delta_1$
18     for $k \in \delta$ do
19         if $k \neq \delta_r$ and $t^e_k + \eta_k + \theta_{k,k+1} < t^e_{k+1}$ then
               //Separate the block between services $k$ and $k + 1$
20             $\hat{\delta} \leftarrow \{j, ..., k\}$, $\hat{\Delta} \leftarrow \hat{\Delta} \cup \hat{\delta}$
21             $j \leftarrow k + 1$
22         else if $k = \delta_r$ then
               //End the block
23             $\hat{\delta} \leftarrow \{j, ..., k\}$, $\hat{\Delta} \leftarrow \hat{\Delta} \cup \hat{\delta}$
24 return $\hat{\Delta}$
```

Example 4.A.1. Illustration of Algorithm 4.5. Let us start from the route considered in Example 4.1.1. In order to divide this route into blocks, the algorithm starts initializing $\Delta = \emptyset$ and $\hat{R} = R = \{1, 2, 3, 4, 5, 6\}$. Then, for $j = 1 \in \hat{R}$, its followers ($\{2, 3, 4, 5, 6\}$) will be added to the block (initialized to $\delta = \emptyset$ (line 6)) if they have overlapping soft time windows with Service 1.

$k = 2$. In this case $\bar{\beta}_1 + \theta_{1,2} = 240 + 5 = 245$ and $\underline{\beta}_2 = 120$. Therefore, $\bar{\beta}_1 + \theta_{1,2} > \underline{\beta}_2$ (line 8) so $\delta = \delta \cup \{k\} = \{2\}$.

$k = 3$. In this case $\bar{\beta}_1 = 240$ and $\underline{\beta}_3 = 420$. Therefore, $\bar{\beta}_1 \ngeq \underline{\beta}_3$ (line 10), which means that 3 is not added to δ.

$k = 4$. In this case $\bar{\beta}_1 = 240$ and $\underline{\beta}_4 = 300$. Therefore, $\bar{\beta}_1 \ngeq \underline{\beta}_4$ (line 10), which means that 4 is not added to δ.

$k = 5$. In this case $\bar{\beta}_1 = 240$ and $\underline{\beta}_4 = 750$. Therefore, $\bar{\beta}_1 \ngeq \underline{\beta}_4$ (line 10), which means that 5 is not added to δ.

$k = 6$. In this case $\bar{\beta}_1 = 240$ and $\underline{\beta}_4 = 540$. Therefore, $\bar{\beta}_1 \ngeq \underline{\beta}_4$ (line 10), which means that 6 is not added to δ.

Function ***updateBlock*** results in $\Delta = \{\{1, 2, 3, 4, 5, 6\}\}$ and $\hat{R} = \emptyset$ (see more information in Example 4.A.2). Therefore, the loop terminates (lines 13 - 14).

Finally, the block is separated if the earliest starting times of the services are not consecutive. After initializing $\hat{\Delta} = \emptyset$, the first service of δ is selected $j = \delta_1 = 1$. Then, for each service in the block their earliest starting times are compared:

$k = 1$. In this case $t^e_1 + \eta_1 + \theta_{1,2} = 0 + 60 + 5 = 65$ and $t^e_2 = 65$. Therefore, $t^e_1 + \eta_1 + \theta_{1,2} \nless t^e_2$ (line 19).

$k = 2$. In this case $t^e_2 + \eta_2 + \theta_{2,3} = 65 + 60 + 5 = 130$ and $t^e_3 = 330$. Therefore, $t^e_2 + \eta_2 + \theta_{2,3} < t^e_3$ (line 19), which means that $\hat{\delta} = \{1, 2\}$, $\hat{\Delta} = \{1, 2\}$ and $j = k + 1 = 3$.

$k = 3$. In this case $t^e_3 + \eta_3 + \theta_{3,4} = 330 + 60 + 5 = 395$ and $t^e_4 = 395$. Therefore, $t^e_3 + \eta_3 + \theta_{3,4} \nless t^e_4$ (line 19).

$k = 4$. In this case $t^e_4 + \eta_4 + \theta_{4,5} = 395 + 60 + 5 = 460$ and $t^e_5 = 660$. Therefore, $t^e_4 + \eta_4 + \theta_{4,5} < t^e_5$ (line 19), which means that $\hat{\delta} = \{3, 4\}$, $\hat{\Delta} = \{\{1, 2\}, \{3, 4\}\}$ and $j = k + 1 = 5$.

$k = 5$. In this case $t^e_5 + \eta_5 + \theta_{5,6} = 660 + 60 + 5 = 725$ and $t^e_6 = 725$. Therefore, $t^e_5 + \eta_5 + \theta_{5,6} \nless t^e_6$ (line 19).

$k = 6$. In this case $k = 6 = \delta_r$ (line 22). Therefore, $\hat{\Delta} = \{5, 6\}$, $\hat{\Delta} = \{\{1, 2\}, \{3, 4\}, \{5, 6\}\}$.

This function returns the list of blocks $\hat{\Delta} = \{\{1, 2\}, \{3, 4\}, \{5, 6\}\}$. Figure 4.15 shows that scheduling the services at their earliest times results in the three blocks.

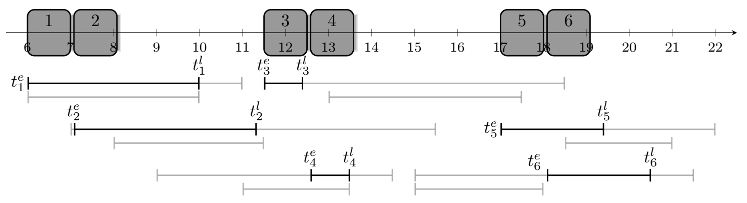


Figure 4.15: Blocks of services.

4.A.1.1 Update the block: *updateBlock*

Algorithm 4.6 is used to update the block adding those services that can overlap with the services belonging to the previous block.

In case there are services in the block (line 1), service j and all the services whose position in the route is between those already selected are added to the block (lines 2 - 3). Then, the algorithm checks if the services outside the block have overlapping soft time windows with any of the ones already in it[2] (lines 4 - 17).

[2]This is done using a while loop, in order to keep performing the method when a new service is added to the block, so that the services that overlap with it can be added to the block in future iterations.

Algorithm 4.6: updateBlock - Add more services to the block

```
Data: the blocks ($\Delta$), the block ($\delta$), the route ($R$), the free services ($\hat{R}$) and the service ($j$)
//Add more services to the block
1  if $\delta \neq \emptyset$ then
2      $\delta \leftarrow \{j\} \cup \delta$
       //Get services between those that overlap soft time windows
3      $i_1 \leftarrow \min_{k\in\delta}\{\textbf{index}(k,R)\}$, $i_2 \leftarrow \max_{k\in\delta}\{\textbf{index}(k,R)\}$, $\delta \leftarrow \{R[i_1],...,R[i_2]\}$
       //Remove from $\hat{R}$ the services of $\delta$ and its predecessors
4      $\hat{R} \leftarrow \hat{R}\backslash(\delta \cup P(\delta,\hat{R}))$
       //Get services that overlap with the ones in $\delta$
5      $keep \leftarrow True$, $\hat{\delta} \leftarrow \delta\backslash\{j,r\}$
6      while $keep = True$ do
7          $keep \leftarrow False$, $\bar{\delta} \leftarrow \emptyset$
8          for $k \in \hat{\delta}$ do
9              for $l \in \hat{R}$ do
10                 if $l = k+1$ then
11                     if $\bar{\beta}_k + \theta_{k,l} > \underline{\beta}_l$ then
12                         $\delta \leftarrow \delta \cup \{l\}$, $\bar{\delta} \leftarrow \bar{\delta} \cup \{l\}$, $\hat{R} \leftarrow \hat{R}\backslash\{l\}$, $keep \leftarrow True$
13                 else
14                     if $\bar{\beta}_k \geq \underline{\beta}_l$ then
15                         $\delta \leftarrow \delta \cup \{l\}$, $\bar{\delta} \leftarrow \bar{\delta} \cup \{l\}$, $\hat{R} \leftarrow \hat{R}\backslash\{l\}$, $keep \leftarrow True$
           //Get services between those that overlap soft time windows
16         $i_1 \leftarrow \min_{k\in\delta}\{\textbf{index}(k,R)\}$, $i_2 \leftarrow \max_{k\in\delta}\{\textbf{index}(k,R)\}$
17         $\hat{\delta} \leftarrow \bar{\delta} \cup (\{R[i_1],...,R[i_2]\}\backslash\delta)$, $\delta \leftarrow \{R[i_1],...,R[i_2]\}$, $\hat{R} \leftarrow \hat{R}\backslash\delta$
18     $\Delta \leftarrow \Delta \cup \delta$
19 else
20     $\Delta \leftarrow \Delta \cup \{j\}$, $\hat{R} \leftarrow \hat{R}\backslash\{j\}$
21 return $\Delta$, $\hat{R}$
```

Example 4.A.2. Illustration of Algorithm 4.6.

Continuing with Example 4.A.1, the given input data are: the list of blocks $\Delta = \emptyset$, the current block $\delta = \{2\}$, the Service $j = 1$, the route $R = \{1,2,3,4,5,6\}$, and the free services $\hat{R} = \{1,2,3,4,5,6\}$. To update the block, Service 1 is added to it, resulting in $\delta = \{1,2\}$ (line 2). Since there are no services in R between those of δ, no services are added to the block (line 3). Then, the services of the block and its predecessors are removed from the set of free services, $\hat{R} = \hat{R}\backslash(\delta \cup P(\delta,\hat{R})) = \{3,4,5,6\}$ (line 4). The parameter to enter the while loop is initialized, $keep = True$, as well as the set of not checked services, $\hat{\delta} = \delta\backslash\{j,r\} = \{2\}$ (line 5)[3].

After that, the while loop (line 6) sets $keep = False$ and $\bar{\delta} = \emptyset$. Then, iterating through the set of not checked services, $\hat{\delta}$, it is verified if there is overlap with any of the free services $\hat{R}$.

$k = 2$. Check if any free service has overlapping soft time window with 2:

$l = 3$. In this case $l = 3 = k+1$ (line 10). Therefore, $\bar{\beta}_2 + \theta_{2,3} = 330+5 = 335$ and $\underline{\beta}_3 = 420$. This means that $\bar{\beta}_2 + \theta_{2,3} \ngtr \underline{\beta}_3$ (line 11), so Service 3 is not added to δ.

$l = 4$. In this case $\bar{\beta}_2 = 330$ and $\underline{\beta}_4 = 330$. This means that $\bar{\beta}_2 \geq \underline{\beta}_4$ (line 14). Therefore, the service is added to the block, $\delta = \delta \cup \{4\} = \{1,2,4\}$, and to the list of checked

[3]Notice that Service $j = 1$ is not included in the set because the services that have overlap with it are already in the block (this was done in Example 4.A.1).

services, $\bar{\delta} = \bar{\delta} \cup \{4\} = \{4\}$. The service is removed from the list of free services, $\hat{R} = \hat{R} \backslash \{4\} = \{3, 5, 6\}$, setting $keep = True$.

$l = 5$. In this case $\bar{\beta}_2 = 330$ and $\underline{\beta}_5 = 750$. Since $\bar{\beta}_2 \not\geq \underline{\beta}_5$ (line 14), Service 5 is not added to δ.

$l = 6$. In this case $\bar{\beta}_2 = 330$ and $\underline{\beta}_6 = 390$. Therefore, $\bar{\beta}_2 \not\geq \underline{\beta}_6$ (line 14), so Service 6 is not added to δ.

Figure 4.16 shows that the only free service that has an overlapping soft time window with 2 (blue soft time window) is Service 4 (green soft time window).

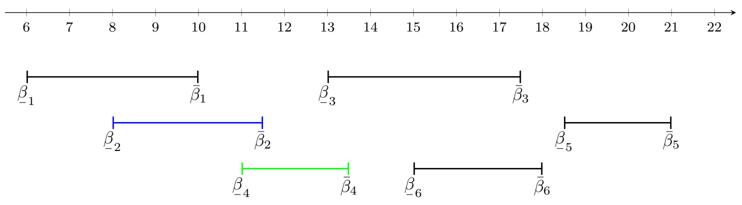


Figure 4.16: Overlapping soft time windows of Service 2.

Now the services of R that are between those of δ must be added to the block. The indices of the services of the block are $i_1 = \min\{1, 2, 4\} = 1$ and $i_2 = \max\{1, 2, 4\} = 4$. Therefore, the list of not checked services is $\hat{\delta} = \bar{\delta} \cup (\{R[i_1], ..., R[i_2]\} \backslash \delta) = \{4\} \cup (\{1, 2, 3, 4\} \backslash \{1, 2, 4\}) = \{3, 4\}$, the block is $\delta = \{R[i_1], ..., R[i_2]\} = \{1, 2, 3, 4\}$, and the free services are $\hat{R} = \hat{R} \cap \delta^c = \{3, 5, 6\} \cap \{1, 2, 3, 4\}^c = \{5, 6\}$.

$k = 3$. Check if any free service has overlapping soft time window with 3:

$l = 5$. In this case $\bar{\beta}_3 = 690$ and $\underline{\beta}_5 = 750$. Since $\bar{\beta}_3 \not\geq \underline{\beta}_5$ (line 14), Service 5 is not added to δ.

$l = 6$. In this case $\bar{\beta}_3 = 690$ and $\underline{\beta}_6 = 540$. This means that $\bar{\beta}_3 \geq \underline{\beta}_6$ (line 14). Therefore, the service is added to the block, $\delta = \delta \cup \{6\} = \{1, 2, 3, 4, 6\}$, and to the list of checked services, $\bar{\delta} = \bar{\delta} \cup \{6\} = \{6\}$. The service is removed from the list of free services, $\hat{R} = \hat{R} \backslash \{6\} = \{5\}$, setting $keep = True$.

Figure 4.17 shows that the only free service that has an overlapping soft time window with 3 (blue soft time window) is Service 6 (green soft time window).

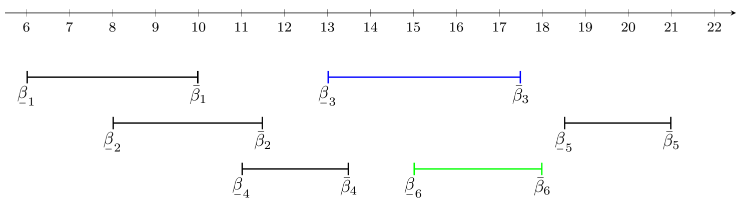


Figure 4.17: Overlapping soft time windows of Service 3.

$k = 4$. Check if any free service has overlapping soft time window with 4:

$l = 5$. In this case $l = 5 = k + 1$ (line 10). Therefore, $\bar{\beta}_4 + \theta_{4,5} = 450 + 5 = 455$ and $\underline{\beta}_5 = 750$. Since $\bar{\beta}_4 + \theta_{4,5} \not\geq \underline{\beta}_5$ (line 11), Service 5 is not added to δ.

Figure 4.18 shows that no free service has an overlapping soft time window with 4 (blue soft time window).

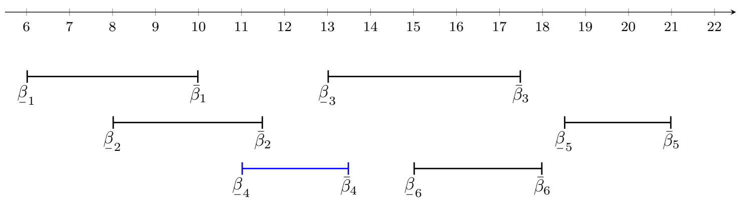


Figure 4.18: Overlapping soft time windows of Service 4.

Now it is necessary to add to δ the services of R that are between those of the block. The indices of the services of the block are $i_1 = \min\{1,2,3,4,6\} = 1$ and $i_2 = \max\{1,2,3,4,6\} = 6$. Therefore, the list of not checked services is $\hat{\delta} = \bar{\delta} \cup (\{R[i_1], ..., R[i_2]\} \backslash \delta) = \{6\} \cup (\{1,2,3,4,5,6\} \backslash \{1,2,4,6\}) = \{3,5,6\}$, the block is $\delta = \{R[i_1], ..., R[i_2]\} = \{1,2,3,4,5,6\}$ and the free services are $\hat{R} = \hat{R} \cap \delta^c = \{3,5,6\} \cap \{1,2,3,4,5,6\}^c = \emptyset$.

Since there are no more free services, $\hat{R} = \emptyset$, the while loop stops and the block is added to the list, $\Delta = \Delta \cup \delta = \{1,2,3,4,5,6\}$.

4.A.2 Schedule the block: *getScheduleBlock*

The function described in Algorithm 4.7 is used to set the starting times of the services of the block in such a way that they are as close as possible.

Algorithm 4.7: getScheduleBlock - Schedule the services of a block

```
  Data: the earliest starting times ($t^e$), the latest starting times ($t^l$), the schedule ($t$) and the
        block ($\delta$)
  //Set the times for the block
1 if $\delta_1 = 1$ then
2 |   $t_{\delta_r} \leftarrow t^e_{\delta_r}$
3 else
4 |   $t_k \leftarrow \max\{t^e_k, t_{k-1} + \eta_{k-1} + \theta_{k-1,k}\}\ \forall k \in \bar{\delta}$
5 $t_k \leftarrow \min\{t^l_k, t_{k+1} - \theta_{k,k+1} - \eta_k\}\ \forall k \in \{\delta_{r-1}, ..., \delta_1\}$
6 return $t = t_j\ \forall j \in \bar{\delta}$
```

Example 4.A.3. Illustration of Algorithm 4.7.

The data provided to this function, given by Example 4.1.2, are: the earliest(t^e) and latest (t^l) start of the services, the schedule of the services ($t_1 = 0$, $t_2 = 120$) and the block to schedule ($\delta = \{3,4\}$).

To obtain the schedule of the block $\delta = \{3,4\}$, the last service of the block is scheduled at its earliest time, according to its predecessors, (lines 3 - 4):

$k = 3$. $t_3 = \max\{t^e_3, t_2 + \eta_2 + \theta_{2,3}\} = \max\{330, 120 + 60 + 5\} = 330$.

$k = 4$. $t_4 = \max\{t^e_4, t_3 + \eta_3 + \theta_{3,4}\} = \max\{395, 330 + 60 + 5\} = 395$.

After that, the other services are scheduled at their latest possible time (line 5).

$k = 3$. $t_3 = \min\{t^l_3, t_4 - \theta_{3,4} - \eta_3\} = \min\{385, 395 - 60 - 5\} = 330$.

This function returns the schedule for the services $\delta = \{3, 4\}$, $t_3 = 330$ and $t_4 = 395$, presented in Figure 4.19.

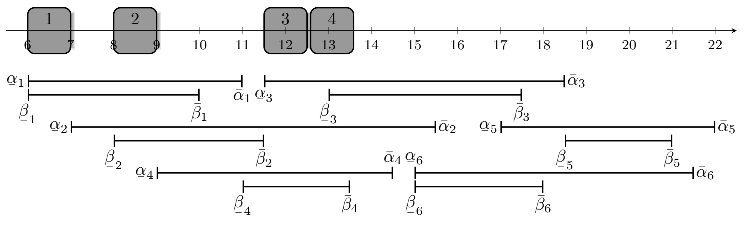


Figure 4.19: Schedule of block $\{3, 4\}$.

Example 4.A.4. Illustration of Algorithm 4.7.

The data provided to this function, given by Example 4.1.2, are: the earliest(t^e) and latest (t^l) start of the services, the schedule of the services ($t_1 = 0$, $t_2 = 120$, $t_3 = 330$ and $t_4 = 395$) and the block to schedule ($\delta = \{5, 6\}$).

To obtain the schedule of the block $\delta = \{5, 6\}$, the last service of the block is scheduled at its earliest time, according to its predecessors (lines 3 - 4):

$k = 5$. $t_5 = \max\{t^e_5, t_4 + \eta_4 + \theta_{4,5}\} = \max\{660, 395 + 60 + 5\} = 660$.

$k = 6$. $t_6 = \max\{t^e_6, t_5 + \eta_5 + \theta_{5,6}\} = \max\{725, 660 + 60 + 5\} = 725$.

After that, the other services are scheduled at their latest possible time (line 5).

$k = 5$. $t_5 = \min\{t^l_5, t_6 - \theta_{5,6} - \eta_5\} = \min\{805, 725 - 60 - 5\} = 660$.

This function returns the schedule for the services $\delta = \{5, 6\}$, $t_5 = 660$ and $t_6 = 725$, presented in Figure 4.20.

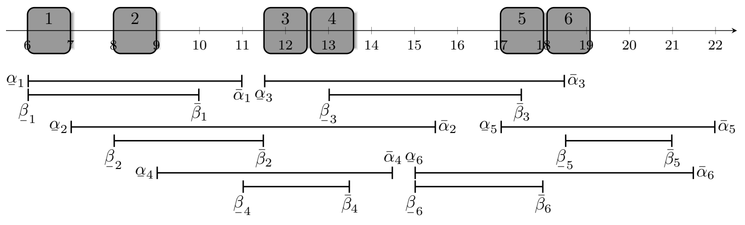


Figure 4.20: Schedule of block $\{5, 6\}$.

4.A.3 Delay the block: *delayBlock*

Algorithm 4.8 delays the block in order to reduce the penalization of the soft time windows. It obtains the set of all the possible times that the block can be delayed to change its penalization (line 4). If the delay of the block increases the penalization, then the previous block is added (lines 5 - 7). In case the penalization can be reduced delaying the block (line 8), the new schedule for the block is computed (lines 8 - 19).

Algorithm 4.8: delayBlock - Delay the block to improve penalization

```
Data: the earliest starting times ($t^e$), the latest starting times ($t^l$), the schedule ($t$), the block ($\delta$) and the blocks ($\Delta$)
//Delay block to improve penalization
1  $delay \leftarrow True$
2  while $delay = True$ do
       //Obtain the maximum time that the block can be delayed
3      $\lambda_d \leftarrow \min_{k\in\delta}\{t^l_k - t_k\}$
       //Obtain all the possible times that the block can be delayed
4      $E$, $\sigma \leftarrow$ getAllDelays($\delta$, $\lambda_d$, $t$)
5      if $\sigma > 0$ and $\delta_1 \neq 1$ then
           //Add the previous block
6          $\bar{\delta} \leftarrow \delta + previousblock$
           //Schedule $\bar{\delta} = \{\delta_1, ..., \delta_r\}$ so the services are as close as possible
7          $t \leftarrow$ getScheduleBlock($t^e, t, \bar{\delta}$)
8      else if $\sigma < 0$ then
           //Get the maximum delay time that reduces the block penalization
9          $\epsilon_{final}$, $\epsilon_{max} \leftarrow$ getMaxDelay($\delta, t$)
10         $t_k \leftarrow t_k + \epsilon_{final}\ \forall k \in \delta$
           //Add following block if necessary
11         if $\bar{\delta}$ is not the last block and $\epsilon_{final} = \epsilon_{max}$ then
12             $\bar{\delta} \leftarrow \bar{\delta} + nextblock$
13             $t_l \leftarrow \max\{\underline{\alpha}_l, t_{l-1} + \eta_{l-1} + \theta_{l-1,l}\}\ \forall l \in nextblock$
               //Obtain all the possible times that the block can be delayed
14             $E$, $\sigma \leftarrow$ getAllDelays($\delta$, $\lambda_d$)
15             if $\sigma > 0$ then
                   //Schedule $\bar{\delta} = \{\delta_1, ..., \delta_r\}$ so the services are as close as possible
16                 $t \leftarrow$ getScheduleBlock($t^e, t, \bar{\delta}$)
17         else
18             $delay \leftarrow False$
19     else
20         $\bar{\delta} \leftarrow \delta$, $delay \leftarrow False$
21 return $\bar{\delta}$, $t$
```

Example 4.A.5. Illustration of Algorithm 4.8.

According to Example 4.1.2, the data are: the earliest (t^e) and latest (t^l) starting times for the services, the schedule ($t_1 = 0$, $t_2 = 120$, $t_3 = 330$ and $t_4 = 395$), the block ($\delta = \{3, 4\}$) and the list of blocks (($\Delta = \{\{1, 2\}, \{3, 4\}, \{5, 6\}\}$).

After setting $delay = True$, the maximum delay time of the block is computed, $\lambda_d = \min_{k\in\delta}\{t^l_k - t_k\} = \min\{385 - 330, 450 - 395\} = 55$. Then, function ***getAllDelays*** (more details in Example 4.A.7) is used to obtain the list of times that the block can be delayed, $E = \{55\}$, and how the penalization will change by delaying it, $\sigma = 0$. Because $\sigma \not< 0$ the block is not delayed (line 8), so $\bar{\delta} = \delta = \{3, 4\}$ and $delay = False$.

This function returns $\bar{\delta} = \{3, 4\}$ and $t_1 = 0$, $t_2 = 120$, $t_3 = 330$ and $t_4 = 395$.

Example 4.A.6. Illustration of Algorithm 4.8.

According to Example 4.1.2, the data are: the earliest (t^e) and latest (t^l) starting times for the services, the schedule ($t_1 = 0$, $t_2 = 120$, $t_3 = 330$, $t_4 = 395$, $t_5 = 660$ and $t_6 = 725$), the block ($\delta = \{5, 6\}$) and the list of blocks (($\Delta = \{\{1, 2\}, \{3, 4\}, \{5, 6\}\}$).

After setting $delay = True$ the maximum delay time of the block is computed, $\lambda_d = \min_{k\in\delta}\{t^l_k - t_k\} = \min\{805 - 660, 870 - 725\} = 145$. Then, function ***getAllDelays*** (see Example 4.A.8 for more information) is used to obtain the list of times that the block can be delayed, $E = \{90, 145\}$, and how the penalization will change by delaying it, $\sigma = 0$. Because $\sigma \not< 0$ the block is not delayed (line 8), so $\bar{\delta} = \delta = \{5, 6\}$ and $delay = False$.

This function returns $\bar{\delta} = \{3, 4\}$ and $t_1 = 0$, $t_2 = 120$, $t_3 = 330$, $t_4 = 395$, $t_5 = 660$ and $t_6 = 725$.

4.A.3.1 Obtain potential delay times: *getAllDelays*

The function presented in Algorithm 4.9 obtains the list of delay times that would change the penalization of the block.

This method obtains the delay times by iterating through the services and checking if they have been scheduled before, within or after their soft time window. The algorithm checks if these situations happen:

a. If a service is scheduled before its soft time window, delaying it would reduce the penalization (lines 3 - 10).

b. If the service is scheduled after the soft time window, delaying it would increase the penalization (lines 11 - 12).

c. If the service is within its soft time window, delaying it until the upper bound of its soft time window would maintain the penalization (lines 13 - 16).

Algorithm 4.9: getAllDelays - Get the possible delay times for the block

```
Data: the block ($\delta$), the maximum delay ($\lambda_d$) and the schedule ($t$)
//Get possible delay times for the block
1  $E \leftarrow \emptyset$, $\sigma \leftarrow 0$
2  for $k \in \delta$ do
3      if $t_k < \underline{\beta}_k$ then
4          $\sigma \leftarrow \sigma - 1$ //The penalization would decrease
5          $d \leftarrow \underline{\beta}_k - t_k$
6          if $d \leq \lambda_d$ and $d \notin E$ then
7              $E \leftarrow E \cup \{d\}$
8              $d \leftarrow \bar{\beta}_k - (t_k + \eta_k)$
9              if $d \leq \lambda_d$ and $d \notin E$ then
10                 $E \leftarrow E \cup \{d\}$
11     else if $t_k + \eta_k \geq \bar{\beta}_k$ then
12         $\sigma \leftarrow \sigma + 1$ //The penalization would increase
13     else if $t_k + \eta_k < \bar{\beta}_k$ then
14         $d \leftarrow \bar{\beta}_k - (t_k + \eta_k)$
15         if $d \leq \lambda_d$ and $d \notin E$ then
16             $E \leftarrow E \cup \{d\}$
17 if $\lambda_d \notin E$ then
18     $E \leftarrow E \cup \{\lambda_d\}$
19 return $E$, $\sigma$
```

Example 4.A.7. Illustration of Algorithm 4.9.

The input data, given by Example 4.A.5, are: the block ($\delta = \{2, 3\}$), the maximum delay ($\lambda_d = 55$) and the schedule ($t_3 = 330$, $t_4 = 395$).

This method starts by initializing the list of delay times, $E = \emptyset$, and the integer that measures the change on the penalization, $\sigma = 0$ (line 1). Then, the delay times are obtained iterating through the services:

$k = 3$. In this case $t_3 = 330$ and $\underline{\beta}_3 = 420$, which means that $t_3 < \underline{\beta}_3$ (line 3). Therefore, $\sigma = \sigma - 1 = -1$ (line 4) and $d = \underline{\beta}_3 - t_3 = 420 - 330 = 90$ (line 5). Since $d \nleq \lambda_d$ (line 6), d is not added to the list E.

$k = 4$. In this case $t_4 = 395$ and $\bar{\beta}_4 = 450$, which means that $t_4 + \eta_4 = 395 + 60 = 455 \geq \bar{\beta}_4$ (line 11). Therefore, $\sigma = \sigma + 1 = -1 + 1 = 0$ (line 12).

Finally, the maximum delay time is added to the list $E = \{55\}$ (lines 18 - 19).

Example 4.A.8. Illustration of Algorithm 4.9.

The input data, given by Example 4.A.5, are: the block ($\delta = \{5, 6\}$), the maximum delay ($\lambda_d = 145$) and the schedule ($t_5 = 660$, $t_6 = 725$).

This method starts by initializing the list of delay times, $E = \emptyset$, and the integer that registers the change on the penalization, $\sigma = 0$ (line 1). Then, the delay times are obtained iterating through the services:

$k = 5$. In this case $t_5 = 660$ and $\underline{\beta}_5 = 750$, which means that $t_5 < \underline{\beta}_5$ (line 3). Therefore, $\sigma = \sigma - 1 = -1$ (line 4) and $d = \underline{\beta}_5 - t_5 = 750 - 660 = 90$ (line 5). Since $d \leq \lambda_d$ (line 6), d is added to the list E, $E = \{90\}$. The delay time get the end of Service 5 with its soft time window is $d = \bar{\beta}_5 - (t_5 + \eta_5) = 900 - (660 + 60) = 180$, which is greater than λ_d (line 9). Therefore, d is not added to E.

$k = 6$. In this case $t_6 = 725$ and $\bar{\beta}_6 = 720$, which means that $t_6 + \eta_6 = 725 + 60 = 785 \geq \bar{\beta}_6$ (line 11). Thus, $\sigma = \sigma + 1 = -1 + 1 = 0$ (line 12).

Finally, the maximum delay time is added to the list $E = \{90, 145\}$ (lines 18 - 19).

4.A.3.2 Get best delay of the block: *getMaxDelay*

The function described in Algorithm 4.10 is used to obtain the maximum time that the block can be delayed, in order to reach its following services. Besides, it also computes the delay time necessary to reduce the penalization as much as possible.

The first step is to obtain the delay time that makes the block end at the beginning of the earliest start of the follower (lines 1 - 3). After that, for each delay time (line 4), the change in the penalization of the block is computed. To this aim, the algorithm iterates through the services (line 6). If the delayed schedule of the service is before its soft time window, then its penalization could be decreased (lines 7 - 8). If the delayed service ends when its soft time window finishes, or after, the penalization of the service would only increase if it is delayed more time (lines 9 - 10). The maximum delay time is the one that, if delaying the block more, the penalization would not decrease (lines 11 - 12). In case the penalization always decreases, the delay time would be the last one of them (lines 13 - 14).

Algorithm 4.10: getMaxDelay - Get maximum delay time of the block

```
   Data: the block ($\delta$) and the schedule ($t$)
   //Add the delay time needed to reach the follower block
1  if $\bar{\delta}$ is not the last block then
2      $\epsilon_{max} \leftarrow b^e_{\delta_r+1} - (t_{\delta_r} + \eta_{\delta_r} + \theta_{\delta_r,\delta_r+1})$
3      $E \leftarrow \{\epsilon_i \in E : \epsilon_i < \epsilon_{max}\} \cup \{\epsilon_{max}\}$
   //Delay time for the block
4  for $\epsilon \in sorted(E)$ do
5      $\sigma \leftarrow 0$
6      for $k \in \delta$ do
7          if $t_k + \epsilon < \underline{\beta}_k$ then
8              $\sigma \leftarrow \sigma - 1$
9          else if $t_k + \epsilon + \eta_k \geq \bar{\beta}_k$ then
10             $\sigma \leftarrow \sigma + 1$
11     if $\sigma \geq 0$ then
12         $\epsilon_{final} \leftarrow \epsilon$, exit loop
13     else if $\epsilon = \epsilon_{max}$ then
14         $\epsilon_{final} \leftarrow \epsilon_{max}$
15 return $\epsilon_{final}$, $\epsilon_{max}$
```

4.A.4 Get blocks of consecutive services: *getBlocksConsecutiveServices*

Algorithm 4.11 is used to divide the route into blocks, separated by breaks in the schedule.

Algorithm 4.11: getBlocksConsecutiveServices - Separate the route into blocks

```
   Data: the route ($R$) and the schedule ($t$)
   //Get block of consecutive services
1  $\Delta \leftarrow \emptyset$, $\delta \leftarrow \emptyset$
2  for $j \in R$ do
3      if $j \neq r$ then
4          if $t_j + \eta_j + \theta_{j,j+1} < t_{j+1}$ then
5              $\delta \leftarrow \delta \cup \{j\}$
6              $\Delta \leftarrow \Delta \cup \delta$
7              $\delta \leftarrow \emptyset$
8          else
9              $\delta \leftarrow \delta \cup \{j\}$
10     else
11         $\delta \leftarrow \delta \cup \{j\}$
12         $\Delta \leftarrow \Delta \cup \delta$
13 return $\Delta$
```

Example 4.A.9. Illustration of Algorithm 4.11.

According to Example 4.1.3, the input data are: the route (R) and the schedule ($t_1 = 0$, $t_2 = 120$, $t_3 = 330$, $t_4 = 395$, $t_5 = 660$ and $t_6 = 725$).

After initializing the block, $\delta = \emptyset$, and the list of blocks, Δ (line 1), the algorithm iterates over the services:

$j = 1$. In this case $t_1 + \eta_1 + \theta_{1,2} = 0 + 60 + 5 = 65$ and $t_2 = 120$, that is, $t_1 + \eta_1 + \theta_{1,2} < t_2$. Therefore, there is a break between 1 and 2 so the service forms a block: $\delta = \delta \cup \{1\} = \{1\}$, $\Delta = \Delta \cup \delta = \{\{1\}\}$ and $\delta = \emptyset$.

$j = 2$. In this case $t_2 + \eta_2 + \theta_{2,3} = 120 + 60 + 5 = 185$ and $t_3 = 330$, that is, $t_2 + \eta_2 + \theta_{2,3} < t_3$. Therefore, there is a break between 2 and 3 so the service forms a block: $\delta = \delta \cup \{j\} = \{2\}$, $\Delta = \Delta \cup \delta = \{\{1\}, \{2\}\}$ and $\delta = \emptyset$.

$j = 3$. In this case $t_3 + \eta_3 + \theta_{3,4} = 330 + 60 + 5 = 395$ and $t_4 = 395$, that is, $t_3 + \eta_3 + \theta_{3,j+1} \not< t_4$. Therefore, the service is added to the block: $\delta = \delta \cup \{3\} = \{3\}$.

$j = 4$. In this case $t_4 + \eta_4 + \theta_{4,5} = 395 + 60 + 5 = 460$ and $t_5 = 660$, that is, $t_4 + \eta_4 + \theta_{4,5} < t_5$. Therefore, there is a break between 4 and 5 so the service is the last of the block: $\delta = \delta \cup \{4\} = \{3, 4\}$, $\Delta = \Delta \cup \delta = \{\{1\}, \{2\}, \{3, 4\}\}$ and $\delta = \emptyset$.

$j = 5$. In this case $t_5 + \eta_5 + \theta_{5,6} = 660 + 60 + 5 = 725$ and $t_6 = 725$, that is, $t_5 + \eta_5 + \theta_{5,6} \not< t_6$. Therefore, the service is added to the block: $\delta = \delta \cup \{5\} = \{5\}$.

$j = 6$. In this case $j = r$. Therefore, the service is the last one of the block: $\delta = \delta \cup \{6\} = \{5, 6\}$ and $\Delta = \Delta \cup \delta = \{\{1\}, \{2\}, \{3, 4\}, \{5, 6\}\}$

The list of blocks is $\Delta = \{\{1\}, \{2\}, \{3, 4\}, \{5, 6\}\}$.

4.A.5 Get earliest and latest starting times of the block: *getBlocksEarliestLatestStart*

The function described in Algorithm 4.12 obtains the earliest and latest starting times for the services, in such a way that moving the schedule of the blocks within these times would maintain the penalization. This is done by obtaining all possible delay and advance times of the block (lines 5 - 7) and choosing the ones that do not increase the penalization (lines 8 - 11). Finally, the time window of the block is adjusted, in order to guarantee that all blocks can be scheduled at their earliest and latest times (line 12).

Algorithm 4.12: getBlocksEarliestLatestStart - Get earliest and latest start for the services

```
   Data: the earliest starting times ($t^e$), the latest starting times ($t^l$), the blocks ($\Delta$) and the
         schedule ($t$)
   //Get starting times for each block so the penalization is maintained
1  for $\delta \in \Delta$ do
2      if $\delta$ only has one service, $j$ then
3          $a^e_j \leftarrow b^e_j$, $a^l_j \leftarrow b^l_j$
4      else
           //Get maximum possible delay and advance times for the block
5          $\lambda_a \leftarrow \min_{k\in\delta}\{t_k - t^e_k\}$, $\lambda_d \leftarrow \min_{k\in\delta}\{t^l_k - t_k\}$
6          $E_d \leftarrow$ getDelayTimes($t$,$\delta$,$\lambda_d$) //Possible delay times for the block
7          $E_a \leftarrow$ getAdvanceTimes($t$,$\delta$,$\lambda_a$) //Possible advance times for the block
8          $\epsilon_d \leftarrow$ getMaxDelayTime($t$,$\delta$,$E_d$) //Delay time to maintain penalization
9          $\epsilon_a \leftarrow$ getMaxAdvanceTime($t$,$\delta$,$E_d$) //Advance time to maintain penalization
           //Get the time window for the block
10         for $j \in \delta$ do
11             $a^e_j \leftarrow t_j - \epsilon_a$, $a^l_j \leftarrow t_j + \epsilon_d$
   //Adjust the earliest and latest starting times
12 $a^e, a^l \leftarrow$ adjustTimes($a^e$, $a^l$,$\Delta$)
13 return $a^e, a^l$
```

Example 4.A.10. Illustration of Algorithm 4.12.

The input data, obtained from Example 4.1.3, are: the earliest (t^e) an latest (t^l) starting times, the blocks ($\Delta = \{\{1\}, \{2\}, \{3, 4\}, \{5, 6\}\}$) and the schedule ($t_1 = 0$, $t_2 = 120$, $t_3 = 330$, $t_4 = 395$, $t_5 = 660$ and $t_6 = 725$).

For each block, their delay and advance times, so the penalization is maintained, are:

$\delta = \{1\}$. There is only one service in the block. Thus, $a^e_1 = b^e_1 = 0$ and $a^l_1 = b^l_1 = 180$.

$\delta = \{2\}$. There is only one service in the block. Thus, $a^e_2 = b^e_2 = 120$ and $a^l_2 = b^l_2 = 270$.

$\delta = \{3, 4\}$. The maximum advance and delay time of the services is computed: $\lambda_a = \min_{k\in\delta}\{t_k - t^e_k\} = \min\{330 - 330, 395 - 395\} = 0$ and $\lambda_d = \min_{k\in\delta}\{t^l_k - t_k\} = \min\{385 - 330, 450 - 395\} = 55$. Functions ***getDelayTimes*** (more information in Example 4.A.11) and ***getAdvanceTimes*** are used to obtain the possible delay, $E_d = \{55\}$, and advance, $E_a = \{0\}$, times of the block. After that, function ***getMaxDelayTime*** (see Example 4.A.13 for more details) gets the delay time that does not increase the penalization, $\epsilon_d = 55$. Finally, the earliest and latest starting times for the block are obtained:

$j = 3$. $a^e_3 = t_3 - \epsilon_a = t_3 = 330$, $a^l_3 = t_3 + \epsilon_d = 330 + 55 = 385$.

$j = 4$. $a^e_4 = t_4 - \epsilon_a = t_4 = 395$, $a^l_4 = t_4 + \epsilon_d = 395 + 55 = 450$.

$\delta = \{5, 6\}$. The maximum advance and delay time of the services is computed: $\lambda_a = \min_{k\in\delta}\{t_k - t^e_k\} = \min\{660 - 660, 725 - 725\} = 0$ and $\lambda_d = \min_{k\in\delta}\{t^l_k - t_k\} = \min\{805 - 660, 870 - 725\} = 145$. Functions ***getDelayTimes*** (more details can be found in Example 4.A.12) and ***getAdvanceTimes*** are used to obtain the possible delay, $E_d = \{90, 145\}$, and advance, $E_a = \{0\}$, times of the block. After that, function ***getMaxDelayTime*** (see Example 4.A.14 for more information) gets the delay time that does not increase the penalization, $\epsilon_d = 90$. Finally, the earliest and latest starting times for the block are obtained:

$j = 5$. $a^e_5 = t_5 - \epsilon_a = t_5 = 660$, $a^l_5 = t_5 + \epsilon_d = 660 + 90 = 750$.

$j = 6$. $a^e_6 = t_6 - \epsilon_a = t_6 = 725$, $a^l_6 = t_6 + \epsilon_d = 725 + 90 = 815$.

Finally, function ***adjustTimes*** (line 12) adjusts the time window of the blocks, in order to guarantee that all blocks could be scheduled at their earliest and latest starting times (for more information see Example 4.A.15).

The obtained earliest starting times are: $a^e_1 = 0$, $a^e_2 = 120$, $a^e_3 = 330$, $a^e_4 = 395$, $a^e_5 = 660$ and $a^e_6 = 725$. The latest ones are $a^l_1 = 180$, $a^l_2 = 270$, $a^l_3 = 385$, $a^l_4 = 450$, $a^l_5 = 750$ and $a^l_6 = 815$.

4.A.5.1 Get possible delay times of the block: *getDelayTimes*

Algorithm 4.13 obtains the list of delay times for the block that could result in a change on its soft time window penalization. To this aim, the algorithm iterates through the services computing the delay times necessary to change their soft time window penalization.

Algorithm 4.13: getDelayTimes - Get possible delay times for the block

```
Data: the schedule ($t$), the block ($\delta$) and the maximum delay time ($\lambda_d$)
//Get possible delay times for the block
1  $E_d \leftarrow \emptyset$
2  for $k \in \delta$ do
3      if $t_k < \underline{\beta}_k$ then
           //Delay the service to reduce the penalization
4          $d \leftarrow \underline{\beta}_k - t_k$
5          if $d \leq \lambda_d$ and $d \notin E_d$ then
6              $E_d \leftarrow E_d \cup \{d\}$
7          $d \leftarrow \bar{\beta}_k - (t_k + \eta_k)$
8          if $d \leq \lambda_d$ and $d \notin E_d$ then
9              $E_d \leftarrow E_d \cup \{d\}$
10     else if $t_k + \eta_k < \bar{\beta}_k$ then
           //Delay the service to maintain the penalization
11         $d \leftarrow \bar{\beta}_k - (t_k + \eta_k)$
12         if $d \leq \lambda_d$ and $d \notin E_d$ then
13             $E_d \leftarrow E_d \cup \{d\}$
14 if $\lambda_d \notin E_d$ then
15     $E_d \leftarrow E_d \cup \{\lambda_d\}$
16 return $E_d$
```

Example 4.A.11. Illustration of Algorithm 4.13.

The input data, according to Example 4.A.10, are: the schedule ($t_3 = 330$, $t_4 = 395$), the block ($\delta = \{2, 3\}$) and the maximum delay ($\lambda_d = 55$). This method starts by initializing the list of delay times, $E = \emptyset$ (line 1). Then, the delay times are obtained iterating through the services:

$k = 3$. In this case $t_3 = 330$ and $\underline{\beta}_3 = 420$, which means that $t_3 < \underline{\beta}_3$ (line 3). Therefore, $d = \underline{\beta}_3 - t_3 = 420 - 330 = 90$ (line 4). Since $d \nleq \lambda_d$ (line 5), d is not added to E.

$k = 4$. In this case $t_4 = 395$ and $\bar{\beta}_4 = 450$, which means that $t_4 + \eta_4 \geq \bar{\beta}_4$ (line 10).

Finally, the maximum delay time is added to the list $E_d = \{55\}$ (lines 14 - 15).

Example 4.A.12. Illustration of Algorithm 4.13.

The input data, according to Example 4.A.10, are: the schedule ($t_5 = 660$, $t_6 = 725$), the block ($\delta = \{5, 6\}$) and the maximum delay ($\lambda_d = 145$). This method starts by initializing the list of delay times, $E_d = \emptyset$ (line 1). Then the delay times are obtained iterating through the services:

$k = 5$. In this case $t_5 = 660$ and $\underline{\beta}_5 = 750$, which means that $t_5 < \underline{\beta}_5$ (line 3). Therefore, $d = \underline{\beta}_5 - t_5 = 750 - 660 = 90$ (line 4). Since $d \leq \lambda_d$ (line 5), d is added to the list, $E_d = \{90\}$. The delay time needed to have the service end at the same time as its soft time window is $d = \bar{\beta}_5 - (t_5 + \eta_5) = 900 - (660 + 60) = 180$. This value is greater than λ_d (line 9) so it is not added to E_d.

$k = 6$. In this case $t_6 = 725$ and $\bar{\beta}_6 = 720$, which means that $t_6 + \eta_6 \geq \bar{\beta}_6$ (line 10).

Finally, the maximum delay time is added to the list $E_d = \{90, 145\}$ (lines 14 - 15).

4.A.5.2 Get possible advance times of the block: *getAdvanceTimes*

The function described in Algorithm 4.14 obtains the list of advance times that would result in a change on the soft time window penalization of the block. To this aim, the algorithm iterates through the services obtaining the advance times necessary to change their soft time window penalization.

Algorithm 4.14: getAdvanceTimes - Get possible advance times for the block

```
   Data: the schedule ($t$), the block ($\delta$) and the maximum advance time ($\lambda_a$)
   //Get possible advance times for the block
1  $E_a \leftarrow \emptyset$
2  for $k \in \delta$ do
3      if $t_k + \eta_k > \bar{\beta}_k$ then
           //Advance the service to reduce the penalization
4          $a \leftarrow t_k + \eta_k - \bar{\beta}_k$
5          if $d \leq \lambda_a$ and $d \notin E_a$ then
6              $E_a \leftarrow E_a \cup \{d\}$
7          $a \leftarrow t_k - \underline{\beta}_k$
8          if $a \leq \lambda_a$ and $a \notin E_a$ then
9              $E_a \leftarrow E_d \cup \{a\}$
10     else if $t_k > \underline{\beta}_k$ then
           //Advance the service to maintain the penalization
11         $a \leftarrow t_k - \underline{\beta}_k$
12         if $a \leq \lambda_a$ and $a \notin E_a$ then
13             $E_a \leftarrow E_a \cup \{a\}$
14 if $\lambda_a \notin E_a$ then
15     $E_a \leftarrow E_a \cup \{\lambda_a\}$
16 return $E_a$
```

4.A.5.3 Get maximum delay time: *getMaxDelayTime*

Algorithm 4.15 is used to obtain the maximum delay time that maintains the penalization of its soft time windows.

First, the algorithm checks if the delay of the block maintains the penalization (lines 2 - 6). Then, for each delay time (line 5) the change in the blocks penalization is computed. This is done by iterating through the services (line 9). If the delayed schedule of the service starts before its soft time windows, its penalization could be decreased (lines 10 -11). If the delayed service ends after, or at, its soft time window, the penalization of the service would increase if it is delayed more time (lines 12 - 14). The maximum delay time is the highest delay without an increment in the penalization (lines 15 - 16). In case the penalization would never increase, the delay time would be the last one of them (lines 17 - 18).

Algorithm 4.15: getMaxDelayTime - Get maximum delay time for the block

```
Data: the schedule ($t$), the block ($\delta$)and the delay times ($E_d$)
//Get delay time so the penalization is maintained
1  $\epsilon_d \leftarrow 0$, $\sigma \leftarrow 0$
   //Get how the penalization would change if the service is delayed
2  for $j \in \delta$ do
3      if $t_j < \underline{\beta}_j$ then
4          $\sigma \leftarrow \sigma - 1$ //Delaying the service decreases the penalization
5      else if $t_j + \eta_j \geq \bar{\beta}_j$ then
6          $\sigma \leftarrow \sigma + 1$ //Delaying the service increases the penalization
   //Get how the penalization would change for different delay times
7  if $\sigma \leq 0$ then
8      for $\epsilon \in sorted(E_d)$ do
9          $\sigma_d \leftarrow 0$
10         for $j \in \delta$ do
11             if $t_j + \epsilon < \underline{\beta}_j$ then
12                 $\sigma_d \leftarrow \sigma_d - 1$ //Delaying the service decreases the penalization
13             else if $t_j + \epsilon + \eta_j \geq \bar{\beta}_j$ then
14                 $\sigma_d \leftarrow \sigma_d + 1$ //Delaying the service increases the penalization
15         if $\sigma_d > 0$ then
               //Delaying the block would increase the penalization
16             $\epsilon_d \leftarrow \epsilon$, break loop
17         else if $\epsilon = \max\{E_d\}$ then
18             $\epsilon_d \leftarrow \epsilon$
19 return $\epsilon_d$
```

Example 4.A.13. Illustration of Algorithm 4.15.

According to Example 4.A.10 the input data are: the schedule ($t_3 = 330$, $t_4 = 395$), the block ($\delta = \{2, 3\}$) and the delay times ($E_d = \{55\}$). This method starts initializing the delay time, $\epsilon_d = 0$, and the change of the penalization, $\sigma = 0$ (line 1). Then, it checks if a delay of the block does not affect to its penalization:

$j = 3$. In this case $t_3 = 330$ and $\underline{\beta}_3 = 420$, which means that $t_3 < \underline{\beta}_3$ (line 3). Therefore, the penalization is reduced if the service is delayed $\sigma = \sigma - 1 = -1$ (line 4).

$j = 4$. In this case $t_4 = 395$ and $\bar{\beta}_4 = 450$, which means that $t_4 + \eta_4 = 395 + 60 = 455 \geq \bar{\beta}_4$ (line 5). Therefore, the penalization is increased if the service is delayed $\sigma = \sigma + 1 = 0$ (line 4).

After that, because the penalization remains constant if the block is delayed, the maximum delay time is computed:

$\epsilon = 55$. The change of the penalization is initialized to $\sigma_d = 0$. Then, for each service, the value is updated:

$j = 3$. In this case $t_3 + \epsilon = 330 + 55 = 385$ and $\underline{\beta}_3 = 420$, which means that $t_3 + \epsilon < \underline{\beta}_3$ (line 11). Therefore, the penalization is reduced if the service is delayed, setting $\sigma_d = \sigma_d - 1 = -1$ (line 12).

$j = 4$. In this case $t_4 + \epsilon = 395 + 55 = 450$ and $\bar{\beta}_4 = 450$, which means that $t_4 + \eta_4 \geq \bar{\beta}_4$ (line 13). Therefore, the penalization is increased if the service is delayed, setting $\sigma_d = \sigma_d + 1 = 0$ (line 4).

Since $\epsilon = \max\{E_d\}$, the delay time is $\epsilon_d = \epsilon = 55$.

This function returns the delay time $\epsilon_d = 55$.

Example 4.A.14. Illustration of Algorithm 4.15.

According to Example 4.A.10 the input data are: the schedule ($t_5 = 660$, $t_6 = 725$), the block ($\delta = \{5, 6\}$) and the delay times ($E_d = \{90, 145\}$). This method starts initializing the delay time, $\epsilon_d = 0$, and the change of the penalization, $\sigma = 0$ (line 1). Then, it checks if a delay in the block affects to the penalization:

$j = 5$. In this case $t_5 = 660$ and $\underline{\beta}_5 = 750$, which means that $t_5 < \underline{\beta}_5$ (line 3). Therefore, the penalization is reduced if it is delayed, setting $\sigma = \sigma - 1 = -1$ (line 4).

$j = 6$. In this case $t_6 = 725$ and $\bar{\beta}_6 = 540$, which means that $t_6 + \eta_6 = 725 + 60 = 785 \geq \bar{\beta}_6$ (line 5). Therefore, the penalization is increased if it is delayed, setting $\sigma = \sigma + 1 = 0$ (line 4).

After that, because the penalization remains constant if the block is delayed, the maximum delay time is computed:

$\epsilon = 90$. The change of the penalization is initialized to $\sigma_d = 0$. Then, for each service, the value is updated:

$j = 5$. In this case $t_5 + \epsilon = 660 + 90 = 750$ and $\underline{\beta}_5 = 750$, which means that $t_5 + \epsilon = \underline{\beta}_5$ (line 11). Therefore, the penalization is maintained if it is delayed.

$j = 6$. In this case $t_6 + \epsilon = 725 + 90 = 815$ and $\bar{\beta}_6 = 750$, which means that $t_6 + \epsilon + \eta_6 \geq \bar{\beta}_6$ (line 13). Therefore, the penalization is increased if it is delayed, setting $\sigma_d = \sigma_d + 1 = 1$ (line 4).

Because $\epsilon > 0$ the delay time is $\epsilon_d = \epsilon = 90$ and the loop terminates.

This function returns the delay time $\epsilon_d = 90$.

4.A.5.4 Get maximum advance time: *getMaxAdvanceTime*

Algorithm 4.16 is used to obtain the maximum advance time of the block to maintain its soft time window penalization.

First, the algorithm checks if the penalization is maintained when the block is advanced (lines 2 - 6). Then, for each advance time (line 5), the change in the blocks penalization is computed. This is done by iterating through the services (line 9). If the advanced schedule of the service ends after its soft time windows, its penalization could be decreased (lines 10 -11). If the advanced service starts before, or at, its soft time window, then the penalization of the service would only increase in case it is advanced more time (lines 12 - 14). The maximum advance time is the one that, if the block is delayed more, then the penalization would increase (lines 15 - 16). In case the penalization would never increase, the advance time would be the last one of them (lines 17 - 18).

Algorithm 4.16: getMaxAdvanceTime - Get maximum advance time for the block

```
Data: the schedule ($t$), the block ($\delta$)and the advance times ($E_a$)
//Get advance time so the penalization is maintained
1  $\epsilon_a \leftarrow 0$, $\sigma \leftarrow 0$
   //Get how the penalization would change if the service is advanced
2  for $j \in \delta$ do
3      if $t_j + \eta_j > \bar{\beta}_j$ then
4          $\sigma \leftarrow \sigma - 1$ //Advancing the service reduces the penalization
5      else if $t_j \leq \underline{\beta}_j$ then
6          $\sigma \leftarrow \sigma + 1$ //Advancing the service increases the penalization
   //Get how the penalization would change for different advance time
7  if $\sigma \leq 0$ then
8      for $\epsilon \in sorted(E_a)$ do
9          $\sigma_a \leftarrow 0$
10         for $j \in \delta$ do
11             if $t_j - \epsilon + \eta_j > \bar{\beta}_j$ then
12                 $\sigma_a \leftarrow \sigma_a - 1$ //Advancing the service reduces the penalization
13             else if $t_j - \epsilon \leq \underline{\beta}_j$ then
14                 $\sigma_a \leftarrow \sigma_a + 1$ //Advancing the service increases the penalization
15         if $\sigma_a > 0$ then
               //Advnacing the block increases its penalization
16             $\epsilon_a \leftarrow \epsilon$, break
17         else if $\epsilon = \epsilon_{max}$ then
18             $\epsilon_a \leftarrow \epsilon$
19 return $\epsilon_a$
```

4.A.5.5 Adjust earliest and latest times: *adjustTimes*

Algorithm 4.17 adjusts the earliest and latest starting times of the blocks in order to be able to schedule all of them at the earliest and latest times.

The first part of this function consist in iterating through the services, and updating the earliest start according to the one of its predecessor (lines 1 - 2). Then, the algorithm iterates through the services in reverse order, and updates the latest start according to the one of its follower (lines 3 - 4).

Algorithm 4.17: adjustTimes -Adjust the earliest and latest starts of the blocks

```
Data: the earliest starting times ($a^e$), the latest starting times ($a^l$) and the blocks ($\Delta$)
//Guarantee that blocks can start at the beginning of their time window
1  for $j \in R$ do
2      $a^e_j \leftarrow \max\{a^e_j, a^e_{j-1} + \eta_{j-1} + \theta_{j-1,j}\}$
   //Guarantee that blocks can start at the ending of their time window
3  for $j \in reversed(R)$ do
4      $a^l_j \leftarrow \min\{a^l_j, a^l_{j+1} - \theta_{j,j+1} - \eta_j\}$
5  return $a^e, a^l \ \forall j \in R$
```

Example 4.A.15. Illustration of Algorithm 4.17.

The input data, given by Example 4.A.10, are: the earliest starting times ($a^e_1 = 0$, $a^e_2 = 120$, $a^e_3 = 330$, $a^e_4 = 395$, $a^e_5 = 660$ and $a^e_6 = 725$), the latest starting times ($a^l_1 = 180$, $a^l_2 = 270$,

$a_3^l = 385$, $a_4^l = 450$, $a_5^l = 750$ and $a_6^l = 815$) and blocks ($\Delta = \{\{1\}, \{2\}, \{3, 4\}, \{5, 6\}\}$). First, the earliest starting times are updated:

$j = 2$. $a_2^e = \max\{a_2^e, a_1^e + \eta_1 + \theta_{1,2}\} = \max\{120, 0 + 60 + 5\} = 120$.

$j = 3$. $a_3^e = \max\{a_3^e, a_2^e + \eta_2 + \theta_{2,3}\} = \max\{330, 120 + 60 + 5\} = 330$.

$j = 4$. $a_4^e = \max\{a_4^e, a_3^e + \eta_3 + \theta_{3,4}\} = \max\{395, 330 + 60 + 5\} = 395$.

$j = 5$. $a_5^e = \max\{a_5^e, a_5^e + \eta_5 + \theta_{4,5}\} = \max\{660, 395 + 60 + 5\} = 660$.

$j = 6$. $a_6^e = \max\{a_6^e, a_6^e + \eta_6 + \theta_{5,6}\} = \max\{725, 660 + 60 + 5\} = 725$.

Then, the latest starting times are updated:

$j = 5$. $a_5^e = \min\{a_5^l, a_6^l - \theta_{5,6} - \eta_5\} = \min\{750, 815 - 5 - 60\} = 750$.

$j = 4$. $a_4^e = \min\{a_4^l, a_5^l - \theta_{4,5} - \eta_4\} = \min\{450, 750 - 5 - 60\} = 450$.

$j = 3$. $a_3^e = \min\{a_3^l, a_4^l - \theta_{3,4} - \eta_3\} = \min\{385, 450 - 5 - 60\} = 385$.

$j = 2$. $a_2^e = \min\{a_2^l, a_3^l - \theta_{2,3} - \eta_2\} = \min\{270, 385 - 5 - 60\} = 270$.

$j = 1$. $a_1^e = \min\{a_1^l, a_2^l - \theta_{1,2} - \eta_1\} = \min\{180, 270 - 5 - 60\} = 180$.

Chapter 5

Hierarchical approach: cost over welfare

Following on from what was previously explained, this chapter focuses on a new heuristic algorithm designed to establish the schedule of a route. In this case, the schedules will be obtained so the cost of the routes will be prioritized over the welfare of the users (which is the second approach considered to tackle the HCSP, as explained in Section 2.3). The combination of the ALNS with the scheduling algorithm described in this chapter is denoted by ALNS_CW.

The first part of the chapter, Section 5.1, explains the general behavior of the algorithm, describing the most important functions that constitute the method. Next, the algorithm is further elaborated in Appendix 5.A by describing a set of auxiliary functions, which may be helpful to fully understand the algorithm.

5.1 Algorithm to schedule a route prioritizing cost over welfare

Figure 5.1 presents a simple diagram to explain the general scheme of the algorithm designed to obtain the schedule of a route prioritizing the cost over the welfare.

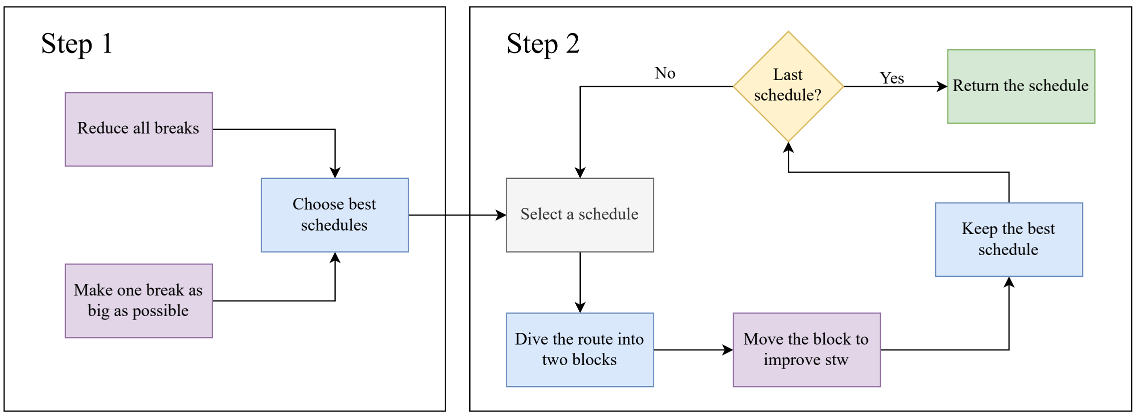


Figure 5.1: Scheme of algorithm ALNS_CW.

The algorithm is divided into two steps: in the first one the schedules with best cost value are found and, in the second step, the schedules are modified in order to improve the soft time window penalization. In the first step there are two options to obtain the schedules: reduce the duration of all the breaks between services or making one of the breaks as big as possible. In the second step, for each schedule obtained before, the route is divided into two blocks (separated by the largest break[1] if it reaches a duration of π_{min}) and the schedule of each block is modified in order to reduce its penalization.

Algorithm 5.1 describes de procedure used to obtain the schedule, for a given route ($R = \{1, ..., r\}$), that first minimizes the cost and second the penalization for carrying out the services outside its preferred time window. It is important to mention that, since the route and its services have been assigned previously to the caregiver, it is not necessary to take care of the affinity levels between services and caregivers. First, the earliest and latest starting times of the services are computed (line 1), and then the schedules with the best cost value are found (line 2). After that, these schedules are updated in order to improve the soft time window penalization (line 3). Finally, the schedule with the best penalization value is chosen (lines 5 - 10).

Algorithm 5.1: ALNS_CW - Schedule optimizing first the cost and second the welfare

```
   Data: the route ($R$)
   //Get earliest and latest times for the services
1  $t^e, t^l \leftarrow$ getEarliestLatest($R$)
   //Get schedules with the best cost
2  $T \leftarrow$ getBestCostSchedules($R$, $t^e$, $t^l$)
   //Get different schedules with the best stw penalization
3  $\hat{T} \leftarrow$ getBestStwSchedules($R$, $T$, $t^e$, $t^l$)
   //Define best schedule as one of $T$
4  $\bar{t} \leftarrow T_0$
   //Get preferred time window penalization
5  $\bar{v} \leftarrow$ stwPenalization($t$)
6  for $\hat{t} \in \hat{T}$ do
       //Get soft time window penalization
7      $\hat{v} \leftarrow$ stwPenalization($\hat{t}$)
       //Update the best schedule for the route
8      if $\hat{v} < \bar{v}$ then
9          $\bar{t} \leftarrow \hat{t}$
10         $\bar{v} \leftarrow \hat{v}$
   //Return the best schedule
11 return $\bar{t}$
```

Now the three most important functions outlined in the algorithm (***getEarliestLatest***, ***getBestCostSchedules*** and ***getBestStwSchedules***) will be explained in detail.

5.1.1 Get earliest and latest times: *getEarliestLatest*

The function presented in Algorithm 5.2 obtains, for the services, the earliest and latest starting times according to the hard time windows.

[1]In case the largest break does not reach a duration of π_{min} the route is considered to be composed by only one block.

Algorithm 5.2: getEarliestLatest - Get earliest times and blocks of services

```
Data: the route ($R$)
//Get earliest start for the services, according to hard time windows
1  for $j \in R$ do
2      if $j = 1$ then
3          $t^e_j \leftarrow \max\{\underline{\alpha}_j, \underline{\gamma}\}$
4      else
5          $t^e_j \leftarrow \max\{\underline{\alpha}_j, t^e_{j-1} + \eta_j + \theta_{j-1,j}\}$
//Get latest start for the service, according to hard time windows
6  for $j \in reversed(R)$ do
7      if $j = r$ then
8          $t^l_j \leftarrow \min\{\bar{\alpha}_j - \eta_j, \bar{\gamma} - \eta_j\}$
9      else
10         $t^l_j \leftarrow \min\{\bar{\alpha}_j - \eta_j, t^l_{j+1} - \theta_{j,j+1} - \eta_j\}$
11 return $t^e, t^l$
```

Example 5.1.1. Illustration of Algorithm 5.2.

To illustrate the algorithm a route composed of 6 services is considered, whose time windows are shown in Figure 5.2. For simplicity, all services have a duration of 1 hour, the travel time between them is 10 minutes and 7:00 is considered to be time 0 of the planning horizon. The available working times for the caregiver are $\underline{\gamma} = 0$ and $\bar{\gamma} = 900$.

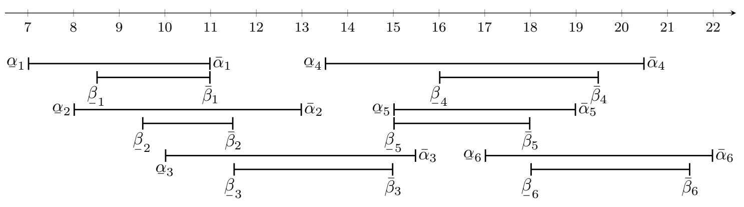


Figure 5.2: Hard and soft time windows of the services (ALNS_CW).

The time windows of the services are presented in Table 5.1.

j	1	2	3	4	5	6
$\underline{\alpha}_j$	0	60	180	390	480	600
$\bar{\alpha}_j$	240	360	510	810	720	900
$\underline{\beta}_j$	90	150	270	540	480	660
$\bar{\beta}_j$	300	270	480	750	660	870

Table 5.1: Hard and soft time windows of the services (ALNS_CW).

The earliest starting times, according to hard time windows, are:

$j = 1.\ t^e_1 = \max\{\underline{\alpha}_1, \underline{\gamma}\} = \max\{0, 0\} = 0.$

$j = 2.\ t^e_2 = \max\{\underline{\alpha}_2, t^e_1 + \eta_1 + \theta_{1,2}\} = \max\{60, 0 + 60 + 10\} = 70.$

$j = 3.\ t^e_3 = \max\{\underline{\alpha}_3, t^e_2 + \eta_2 + \theta_{2,3}\} = \max\{180, 70 + 60 + 10\} = 180.$

$j = 4.\ t^e_4 = \max\{\underline{\alpha}_4, t^e_3 + \eta_3 + \theta_{3,4}\} = \max\{390, 180 + 60 + 10\} = 390.$

$j = 5.\ t^e_5 = \max\{\underline{\alpha}_5, t^e_4 + \eta_4 + \theta_{4,5}\} = \max\{480, 390 + 60 + 10\} = 480.$

$j = 6$. $t_6^e = \max\{\underline{\alpha}_6, t_5^e + \eta_5 + \theta_{5,6}\} = \max\{600, 480 + 60 + 10\} = 600$.

The latest starting times, according to hard time windows, are:

$j = 6$. $t_6^l = \min\{\bar{\alpha}_6 - \eta_6, \bar{\gamma} - \eta_6\} = \min\{900 - 60, 900 - 60\} = 840$.

$j = 5$. $t_5^l = \min\{\bar{\alpha}_5 - \eta_5, t_6^l - \theta_{5,6} - \eta_5\} = \min\{720 - 60, 840 - 10 - 60\} = 660$.

$j = 4$. $t_4^l = \min\{\bar{\alpha}_4 - \eta_4, t_5^l - \theta_{4,5} - \eta_4\} = \min\{810 - 60, 660 - 10 - 60\} = 590$.

$j = 3$. $t_3^l = \min\{\bar{\alpha}_3 - \eta_3, t_4^l - \theta_{3,4} - \eta_3\} = \min\{510 - 60, 590 - 10 - 60\} = 450$.

$j = 2$. $t_2^l = \min\{\bar{\alpha}_2 - \eta_2, t_3^l - \theta_{2,3} - \eta_2\} = \min\{360 - 60, 450 - 10 - 60\} = 300$.

$j = 1$. $t_1^l = \min\{\bar{\alpha}_1 - \eta_1, t_2^l - \theta_{1,2} - \eta_1\} = \min\{240 - 60, 300 - 10 - 60\} = 180$.

The earliest and latest starting times of the services are presented in Figure 5.3. Notice that the hard and soft time windows are presented in grey.

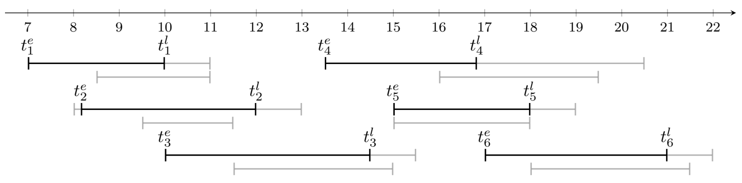


Figure 5.3: Earliest and latest starting times.

5.1.2 Get schedules with best cost value: *getBestCostSchedules*

Algorithm 5.3 describes the procedure used to obtain schedules with the minimum cost value for the route. To this aim, services are divided into two groups: the fixed services, that can only be carried out at one specific time, and the free services, that have a time interval within which they can be performed (lines 2 - 6).

If there are no free services, the schedule is fixed (lines 7 - 9). Otherwise, different sets of services are obtained by taking into account whether it is possible to get a break with a duration equal to or larger than π_{min} (line 11). If the free services are between fixed services and there cannot be a break in the route with a duration of at least π_{min}, the schedule of the free services is obtained according to their soft time windows[2] (lines 12 - 14). Otherwise, the schedule is modified so that the breaks between services are as small as possible (lines 16 -17). Then, the procedure iterates through the free services to get the schedule that makes the break before the service as big as possible (lines 15 - 21).

[2]It means that the cost of the schedule is fixed. Therefore, the schedule of the free services will only affect to the soft time window penalization.

Algorithm 5.3: getBestCostSchedules - Get a schedule with the best value for the cost

```
   Data: the route ($R$), the earliest starting times ($t^e$) and the latest starting times ($t^l$)
   //Define the sets as empty lists
1  $T \leftarrow \emptyset$, $S_{free} \leftarrow \emptyset$
   //Get free and fixed services
2  for $j \in R$ do
3      if $t^e_j \neq t^l_j$ then
4          $S_{free} \leftarrow S_{free} \cup \{j\}$
5      else
6          $S_{fixed} \leftarrow S_{fixed} \cup \{j\}$
   //Get best schedule for the route
7  if $S_{free} = \emptyset$ then
       //Set the schedule as the available starting times
8      $t \leftarrow t^e$
9      $T \leftarrow$ updateBestSchedulesCost($T$, $t$)
10 else
       //Get the sets of services
11     $S1, S2, S3 \leftarrow$ getSetsOfServices($R$, $S_{fixed}$, $t^e$, $t^l$)
       //Set times for the free services
12     if $S_{free} = S2 \cap S3$ then
           //Set the times to reduce stw penal
13         $t \leftarrow$ getStwSchedule($T$, $t$)
14         $T \leftarrow$ updateBestSchedulesCost($T$, $t$)
15     else
           //Set the times to get the breaks as small as possible
16         $t \leftarrow$ firstSchedule(R, $t^e$, $t^l$)
17         $T \leftarrow$ updateBestSchedulesCost($T$, $t$)
           //Set the times to find a big break between services
18         for $j \in R$ do
19             if $j \notin S1$ and $j \neq 1$ then
20                 $t \leftarrow$ secondSchedule($j$, $R$, $t^e$, $t^l$)
21                 $T \leftarrow$ updateBestSchedulesCost($T$, $t$)
   //Return the best schedules
22 return $T$
```

Example 5.1.2. Illustration of Algorithm 5.3.

Continuing with Example 5.1.1, the algorithm starts by initializing $T = \emptyset$ and obtaining the list of free and fixed services. In this case $S_{free} = \{1, 2, 3, 4, 5, 6\}$ because all the services have different earliest and latest starting times (line 3). Then, the sets of services $S1$, $S2$ and $S3$ are obtained using function ***getSetsOfServices*** (line 11). In this case, $S1, S2, S3 = \emptyset$, which means that there can be a break of duration at least π_{min} before each service of the route (for more information see Example 5.A.1).

Two options are contemplated to get schedules. The first one consists in trying to make the breaks between services as small as possible, using function ***firstSchedule***. It consists in scheduling the first service of the route at its latest starting time and, after that, scheduling the remaining services as early as possible. The obtained schedule is: $t_1 = 180$, $t_2 = 250$, $t_3 = 320$, $t_4 = 390$, $t_5 = 480$ and $t_6 = 600$ and it is shown in Figure 5.4 (see Example 5.A.3 for more information), with cost 480. Function ***updateBestSchedulesCost*** is used to keep the best solutions (for more details see Example 5.A.9).

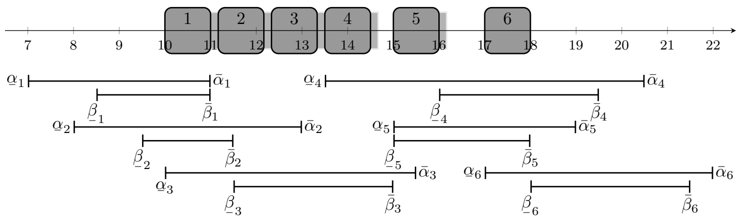


Figure 5.4: Schedule with the smallest breaks.

The second option consists in making one of the breaks as big as possible while reducing the other ones, using function ***secondSchedule***:

$j = 2$. The obtained schedule, presented in Figure 5.5, is: $t_1 = 0$, $t_2 = 300$, $t_3 = 370$, $t_4 = 440$, $t_5 = 510$ and $t_6 = 600$ (for more details see Example 5.A.4), with cost 430.

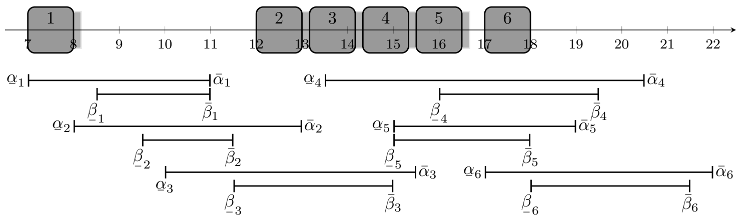


Figure 5.5: Schedule with the largest break between services 1 and 2.

$j = 3$. The obtained schedule, presented in Figure 5.6, is: $t_1 = 0$, $t_2 = 70$, $t_3 = 450$, $t_4 = 520$, $t_5 = 590$ and $t_6 = 660$ (for more details see Example 5.A.5), with cost 410.

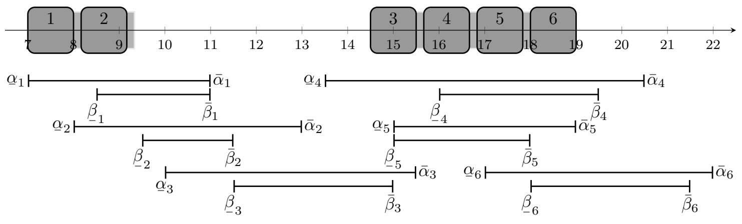


Figure 5.6: Schedule with the largest break between services 2 and 3.

$j = 4$. The obtained schedule, presented in Figure 5.7, is: $t_1 = 40$, $t_2 = 110$, $t_3 = 180$, $t_4 = 590$, $t_5 = 660$ and $t_6 = 730$ (for more details see Example 5.A.6), with cost 410.

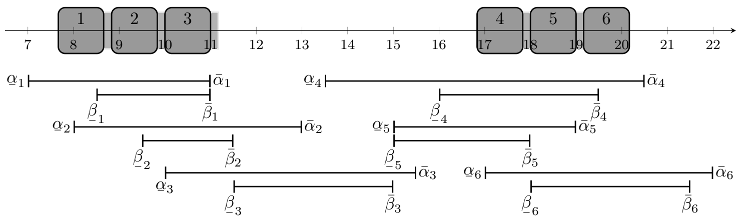


Figure 5.7: Schedule with the largest break between services 3 and 4.

$j = 5$. The obtained schedule, presented in Figure 5.8, is: $t_1 = 180$, $t_2 = 250$, $t_3 = 320$, $t_4 = 390$, $t_5 = 660$ and $t_6 = 730$ (for more details see Example 5.A.7), with cost 410.

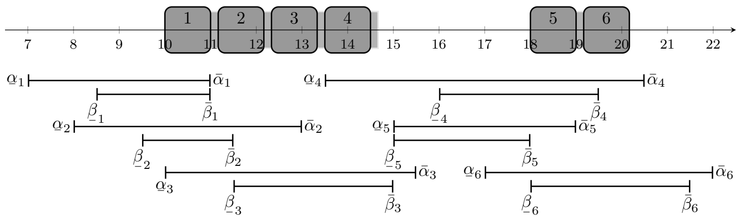


Figure 5.8: Schedule with the largest break between services 4 and 5.

$j = 5$. The obtained schedule, presented in Figure 5.9, is: $t_1 = 180$, $t_2 = 270$, $t_3 = 340$, $t_4 = 410$, $t_5 = 480$ and $t_6 = 840$ (for more details see Example 5.A.8), with cost 430.

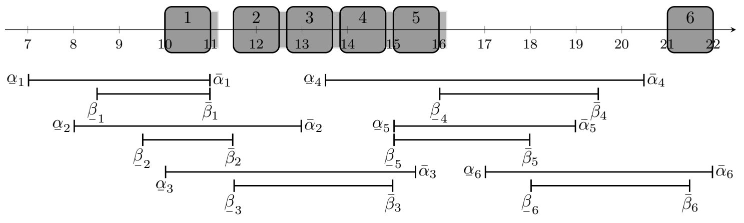


Figure 5.9: Schedule with the largest break between services 5 and 6.

After obtaining each solution, ***updateBestSchedulesCost*** is used to keep the ones with best cost (for more information see Examples 5.A.10, 5.A.11, 5.A.12, 5.A.13 and 5.A.14). The best possible cost is 410, and the list of schedules that reach this value (denoted by T) is given by:

Schedule 1. $t_1 = 0$, $t_2 = 70$, $t_3 = 450$, $t_4 = 520$, $t_5 = 590$ and $t_6 = 660$, which is the solution presented in Figure 5.6.

Schedule 2. $t_1 = 40$, $t_2 = 110$, $t_3 = 180$, $t_4 = 590$, $t_5 = 660$ and $t_6 = 730$, which is the solution presented in Figure 5.7.

Schedule 3. $t_1 = 180$, $t_2 = 250$, $t_3 = 320$, $t_4 = 390$, $t_5 = 660$ and $t_6 = 730$, which is the solution presented in Figure 5.8.

5.1.3 Improve the soft time window penalization: *getBestStwSchedules*

Algorithm 5.4, starting from a list of schedules with the same cost, obtains multiple schedules that maintain the cost but have different soft time window penalization value. Thus, for each of the schedules, it is necessary to obtain the blocks (separated by the largest break with duration of at least π_{min}) that form the schedule (line 2). These blocks are then moved in order to improve their soft time window penalization (lines 10 - 19). Finally, the schedules of both blocks are combined, in order to get a new feasible schedule for the route (line 20).

Algorithm 5.4: GetBestStwSchedules - Get multiple schedules with the same cost

```
Data: the route ($R$), the schedules ($T$), the earliest starting times ($t^e$) and the latest starting
      times ($t^l$)
1  for $t \in T$ do
2      $\Delta \leftarrow$ getBreakBlocks($R$, $t$)
3      if there is only one block then
4          $a^e, a^l \leftarrow$ getBlockTw($R$, $t$, $t^e$, $t^l$)
5          if $a^e_{\delta} \neq t_{\delta_1}$ or $a^e_{\delta_r} \neq t_{\delta_r}$ then
6              $\hat{t} \leftarrow$ moveBlock($\delta$, $t$, $a^e$, $a^l$) //Schedule if the block can be moved
7          else
8              $\hat{t} \leftarrow$ moveServices($\delta$, $t$, $a^e$, $a^l$) //Schedule if the block cannot be moved
9      else
           //Get schedules for the first block ($\hat{\delta}$)
10         $a^e, a^l \leftarrow$ getBlockTw($\hat{\delta}$, $t$, $t^e$, $t^l$)
11         if $a^e_{\hat{\delta}_1} \neq t_{\hat{\delta}_1}$ or $a^l_{\hat{\delta}_1} \neq t_{\hat{\delta}_1}$ then
12             $\hat{t} \leftarrow$ moveBlock($\hat{\delta}$, $t$, $a^e$, $a^l$) //Schedule if the block can be moved
13         else
14             $\hat{t} \leftarrow$ moveServices($\hat{\delta}$, $t$, $a^e$, $a^l$) //Schedule if the block cannot be moved
           //Get schedules for the second block ($\tilde{\delta}$)
15         $a^e, a^l \leftarrow$ getBlockTw($\tilde{\delta}$, $t$, $t^e$, $t^l$)
16         if $a^e_{\tilde{\delta}_1} \neq t_{\tilde{\delta}_1}$ or $a^l_{\tilde{\delta}_1} \neq t_{\tilde{\delta}_1}$ then
17             $\tilde{t} \leftarrow$ moveBlock($\tilde{\delta}$, $t$, $a^e$, $a^l$) //Schedule if the block can be moved
18         else
19             $\tilde{t} \leftarrow$ moveServices($\tilde{\delta}$, $t$, $a^e$, $a^l$) //Schedule if the block cannot be moved
           //Combine the schedules for the blocks
20         $\hat{T} \leftarrow \hat{T} \cup$ getCombinedSchedules($\delta_1$, $\delta_2$, $\hat{t}$, $\tilde{t}$, $R$, $a^l$, $a^e$)
   //Return the schedules
21 return $\hat{T}$
```

Example 5.1.3. Illustration of Algorithm 5.4.

Let us start from the output obtained in Example 5.1.2, that is, a list of three schedules (T) with the same cost value (410). The goal of this example is to modify each of the given schedules to improve their penalization while maintaining the cost.

Schedule 1. The schedule is $t_1 = 0$, $t_2 = 70$, $t_3 = 450$, $t_4 = 520$, $t_5 = 590$ and $t_6 = 660$ (initially presented in Figure 5.6). Function ***getBreakBlocks*** (more information available in Example 5.A.15) is used to divide the route into two blocks: $\Delta = \{\{1, 2\}, \{3, 4, 5, 6\}\}$. Then, the following schedules for the blocks are obtained:

$\hat{\delta} = \{1, 2\}$. The room for manoeuvre of the blocks is given by function ***getBlockTw*** (for more details see Example 5.A.18): $a^e_1 = 0$, $a^e_2 = 70$, $a^l_1 = 180$ and $a^l_2 = 250$ (line 10). In this case, $a^l_{\hat{\delta}_1} = 0 \neq t_{\hat{\delta}_1} = 180$. Therefore, the function ***moveBlock*** (for more details see Example 5.A.24) is used to improve the penalization of the block. The obtained schedule is: $\hat{t}_1 = 90$ and $\hat{t}_2 = 160$ (lines 11 - 12).

The change in the penalization of the block is presented in Figure 5.10. The penalization of the original schedule is $(\underline{\beta}_1 - t_1) + (\underline{\beta}_2 - t_2) = 90 + 150 - 70 = 170$ (see Figure 5.10a), whereas the penalization of the new schedule is 0 (see Figure 5.10b).

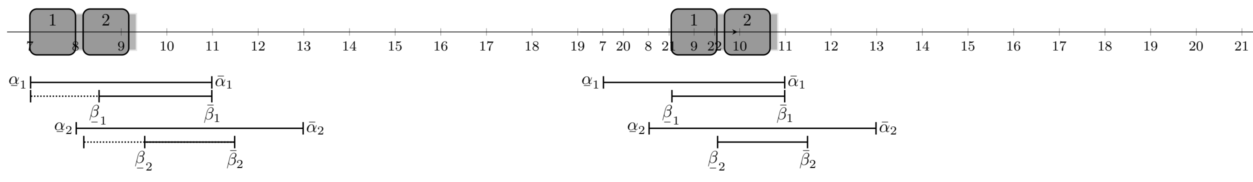

(a) Original schedule. (b) New schedule.

Figure 5.10: Schedule of $\hat{\delta} = \{1, 2\}$.

$\hat{\delta} = \{3, 4, 5, 6\}$. In this case, function ***getBlockTw*** (for more details see Example 5.A.19) returns the following times: $a^e_3 = 390$, $a^e_4 = 460$, $a^e_5 = 530$, $a^e_6 = 600$, $a^l_3 = 450$, $a^l_4 = 520$, $a^l_5 = 590$ and $a^l_6 = 660$ (line 15). Since $a^e_{\hat{\delta}_3} = 390 \neq t_{\hat{\delta}_3} = 450$. Therefore the function ***moveBlock*** is used to improve the penalization of the block (for more details see Example 5.A.25). The obtained schedule is: $\tilde{t}_3 = 450$, $\tilde{t}_4 = 520$, $\tilde{t}_5 = 590$ and $\tilde{t}_6 = 660$ (lines 16 - 17).

Figure 5.11 shows that, in this particular case, both schedules are equal and their penalization is $(t_3 + \eta_3 - \bar{\beta}_3) + (\underline{\beta}_4 - t_4) = (\tilde{t}_3 + \eta_3 - \bar{\beta}_3) + (\underline{\beta}_4 - \tilde{t}_4) = 450 + 60 - 480 + 540 - 520 = 50$.

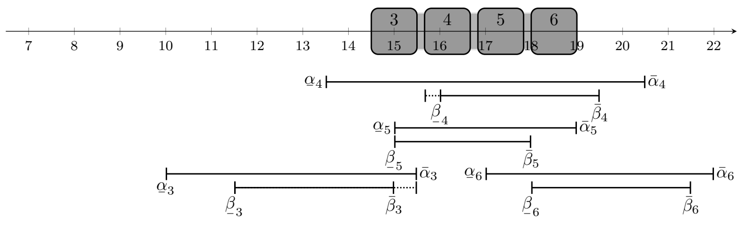


Figure 5.11: Schedule of $\hat{\delta} = \{3, 4, 5, 6\}$.

Finally, the schedules of the block are combined using function ***getCombinedSchedules*** (for more details see Example 5.A.39): $\bar{t}_1 = 90$, $\bar{t}_2 = 160$, $\bar{t}_3 = 450$, $\bar{t}_4 = 520$, $\bar{t}_5 = 590$ and $\bar{t}_6 = 660$.

Schedule 2. The schedule is $t_1 = 40$, $t_2 = 110$, $t_3 = 180$, $t_4 = 590$, $t_5 = 660$ and $t_6 = 730$ (initially presented in Figure 5.7). The function ***getBreakBlocks*** (for more details see Example 5.A.16) divides the route into two blocks: $\Delta = \{\{1, 2, 3\}, \{4, 5, 6\}\}$. Then, the schedule of each block is computed:

$\hat{\delta} = \{1, 2, 3\}$. The times between which the block can be moved are computed with function ***getBlockTw*** (for more details see Example 5.A.20): $a^e_1 = 40$, $a^e_2 = 110$, $a^e_3 = 180$, $a^l_1 = 180$, $a^l_2 = 250$ and $a^l_3 = 320$ (line 10). In this case, $a^l_{\hat{\delta}_1} = 180 \neq t_{\hat{\delta}_1} = 40$. Thus, function ***moveBlock*** (for more details see Example 5.A.26) improves the penalization of the block. The obtained schedule is: $\hat{t}_1 = 130$, $\hat{t}_2 = 200$ and $\hat{t}_3 = 270$ (lines 11 - 12).

The change in the penalization of the block is presented in Figure 5.12. The penalization of the original schedule is $(\underline{\beta}_1 - t_1) + (\underline{\beta}_2 - t_2) + (\underline{\beta}_3 - t_3) = 90 - 40 + 150 - 110 + 270 - 180 = 180$ (see Figure 5.12a), whereas the penalization of the new schedule is 0 (see Figure 5.12b).

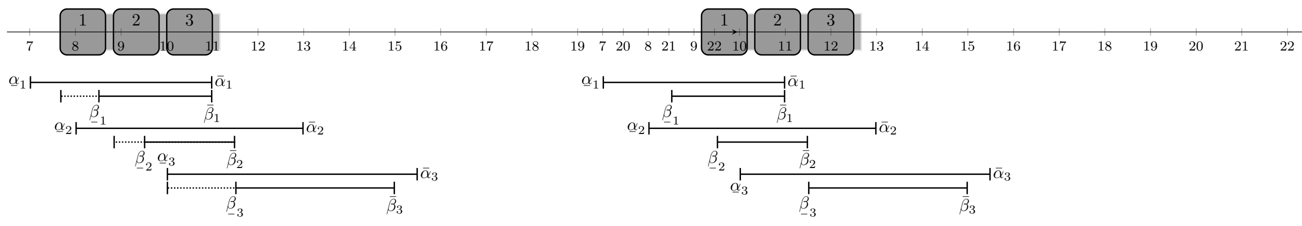

(a) Original schedule. (b) New schedule.

Figure 5.12: Schedule of $\hat{\delta} = \{1, 2, 3\}$.

$\hat{\delta} = \{4, 5, 6\}$. The range of movement per block is given by function ***getBlockTw*** (for more details see Example 5.A.21): $a_4^e = 460$, $a_5^e = 530$, $a_6^e = 600$ and $a_4^l = 590$, $a_5^l = 660$ and $a_6^l = 730$ (line 15). In this case, $a_{\hat{\delta}_4}^e = 460 \neq t_{\hat{\delta}_4} = 590$. Therefore function ***moveBlock*** (for more details see Example 5.A.27) improves the penalization of the block. The obtained schedule is: $\tilde{t}_4 = 530$, $\tilde{t}_5 = 600$ and $\tilde{t}_6 = 670$ (lines 16 - 17).

The change in the penalization of the block is presented in Figure 5.13. The penalization of the original schedule is $t_5 + \eta_5 - \bar{\beta}_5 = 660 + 60 - 660 = 60$ (see Figure 5.13a), whereas the penalization of the new schedule is $\underline{\beta}_4 - \tilde{t}_4 = 540 - 530 = 10$ (see Figure 5.13b).

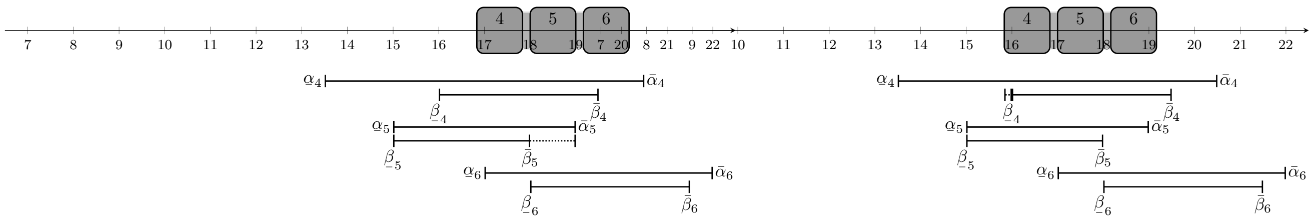

(a) Original schedule. (b) New schedule.

Figure 5.13: Schedule of $\hat{\delta} = \{4, 5, 6\}$.

Finally, both schedules are combined using function ***getCombinedSchedules*** (for more details see Example 5.A.40): $\bar{t}_1 = 130$, $\bar{t}_2 = 200$, $\bar{t}_3 = 270$, $\bar{t}_4 = 530$, $\bar{t}_5 = 600$ and $\bar{t}_6 = 670$.

Schedule 3. The schedule is $t_1 = 180$, $t_2 = 250$, $t_3 = 320$, $t_4 = 390$, $t_5 = 660$ and $t_6 = 730$ (initially presented in Figure 5.8). The function ***getBreakBlocks*** (for more details see Example 5.A.17) divides the route into two blocks: $\Delta = \{\{1, 2, 3, 4\}, \{5, 6\}\}$. Then, the schedule of each block is computed:

$\hat{\delta} = \{1, 2, 3, 4\}$. The range of movement of the block is obtained with function ***getBlockTw*** (for more details see Example 5.A.22): $a_1^e = 180$, $a_2^e = 250$, $a_3^e = 320$, $a_4^e = 390$, $a_1^l = 180$, $a_2^l = 250$, $a_3^l = 320$ and $a_4^l = 390$ (line 10). In this case, $t_1 = a_1^e = a_1^l = 180$, $t_2 = a_2^e = a_2^l = 250$, $t_3 = a_3^e = a_3^l = 320$ and $t_4 = a_4^e = a_4^l = 390$. Therefore, none of the services can be moved $\hat{t}_1 = 180$, $\hat{t}_2 = 250$, $\hat{t}_3 = 320$ and $\hat{t}_4 = 390$. Figure 5.14 shows that the schedule remains unchanged and the penalization is $(t_2 + \eta_2 - \bar{\beta}_2) + (\underline{\beta}_4 - t_4) = (\hat{t}_2 + \eta_2 - \bar{\beta}_2) + (\underline{\beta}_4 - \hat{t}_4) = 250 + 60 - 270 + 540 - 390 = 190$.

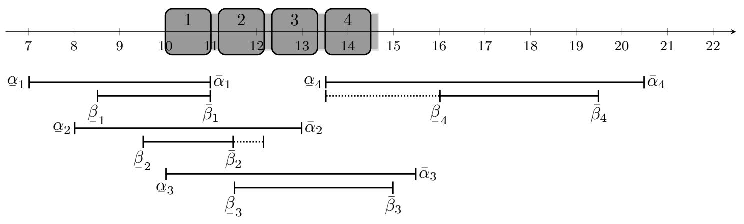
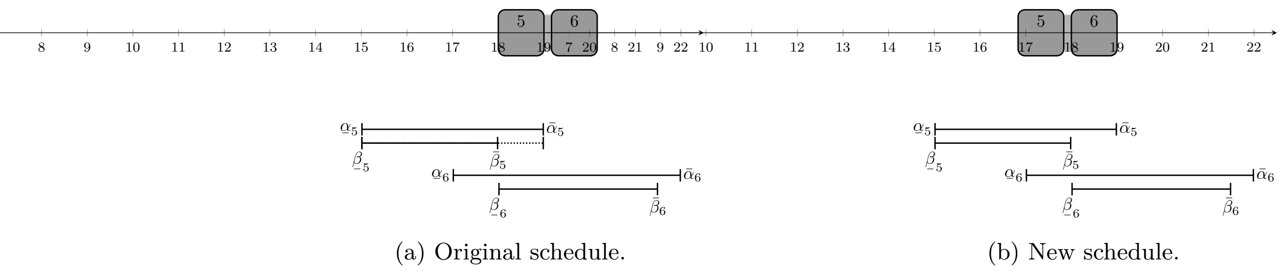


Figure 5.14: Schedule of $\hat{\delta} = \{1, 2, 3, 4\}$.

$\hat{\delta} = \{5, 6\}$. The range of movement of the block is computed with function ***getBlockTw*** (for more details see Example 5.A.23): $a^e_5 = 530$, $a^e_6 = 600$ and $a^l_5 = 660$ and $a^l_6 = 730$ (line 15). In this case, $a^e_{\hat{\delta}_5} = 590 \neq t_{\hat{\delta}_5} = 660$. Therefore function ***moveBlock*** (for more details see Example 5.A.28) improves the penalization of the block. The obtained schedule is: $\tilde{t}_5 = 590$ and $\tilde{t}_6 = 660$ (lines 16 - 17).

The change in the penalization of the block is presented in Figure 5.15. The penalization of the original schedule is $t_5 + \eta_5 - \bar{\beta}_5 = 660 + 60 - 660 = 60$ (see Figure 5.15a), whereas the penalization of the new schedule is 0 (see Figure 5.15b).



(a) Original schedule.

(b) New schedule.

Figure 5.15: Schedule of $\hat{\delta} = \{5, 6\}$.

Finally, the schedules of the block are combined using function ***getCombinedSchedules*** (for more details see Example 5.A.41): $\bar{t}_1 = 180$, $\bar{t}_2 = 250$, $\bar{t}_3 = 320$, $\bar{t}_4 = 390$, $\bar{t}_5 = 590$ and $\bar{t}_6 = 660$.

The list of schedules with best soft time window penalization, $\hat{T}$, is composed by the three solutions obtained before:

Schedule 1. It is represented in Figure 5.16 and the services have the following starting times: $\bar{t}_1 = 90$, $\bar{t}_2 = 160$, $\bar{t}_3 = 450$, $\bar{t}_4 = 520$, $\bar{t}_5 = 590$ and $\bar{t}_6 = 660$, which is the solution obtained combining Figures 5.10b and 5.11. The soft time window penalization of the schedule is $(\bar{t}_3 + \eta_3 - \bar{\beta}_3) + (\underline{\beta}_4 - \bar{t}_4) = (450 + 60 - 480) + (540 - 520) = 30 + 20 = 50$.

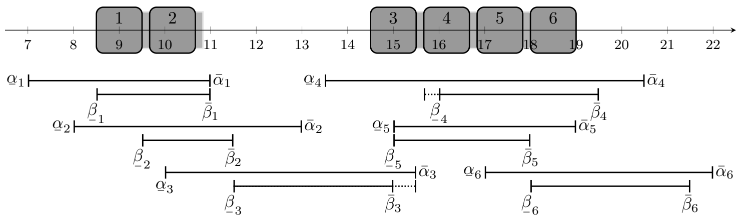


Figure 5.16: Schedule with break between services 2 and 3.

Schedule 2. It is represented in Figure 5.17 and the services have the following starting times: $\bar{t}_1 = 130$, $\bar{t}_2 = 200$, $\bar{t}_2 = 270$, $\bar{t}_4 = 530$, $\bar{t}_5 = 600$ and $\bar{t}_6 = 670$, which is the solution obtained combining Figures 5.12b and 5.13b. The soft time window penalization of the schedule is $\underline{\beta}_4 - \bar{t}_4 = 540 - 530 = 10$.

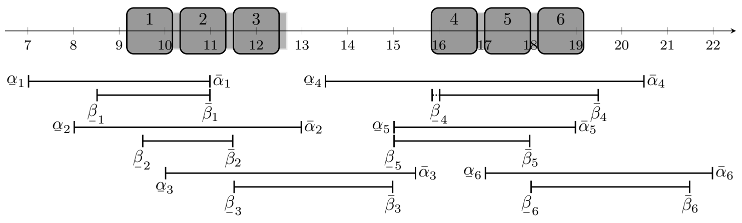


Figure 5.17: Schedule with break between services 3 and 4.

Schedule 3. It is represented in Figure 5.18 and the services have the following starting times: $\bar{t}_1 = 180$, $\bar{t}_2 = 250$, $\bar{t}_3 = 320$, $\bar{t}_4 = 390$, $\bar{t}_5 = 590$ and $\bar{t}_6 = 660$, which is the solution obtained combining Figures 5.14 and 5.15b. The soft time window penalization of the schedule is $(\bar{t}_2 + \eta_2 - \bar{\beta}_2) + (\underline{\beta}_4 - \bar{t}_4) = (250 + 60 - 270) + (540 - 390) = 40 + 150 = 190$.

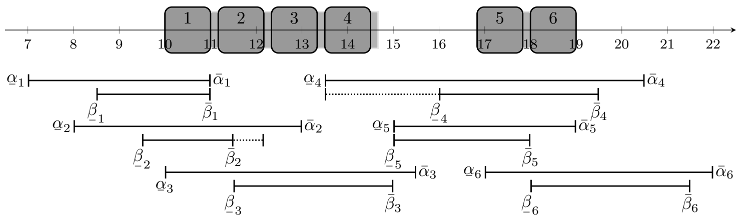


Figure 5.18: Schedule with break between services 4 and 5.

Therefore, the best solution is Schedule 2, giving a cost of 410 and a soft time window penalization of 10.

Appendix 5.A Auxiliary functions

This appendix contains the description in detail of all the auxiliary functions employed by the algorithm to obtain the schedule of a route in order to prioritize its cost over its welfare.

5.A.1 Obtain sets of services: *getSetsOfServices*

Algorithm 5.5 is devoted to assign the services to different sets. It starts obtaining the largest break that can happen before each service (line 1). Then, it iterates through the services and add them to the following sets:

- A service is added to $S1$ if the break before the service does not reach a duration of π_{min} (line 4).
- A service is added to $S2$ if belongs to $S1$ and one of its predecessors is fixed (lines 5 - 6).
- A service is added to $S3$ if belongs to $S1$ and one of its followers is fixed (lines 7 - 8).

Algorithm 5.5: getSetsOfServices - Get the sets of services according to the possibility of having a break bigger than π_{min}

```
Data: the route ($R$), the fixed services ($S_{fixed}$), the earliest starting times ($t^e$) and the latest
      starting times ($t^l$)
//Check between which services there can be a break of $\pi$ or more
1  $D \leftarrow$ maxBreak($R$, $t^e$, $t^l$)
2  $S1, S2, S3 \leftarrow \emptyset$ //Initialize the sets
3  for $j \in \{2, ..., r\}$ do
       //If the break before the service cannot be equal or bigger than $\pi_{min}$
4      if $D_j < \pi_{min}$ then
5          $S1 \leftarrow S1 \cup \{j\}$
           //If any of its predecessors is fixed
6          if $\exists k \in P(j,R)$ such as $k \in S_{fixed}$ then
7              $S2 \leftarrow S2 \cup \{j\}$
           //If any of its followers is fixed
8          if $\exists k \in F(j,R)$ such as $k \in S_{fixed}$ then
9              $S3 \leftarrow S3 \cup \{j\}$
//Return the sets
10 return $S1, S2, S3$
```

Example 5.A.1. Illustration of Algorithm 5.5.

The data given by Example 5.1.2 are: the route ($R = \{1, 2, 3, 4, 5, 6\}$), the fixed services ($S_{fixed} = \emptyset$), earliest start (t^e_j) and latest start (t^l_j).

To obtain the sets of services, first function ***maxBreak*** is used to obtain the largest break that can happen before the each service $D_2 = 230$, $D_3 = 310$, $D_4 = 340$, $D_5 = 200$ and $D_6 = 290$ (more information in Example 5.A.2). Because all of these breaks are larger than $\pi_{min} = 120$, all sets are empty $S1, S2, S3 = \emptyset$.

5.A.1.1 Get potential maximum break before each service: *maxBreak*

Algorithm 5.6 allows us to get the maximum break that can appear before each service of the route. To obtain these breaks, the algorithm iterates through each service and computes the difference between its latest starting time and the earliest end of its predecessor (plus travel time) (line 2).

Algorithm 5.6: maxBreak - Get maximum available break before each service

```
Data: the route ($R$), the earliest starting times ($t^e$) and the latest starting times ($t^l$)
1  for $j \in \{2, ..., r\}$ do
       //Get the maximum possible break between the service and its previous one
2      $D_j \leftarrow t^l_j - (t^e_{j-1} + \eta_{j-1} + \theta_{j-1,j})$
//Return the maximum possible break
3  return $D$
```

Example 5.A.2. Illustration of Algorithm 5.6.

Following Example 5.A.1, the input data are: the route ($R = \{1, 2, 3, 4, 5, 6\}$), the earliest start (t^e_j) and the latest start (t^l_j). For each service, except the first one of the route, the following breaks are found:

$j = 2$. $D_2 = t^l_2 - (t^e_1 + \eta_1 + \theta_{1,2}) = 300 - (0 + 60 + 10) = 230$.

$j = 3$. $D_3 = t^l_3 - (t^e_2 + \eta_2 + \theta_{2,3}) = 450 - (70 + 60 + 10) = 310$.

$j = 4$. $D_4 = t^l_4 - (t^e_3 + \eta_3 + \theta_{3,4}) = 590 - (180 + 60 + 10) = 340$.

$j = 5$. $D_5 = t^l_5 - (t^e_4 + \eta_4 + \theta_{4,5}) = 660 - (390 + 60 + 10) = 200$.

$j = 6$. $D_6 = t^l_6 - (t^e_5 + \eta_5 + \theta_{5,6}) = 840 - (480 + 60 + 10) = 290$.

The result obtained is $D = (D_2, D_3, D_4, D_5, D_6)$.

5.A.2 Obtain the schedule with best penalization value: *getStwSchedule*

The function presented in Algorithm 5.7 is used to schedule the route according to the soft time windows of the services.

The first step is to divide the route into blocks according to their soft time windows (line 1) and obtain the earliest and latest starting times of the services with respect to their soft time windows (lines 2 - 5). Finally, the schedule of the services is found (line 6).

The functions ***getBlocksStw*** and ***getSchedulePenalization*** have been previously described in Sections 4.A.1 and 4.1.2.

Algorithm 5.7: getStwSchedule - Schedule the services according to their soft time window

```
  Data: the route ($R$), the earliest starting times ($t^e$) and the latest starting times ($t^l$)
  //Divide the route into blocks according to stw
1 $\hat{\Delta} \leftarrow$ getBlocksStw($R$)
  //Get earliest start for the services, according to soft time windows
2 for $j \in R$ do
3     $b^e_j \leftarrow \min\{\max\{\underline{\beta}_j, t^e_j\}, t^l_j\}$
  //Get latest start for the services, according to soft time windows
4 for $j \in R$ do
5     $b^l_j \leftarrow \max\{\min\{\bar{\beta}_j - \eta_j, t^l_j\}, t^e_j\}$
  //Get schedule for the services
6 $\bar{t} \leftarrow$ getSchedulePenalization($\hat{t}^e$, $\hat{t}^l$, $b^e$, $b^l$, $\hat{\Delta}$, $R$)
7 return $\epsilon_{final}$
```

5.A.3 Schedule that minimizes all breaks: *firstSchedule*

Algorithm 5.8 computes the schedule of the route so the breaks between services are minimum. To this aim, the first service is scheduled at its latest possible time (line 1). Then, its followers are scheduled at their earliest possible time, according to the new starting time of the first service (lines 2 - 3).

Algorithm 5.8: firstSchedule - Get the schedule with the minimum break between services

```
  Data: the route ($R$), the earliest starting times ($t^e$) and the latest starting times ($t^l$)
  Set the time for the first service as the latest time
1 $t_1 \leftarrow t^l_1$
  Set the time for the followers as the earliest time (given the new time for
  the first service)
2 for $j \in \{2, ..., r\}$ do
3     $t_j \leftarrow \max\{t^e_j, t_{j-1} + \eta_{j-1} + \theta_{j-1,l}\}$
  Return the schedule
4 return $t$
```

Example 5.A.3. Illustration of Algorithm 5.8.

The input data, given by Example 5.1.2, are: the route ($R = \{1, 2, 3, 4, 5, 6\}$), the earliest starting times (t^e_j), the latest starting times (t^l_j).

To make the breaks as small as possible, the first service is scheduled at its latest time and its followers at their earliest time:

$j = 1$. $t_1 = t^l_1 = 180$ (line 1).

$j = 2$. $t_2 = \max\{t^e_2, t_1 + \eta_1 + \theta_{1,2}\} = \max\{70, 180 + 60 + 10\} = 250$ (line 3).

$j = 3$. $t_3 = \max\{t^e_3, t_2 + \eta_2 + \theta_{2,3}\} = \max\{180, 250 + 60 + 10\} = 320$ (line 3).

$j = 4$. $t_4 = \max\{t^e_4, t_3 + \eta_3 + \theta_{3,4}\} = \max\{390, 320 + 60 + 10\} = 390$ (line 3).

$j = 5$. $t_5 = \max\{t^e_5, t_4 + \eta_4 + \theta_{4,5}\} = \max\{480, 390 + 60 + 10\} = 480$ (line 3).

$j = 6$. $t_6 = \max\{t^e_6, t_5 + \eta_5 + \theta_{5,6}\} = \max\{600, 480 + 60 + 10\} = 600$ (line 3).

5.A.4 Schedule that makes one break as big as possible: *secondSchedule*

Algorithm 5.9 schedules the route in such a way that the break between a service and its predecessor is as big as possible. First, the start of the service is set at its latest possible time (line 1). Then, its followers are scheduled at their earliest possible time, according to the new time of the given service (lines 2 - 3). After that, the service's predecessor are scheduled at its earliest time (line 4). Finally, the remaining services are scheduled at their latest possible time, according to the new starting time of the predecessor (lines 5 - 6).

Algorithm 5.9: secondSchedule - Get the schedule with the maximum break between services

```
  Data: the service ($j$), the route ($R$), the earliest starting times ($t^e$) and the latest starting
        times ($t^l$)
  //Set the time for the service at the latest time
1 $t_j \leftarrow t^l_j$
  //Set the time for the followers at the earliest time (according to $j$)
2 for $k \in \{j + 1, ..., r\}$ do
3     $t_k \leftarrow \max\{t^e_k, t_{k-1} + \eta_{k-1} + \theta_{k-1,k}\}$
  //Set the time for the previous service at the earliest time
4 $t_{j-1} \leftarrow t^e_{j-1}$
  //Set the time for the predecessors at the latest time (according to $j - 1$)
5 for $k \in \{j - 2, ..., 1\}$ do
6     $t_k \leftarrow \min\{t^l_k, t_{k+1} - \theta_{k,k+1} - \eta_k\}$
  //Return the schedule
7 return $t$
```

Example 5.A.4. Illustration of Algorithm 5.9.

According to Example 5.1.2, the input data are: the service ($j = 2$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the earliest start (t^e_j) and the latest start (t^l_j).

To make the break as big as possible, the service is scheduled at its latest time and its predecessor at the earliest time:

$k = 2$. $t_2 = t^l_2 = 300$ (line 1).

$k = 3$. $t_3 = \max\{t^e_3, t_2 + \eta_2 + \theta_{2,3}\} = \max\{180, 300 + 60 + 10\} = 370$ (line 3).

$k = 4$. $t_4 = \max\{t_4^e, t_3 + \eta_3 + \theta_{3,4}\} = \max\{390, 370 + 60 + 10\} = 440$ (line 3).

$k = 5$. $t_5 = \max\{t_5^e, t_4 + \eta_4 + \theta_{4,5}\} = \max\{480, 440 + 60 + 10\} = 510$ (line 3).

$k = 6$. $t_6 = \max\{t_6^e, t_5 + \eta_5 + \theta_{5,6}\} = \max\{600, 510 + 60 + 10\} = 600$ (line 3).

$k = 1$. $t_1 = t_1^e = 0$ (line 4).

Example 5.A.5. Illustration of Algorithm 5.9.

According to Example 5.1.2, the input data are: the service ($j = 3$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the earliest start (t_j^e) and the latest start (t_j^l).

To make the break as big as possible, the service is scheduled at its latest time and its predecessor at the earliest time:

$k = 3$. $t_3 = t_3^l = 450$ (line 1).

$k = 4$. $t_4 = \max\{t_4^e, t_3 + \eta_3 + \theta_{3,4}\} = \max\{390, 450 + 60 + 10\} = 520$ (line 3).

$k = 5$. $t_5 = \max\{t_5^e, t_4 + \eta_4 + \theta_{4,5}\} = \max\{480, 520 + 60 + 10\} = 590$ (line 3).

$k = 6$. $t_6 = \max\{t_6^e, t_5 + \eta_5 + \theta_{5,6}\} = \max\{600, 590 + 60 + 10\} = 660$ (line 3).

$k = 2$. $t_2 = t_2^e = 70$ (line 4).

$k = 1$. $t_1 = \min\{t_1^l, t_2 - \theta_{1,2} - \eta_1\} = \min\{180, 70 - 10 - 60\} = 0$ (line 6).

Example 5.A.6. Illustration of Algorithm 5.9.

According to Example 5.1.2, the input data are: the service ($j = 4$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the earliest start (t_j^e) and the latest start (t_j^l).

To make the break as big as possible, the service is scheduled at its latest time and its predecessor at the earliest time:

$k = 4$. $t_4 = t_4^l = 590$ (line 1).

$k = 5$. $t_5 = \max\{t_5^e, t_4 + \eta_4 + \theta_{4,5}\} = \max\{480, 590 + 60 + 10\} = 660$ (line 3).

$k = 6$. $t_6 = \max\{t_6^e, t_5 + \eta_5 + \theta_{5,6}\} = \max\{600, 660 + 60 + 10\} = 730$ (line 3).

$k = 3$. $t_3 = t_3^e = 180$ (line 4).

$k = 2$. $t_2 = \min\{t_2^l, t_3 - \theta_{2,3} - \eta_2\} = \min\{300, 180 - 10 - 60\} = 110$ (line 6).

$k = 1$. $t_1 = \min\{t_1^l, t_2 - \theta_{1,2} - \eta_1\} = \min\{180, 110 - 10 - 60\} = 40$ (line 6).

Example 5.A.7. Illustration of Algorithm 5.9.

According to Example 5.1.2, the input data are: the service ($j = 5$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the earliest start (t_j^e), the latest start (t_j^l).

To make the break as big as possible, the service is scheduled at its latest time and its predecessor at the earliest time:

$k = 5$. $t_5 = t_5^l = 660$ (line 1).

$k = 6$. $t_6 = \max\{t_6^e, t_5 + \eta_5 + \theta_{5,6}\} = \max\{600, 660 + 60 + 10\} = 730$ (line 3).

$k = 4$. $t_4 = t_4^e = 390$ (line 4).

$k = 3$. $t_3 = \min\{t_3^l, t_4 - \theta_{3,4} - \eta_3\} = \min\{450, 390 - 10 - 60\} = 320$ (line 6).

$k = 2$. $t_2 = \min\{t_2^l, t_3 - \theta_{2,3} - \eta_2\} = \min\{300, 320 - 10 - 60\} = 250$ (line 6).

$k = 1$. $t_1 = \min\{t^l_1, t_2 - \theta_{1,2} - \eta_1\} = \min\{180, 250 - 10 - 60\} = 180$ (line 6).

Example 5.A.8. Illustration of Algorithm 5.9.

According to Example 5.1.2, the input data are: the service ($j = 6$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the earliest start (t^e_j) and the latest start (t^l_j).

To make the break as big as possible, the service is scheduled at its latest time and its predecessor at the earliest time:

$k = 6$. $t_6 = t^l_6 = 840$ (line 1).

$k = 5$. $t_5 = t^e_5 = 480$ (line 4).

$k = 4$. $t_4 = \min\{t^l_4, t_5 - \theta_{4,5} - \eta_4\} = \min\{590, 480 - 10 - 60\} = 410$ (line 6).

$k = 3$. $t_3 = \min\{t^l_3, t_4 - \theta_{3,4} - \eta_3\} = \min\{450, 410 - 10 - 60\} = 340$ (line 6).

$k = 2$. $t_2 = \min\{t^l_2, t_3 - \theta_{2,3} - \eta_2\} = \min\{300, 340 - 10 - 60\} = 270$ (line 6).

$k = 1$. $t_1 = \min\{t^l_1, t_2 - \theta_{1,2} - \eta_1\} = \min\{180, 270 - 10 - 60\} = 180$ (line 6).

5.A.5 Update the list of schedules: *updateBestSchedulesCost*

Algorithm 5.10 is used to only keep the schedules that have the best cost value. That is, given a list of schedules and a new solution, the cost of the new solution and the cost of the solutions on the list are obtained (lines 4 -5). Then, the list is updated if the new schedule has a better cost (lines 6 - 7). If the cost is maintained, the new solution is added to the list of best schedules (lines 8 - 9). In case the list is empty, the new schedule is added to it (lines 1 -2).

Algorithm 5.10: updateBestSchedulesCost - Update the best schedules according to the cost of the route

```
   Data: the list of solutions ($T$) and the new solution ($t$)
1  if $T = \emptyset$ then
2      $T \leftarrow T \cup \{t\}$
3  else
       //Update $T$ with the new solution
4      $c \leftarrow$ ScheduleCost($t$)
5      $\hat{c} \leftarrow$ ScheduleCost($\hat{t}$), where $\hat{t} \in T$
6      if $c < \hat{c}$ then
7          $T \leftarrow \{t\}$
8      else if $c = \hat{c}$ then
9          $T \leftarrow T \cup \{t\}$
   //Return the best schedule
10 return $T$
```

Example 5.A.9. Illustration of Algorithm 5.10.

Following Example 5.1.2, the input data are: the list of solutions ($T = \emptyset$) and the new solution ($t = \{t_1 = 180, t_2 = 250, t_3 = 320, t_4 = 390, t_5 = 480$ and $t_6 = 600\}$).

In this case, the list is $T = \{t\}$ because $T = \emptyset$ (lines 1 - 2).

Example 5.A.10. Illustration of Algorithm 5.10.

Following Example 5.1.2, the input data are: the list of solutions (T) and the new solution $t = (t_1 = 0$, $t_2 = 300$, $t_3 = 370$, $t_4 = 440$, $t_5 = 510$ and $t_6 = 600)$.

The largest break of the new schedule happens between services 1 and 2, with a duration of $b = t_2 - (t_1 + \eta_1 + \theta_{1,2}) = 305 - (0 + 60 + 10) = 230$. Therefore, the cost of the schedule is $c = t_6 + \eta_6 - t_1 - \hat{b} = 600 + 60 - 0 - 230 = 430$ (line 4).

The schedule on the list $\hat{t} = (\hat{t}_1 = 180$, $\hat{t}_2 = 250$, $\hat{t}_3 = 320$, $\hat{t}_4 = 390$, $\hat{t}_5 = 480$ and $\hat{t}_6 = 600)$ does not have a break with duration largest than π_{min}, so its cost is $\hat{c} = \hat{t}_6 + \eta_6 - \hat{t}_1 = 600 + 60 - 180 = 480$ (line 5).

The new schedule improves the cost of the list, $c < \hat{c}$ (line 6), which means that the list is updated $T = \{t\}$ (line 7).

Example 5.A.11. Illustration of Algorithm 5.10.

Following Example 5.1.2, the input data are: the list of solutions (T) and the new solution $t = (t_1 = 0$, $t_2 = 70$, $t_3 = 450$, $t_4 = 520$, $t_5 = 590$ and $t_6 = 660)$.

The largest break of the new schedule happens between services 2 and 3, with a duration of $b = t_3 - (t_2 + \eta_2 + \theta_{2,3}) = 450 - (70 + 60 + 10) = 310$. Therefore, the cost of the schedule is $c = t_6 + \eta_6 - t_1 - \hat{b} = 660 + 60 - 0 - 310 = 410$ (line 4). The cost of the schedule on the list is 430, as found in Example 5.A.10 (line 5).

The new schedule improves the cost of the solutions on the list, $c < \hat{c}$ (line 6), which means that the list is updated $T = \{t\}$ (line 7).

Example 5.A.12. Illustration of Algorithm 5.10.

Following Example 5.1.2, the input data are: the list of solutions (T) and the new solution $t = (t_1 = 40$, $t_2 = 110$, $t_3 = 180$, $t_4 = 590$, $t_5 = 660$ and $t_6 = 730)$.

The largest break of the new schedule happens between services 3 and 4, with a duration of $b = t_4 - (t_3 + \eta_3 + \theta_{3,4}) = 590 - (180 + 60 + 10) = 340$. Therefore, the cost of the schedule is $c = t_6 + \eta_6 - t_1 - \hat{b} = 730 + 60 - 40 - 340 = 410$ (line 4). The cost of the schedule on the list is 410, as found in Example 5.A.11 (line 5).

The new schedule has the same cost as the solution on the list, $c = \hat{c}$ (line 8), which means that it is added to the list $T = T \cup \{t\}$ (line 9).

Example 5.A.13. Illustration of Algorithm 5.10.

Following Example 5.1.2, the input data are: the list of solutions (T) and the new solution $t = (t_1 = 180$, $t_2 = 250$, $t_3 = 320$, $t_4 = 390$, $t_5 = 660$ and $t_6 = 730)$.

The largest break of the new schedule happens between services 4 and 5, with a duration of $b = t_5 - (t_4 + \eta_4 + \theta_{4,5}) = 660 - (390 + 60 + 10) = 200$. Therefore, the cost of the schedule is $c = t_6 + \eta_6 - t_1 - \hat{b} = 730 + 60 - 180 - 200 = 410$ (line 4). The cost of the schedules on the list is 410, as found in Example 5.A.12 (line 5).

The new schedule has the same cost as the solutions on the list, $c = \hat{c}$ (line 8), which means that it is added to the list $T = T \cup \{t\}$ (line 9).

Example 5.A.14. Illustration of Algorithm 5.10.

Following Example 5.1.2, the input data are: the list of solutions (T) and the new solution $t = (t_1 = 180$, $t_2 = 270$, $t_3 = 340$, $t_4 = 410$, $t_5 = 480$ and $t_6 = 840)$.

The largest break of the new schedule happens between services 5 and 6, with a duration of $b = t_6 - (t_5 + \eta_5 + \theta_{5,6}) = 840 - (480 + 60 + 10) = 290$. Therefore, the cost of the schedule is $c = t_6 + \eta_6 - t_1 - \hat{b} = 840 + 60 - 180 - 290 = 430$ (line 4). The cost of the schedule on the list is 410, as found in Example 5.A.12 (line 5).

The new schedule has worst cost than the solutions on the list, $c > \hat{c}$, therefore it is not added to the list.

5.A.6 Separate the route into two blocks: *getBreakBlocks*

Algorithm 5.11 presents the function used to separate the route into blocks, divided by the largest break of duration equal to or greater than π_{min}. First, the largest break of the schedule is computed (lines 1 - 6). Then, if the break has a duration equal to or greater than π_{min}, the first block is composed by the services before the break and the second block by the ones after the break (lines 7 - 8). If the break does not reach the specified duration, the route is composed of a single block (lines 9 -10).

Algorithm 5.11: getBreakBlocks - Get blocks of services separated by the largest break ($\geq \pi_{min}$)

```
   Data: the route ($R$) and the schedule ($t$)
1  $\hat{b} \leftarrow 0$
2  for $j \in \{1, ..., r-1\}$ do
       //Get the break between the service and its follower
3      $b \leftarrow t_{j+1} - (t_j + \eta_j + \theta_{j,j+1})$
4      if $b > \hat{b}$ then
5          $\hat{b} \leftarrow b$
6          $k \leftarrow j$
7  if $\hat{b} \geq \pi$ then
8      $\Delta \leftarrow \{\{1, ..., k\}, \{k+1, ..., r\}\}$
9  else
10     $\Delta \leftarrow \{\{1, ..., r\}\}$
   //Return the blocks of services
11 return $\Delta$
```

Example 5.A.15. Illustration of Algorithm 5.11.

The data given by Example 5.1.3 are: the route ($R = \{1, 2, 3, 4, 5, 6\}$) and the schedule ($t_1 = 0$, $t_2 = 70$, $t_3 = 450$, $t_4 = 520$, $t_5 = 590$ and $t_6 = 660$).

First, the largest break of the route is obtained, which is initialized $\hat{b} = 0$:

$j = 1$. The break with $j+1 = 2$ is $b = t_2 - (t_1 + \eta_1 + \theta_{1,2}) = 70 - (0 + 60 + 10) = 0$ (line 3).

$j = 2$. The break with $j+1 = 3$ is $b = t_3 - (t_2 + \eta_2 + \theta_{2,3}) = 450 - (70 + 60 + 10) = 310$ (line 3). In this case $b > \hat{b}$ so $\hat{b} = b$ and $k = j = 2$ (lines 4 - 6).

$j = 3$. The break with $j+1 = 4$ is $b = t_4 - (t_3 + \eta_3 + \theta_{3,4}) = 520 - (450 + 60 + 10) = 0$ (line 3).

$j = 4$. The break with $j+1 = 5$ is $b = t_5 - (t_4 + \eta_4 + \theta_{4,5}) = 590 - (520 + 60 + 10) = 0$ (line 3).

$j = 5$. The break with $j+1 = 6$ is $b = t_6 - (t_5 + \eta_5 + \theta_{5,6}) = 660 - (590 + 60 + 10) = 0$ (line 3).

Thus, the blocks are $\Delta = \{\{1, ..., k\}, \{k+1, ..., r\}\} = \{\{1, 2\}, \{3, 4, 5, 6\}\}$ because $\hat{b} \geq \pi_{min} = 120$ (lines 7 - 8).

Example 5.A.16. Illustration of Algorithm 5.11.

The data given by Example 5.1.3 are: the route ($R = \{1, 2, 3, 4, 5, 6\}$) and the schedule ($t_1 = 40$, $t_2 = 110$, $t_3 = 180$, $t_4 = 590$, $t_5 = 660$, $t_6 = 730$).

First, the largest break of the route is obtained, which is initialized $\hat{b} = 0$:

$j = 1$. The break with $j+1 = 2$ is $b = t_2 - (t_1 + \eta_1 + \theta_{1,2}) = 110 - (40 + 60 + 10) = 0$ (line 3).

$j = 2$. The break with $j+1 = 3$ is $b = t_3 - (t_2 + \eta_2 + \theta_{2,3}) = 180 - (110 + 60 + 10) = 0$ (line 3).

$j = 3$. The break with $j + 1 = 4$ is $b = t_4 - (t_3 + \eta_3 + \theta_{3,4}) = 590 - (180 + 60 + 10) = 340$ (line 3). In this case $b > \hat{b}$ so $\hat{b} = b$ and $k = j = 3$ (lines 4 - 6).

$j = 4$. The break with $j + 1 = 5$ is $b = t_5 - (t_4 + \eta_4 + \theta_{4,5}) = 660 - (590 + 60 + 10) = 0$ (line 3).

$j = 5$. The break with $j + 1 = 6$ is $b = t_6 - (t_5 + \eta_5 + \theta_{5,6}) = 730 - (660 + 60 + 10) = 0$ (line 3).

Then, the blocks are $\Delta = \{\{1, ..., k\}, \{k + 1, ..., r\}\} = \{\{1, 2, 3\}, \{4, 5, 6\}\}$ because $\hat{b} \geq \pi_{min} = 120$ (lines 7 - 8).

Example 5.A.17. Illustration of Algorithm 5.11.

The data given by Example 5.1.3 are: the route ($R = \{1, 2, 3, 4, 5, 6\}$) and the schedule ($t_1 = 180$, $t_2 = 250$, $t_3 = 320$, $t_4 = 390$, $t_5 = 660$, $t_6 = 730$).

First, the largest break of the route is obtained, which is initialized $\hat{b} = 0$:

$j = 1$. The break with $j + 1 = 2$ is $b = t_2 - (t_1 + \eta_1 + \theta_{1,2}) = 250 - (180 + 60 + 10) = 0$ (line 3).

$j = 2$. The break with $j + 1 = 3$ is $b = t_3 - (t_2 + \eta_2 + \theta_{2,3}) = 320 - (250 + 60 + 10) = 0$ (line 3).

$j = 3$. The break with $j + 1 = 4$ is $b = t_4 - (t_3 + \eta_3 + \theta_{3,4}) = 390 - (320 + 60 + 10) = 0$ (line 3).

$j = 4$. The break with $j + 1 = 5$ is $b = t_5 - (t_4 + \eta_4 + \theta_{4,5}) = 660 - (390 + 60 + 10) = 200$ (line 3). In this case $b > \hat{b}$ so $\hat{b} = b$ and $k = j = 4$ (lines 4 - 6).

$j = 5$. The break with $j + 1 = 6$ is $b = t_6 - (t_5 + \eta_5 + \theta_{5,6}) = 730 - (660 + 60 + 10) = 0$ (line 3).

So, the blocks are $\Delta = \{\{1, ..., k\}, \{k + 1, ..., r\}\} = \{\{1, 2, 3, 4\}, \{5, 6\}\}$ because $\hat{b} \geq \pi_{min} = 120$ (lines 7 - 8).

5.A.7 Get earliest and latest start for the block: *getBlockTw*

Algorithm 5.12 is used to obtain the earliest and latest starting time for a block of services. First the amount of time[3] that all services can be either advanced (line 1) or delayed (line 2) is computed. Then, these times are used to obtain the earliest and latest start of the services (lines 3 - 5).

Algorithm 5.12: getBlockTw - Get the earliest and latest start for the block

```
  Data: the block ($\delta$), the schedules ($t$), the earliest starting times ($t^e$) and the latest starting
        times ($t^l$)
  //Get time that all the services of the block can be advanced or delayed
1 $e \leftarrow \min_{j \in \delta}\{t_j - t^e_j\}$
2 $l \leftarrow \min_{j \in \delta}\{t^l_j - t_j\}$
  //Obtain the earliest and latest start
3 for $j \in \delta$ do
4     $a^e_j \leftarrow t_j - e$
5     $a^l_j \leftarrow t_j + l$
  //Return the block earliest and latest start of the block
6 return $a^e, a^l$
```

Example 5.A.18. Illustration of Algorithm 5.12.

According to Example 5.1.3, the input data are: the block ($\delta = \{1, 2\}$), the schedule ($t_1 = 0$, $t_2 = 70$), the earliest (t^e_j) and latest (t^l_j) start.

The time that the block can be advanced is $e = \min\{t_1 - t^e_1, t_2 - t^e_2\} = \min\{0 - 0, 70 - 70\} = 0$ (line 1). The time that the block can be delayed is $l = \min\{t^l_1 - t_1, t^l_2 - t_2\} = \min\{180 - 0, 300 - 70\} = 180$ (line 2). The earliest and latest starting times of the services in the block (lines 3 - 5) are:

[3]Note that the minimum values are chosen because this guarantees that, advancing or delaying the block that amount of time, results in a feasible schedule.

$j = 1$. $a^e_1 = t_1 - e = 0 - 0 = 0$, $a^l_1 = t_1 + l = 0 + 180 = 180$.

$j = 2$. $a^e_2 = t_2 - e = 70 - 0 = 70$, $a^l_2 = t_2 + l = 70 + 180 = 250$.

Example 5.A.19. Illustration of Algorithm 5.12.

According to Example 5.1.3, the input data are: the block ($\delta = \{3, 4, 5, 6\}$), the schedule ($t_3 = 450$, $t_4 = 520$, $t_5 = 590$ and $t_6 = 660$), the earliest (t^e_j) and latest (t^l_j) start.

The time that the block can be advanced is $e = \min\{t_3 - t^e_3, t_4 - t^e_4, t_5 - t^e_5, t_6 - t^e_6\} = \min\{450 - 180, 520 - 390, 590 - 480, 660 - 600\} = 60$ (line 1). The time that the block can be delayed is $l = \min\{t^l_3 - t_3, t^l_4 - t_4, t^l_5 - t_5, t^l_6 - t_6\} = \min\{450 - 450, 590 - 520, 660 - 590, 840 - 660\} = 0$ (line 2). The earliest and latest starting times of the services in the block (lines 3 - 5) are:

$j = 3$. $a^e_3 = t_3 - e = 450 - 60 = 390$, $a^l_3 = t_3 + l = 450 + 0 = 450$.

$j = 4$. $a^e_4 = t_4 - e = 520 - 60 = 460$, $a^l_4 = t_4 + l = 520 + 0 = 520$.

$j = 5$. $a^e_5 = t_5 - e = 590 - 60 = 530$, $a^l_5 = t_5 + l = 590 + 0 = 590$.

$j = 6$. $a^e_6 = t_6 - e = 660 - 60 = 600$, $a^l_6 = t_6 + l = 660 + 0 = 660$.

Example 5.A.20. Illustration of Algorithm 5.12.

According to Example 5.1.3, the input data are: the block ($\delta = \{1, 2, 3\}$), the schedule ($t_1 = 40$, $t_2 = 110$, $t_3 = 180$), the earliest (t^e_j) and latest (t^l_j) start.

The time that the block can be advanced is $e = \min\{t_1 - t^e_1, t_2 - t^e_2, t_3 - t^e_3\} = \min\{40 - 0, 110 - 70, 180 - 180\} = 0$ (line 1). The time that the block can be delayed is $l = \min\{t^l_1 - t_1, t^l_2 - t_2, t^l_3 - t_3\} = \min\{180 - 40, 300 - 110, 450 - 180\} = 140$ (line 2). The earliest and latest starting times of the services (lines 3 - 5) are:

$j = 1$. $a^e_1 = t_1 - e = 40 - 0 = 40$, $a^l_1 = t_1 + l = 40 + 140 = 180$.

$j = 2$. $a^e_2 = t_2 - e = 110 - 0 = 110$, $a^l_2 = t_2 + l = 110 + 140 = 250$.

$j = 3$. $a^e_3 = t_3 - e = 180 - 0 = 180$, $a^l_3 = t_3 + l = 180 + 140 = 320$.

Example 5.A.21. Illustration of Algorithm 5.12.

According to Example 5.1.3, the input data are: the block ($\delta = \{4, 5, 6\}$), the schedule ($t_4 = 590$, $t_5 = 660$, $t_6 = 730$), the earliest (t^e_j) and latest (t^l_j) start.

The time that the block can be advanced is $e = \min\{t_4 - t^e_4, t_5 - t^e_5, t_6 - t^e_6\} = \min\{590 - 410, 660 - 480, 730 - 600\} = 130$ (line 1). The time that the block can be delayed is $l = \min\{t^l_4 - t_4, t^l_5 - t_5, t^l_6 - t_6\} = \min\{590 - 590, 660 - 660, 840 - 730\} = 0$ (line 2). The earliest and latest starting times of the services (lines 3 - 5) are:

$j = 4$ $a^e_4 = t_4 - e = 590 - 130 = 460$, $a^l_4 = t_4 + l = 590 + 0 = 590$.

$j = 5$ $a^e_5 = t_5 - e = 660 - 130 = 530$, $a^l_5 = t_5 + l = 660 + 0 = 660$.

$j = 6$ $a^e_6 = t_6 - e = 730 - 130 = 600$, $a^l_6 = t_6 + l = 730 + 0 = 730$.

Example 5.A.22. Illustration of Algorithm 5.12.

According to Example 5.1.3, the input data are: the block ($\delta = \{1, 2, 3, 4\}$), the schedule ($t_1 = 180$, $t_2 = 250$, $t_3 = 320$, $t_4 = 390$), the earliest (t^e_j) and latest (t^l_j) start.

The time that the block can be advanced is $e = \min\{t_1 - t^e_1, t_2 - t^e_2, t_3 - t^e_3, t_4 - t^e_4\} = \min\{180 - 0, 250 - 70, 320 - 180, 390 - 390\} = 0$ (line 1). The time that the block can be delayed is $l = \min\{t^l_1 - t_1, t^l_2 - t_2, t^l_3 - t_3, t^l_4 - t_4\} = \min\{180 - 180, 300 - 250, 450 - 320, 590 - 390\} = 0$ (line 2). The earliest and latest starting times of the services are equal to the current schedule (because the services cannot be moved). Therefore $a^e_j = a^l_j = t_j$, $\forall j in \delta$.

Example 5.A.23. Illustration of Algorithm 5.12.

According to Example 5.1.3, the input data are: the block ($\delta = \{5, 6\}$), the schedule ($t_5 = 660$, $t_6 = 730$), the earliest (t_j^e) and latest (t_j^l) start.

The time that the block can be advanced is $e = \min\{t_5 - t_5^e, t_6 - t_6^e\} = \min\{660 - 480, 730 - 600\} = 130$ (line 1). The time that the block can be delayed is $l = \min\{t_5^l - t_5, t_6^l - t_6\} = \min\{660 - 660, 840 - 730\} = 0$ (line 2). The earliest and latest starting times of the services (lines 3 - 5) are:

$j = 5$. $a_5^e = t_5 - e = 660 - 130 = 530$, $a_5^l = t_5 + l = 660 + 0 = 660$.

$j = 6$. $a_6^e = t_6 - e = 730 - 130 = 600$, $a_6^l = t_6 + l = 730 + 0 = 730$.

5.A.8 Move the schedule of the block: *moveBlock*

Algorithm 5.13 describes the function used to move a block of services in order to obtain a schedule with better soft time window penalization value. It starts by scheduling all services at their earliest time (line 1) and obtaining the maximum time they can be delayed (line 2). Then, a list of all the delay times of the block that could result in a change of the penalization is obtained (line 3). Finally, the block is delayed the maximum amount of time that decreases the penalization (lines 5 - 7).

Algorithm 5.13: moveBlock - Get possible schedules for a block

```
Data: the block ($\delta$), the schedule ($t$), the earliest starting times ($a^e$) and the latest starting
      times ($a^l$)
  //Schedule the block at its earliest time
1 $\hat{t} \leftarrow a^e$
  //Get maximum delay
2 $\lambda_d \leftarrow a^l_{\delta_1} - a^e_{\delta_1}$
  //Get possible delay times for the block
3 $E, \sigma \leftarrow$ getDelays($\delta, \lambda_d, t$)
  //Delay the bock to improve its penalization
4 if $E \neq \emptyset$ then
5     if $\sigma < 0$ then
          //Get the maximum delay time that reduces the block penalization
6         $\epsilon_{final} \leftarrow$ getBestDelay($\delta, t, E$)
          //Delay the block
7         $\hat{t}_k \leftarrow \hat{t}_k + \epsilon_{final}\ \forall k \in \delta$
  //Return the schedule
8 return $\hat{t}$
```

Example 5.A.24. Illustration of Algorithm 5.13.

The input data, given by Example 5.1.3, are: the block ($R = \{1, 2\}$), the schedule ($t_1 = 0$, $t_2 = 70$), the earliest start ($a_1^e = 0$, $a_2^e = 70$) and the latest start ($a_1^l = 180$, $a_2^l = 250$).

The block is scheduled at its earliest time, $\hat{t}_1 = 0$, $\hat{t}_2 = 70$, and the maximum delay time is computed $\lambda_d = a^l_{\delta_1} - a^e_{\delta_1} = 180 - 0 = 180$. Then, the delay times that would change the penalization of the services, $E = \{80, 90, 180\}$ and $\sigma = -2$, are obtained using function ***getDelays*** (see Example 5.A.29 for more information). To obtain the maximum delay that reduces the penalization of the block, $\epsilon_{final} = 90$, function ***getMaxDelay*** is used (more details in Example 5.A.34). Finally, the block is delayed: $\hat{t}_1 = \hat{t}_1 + \epsilon_{final} = 0 + 90 = 90$, $\hat{t}_2 = \hat{t}_2 + \epsilon_{final} = 70 + 90 = 160$ (line 7).

Example 5.A.25. Illustration of Algorithm 5.13.

The input data, given by Example 5.1.3, are: the block ($R = \{3, 4, 5, 6\}$), the schedule ($t_3 = 450$, $t_4 = 520$, $t_5 = 590$, $t_6 = 660$), the earliest start ($a_3^e = 390$, $a_4^e = 460$, $a_5^e = 530$, $a_6^e = 600$) and the latest start ($a_3^l = 450$, $a_4^l = 520$, $a_5^l = 590$, $a_6^l = 660$).

The block is scheduled at its earliest time, $\hat{t}_3 = 390$, $\hat{t}_4 = 460$, $\hat{t}_5 = 530$, $\hat{t}_6 = 600$, and the maximum delay time is computed $\lambda_d = a_{\delta_3}^l - a_{\delta_3}^e = 450 - 390 = 60$. Then, the delay times that would change the penalization of the services, $E = \{30, 60\}$ and $\sigma = -2$, are obtained using function ***getDelays*** (more information in Example 5.A.30). To obtain the maximum delay that reduces the penalization of the block, $\epsilon_{final} = 60$, function ***getBestDelay*** is used (see Example 5.A.35 for more information). Finally, the block is delayed: $\hat{t}_3 = \hat{t}_3 + \epsilon_{final} = 390 + 60 = 450$, $\hat{t}_4 = \hat{t}_4 + \epsilon_{final} = 460 + 60 = 520$, $\hat{t}_5 = \hat{t}_5 + \epsilon_{final} = 530 + 60 = 590$ and $\hat{t}_6 = \hat{t}_6 + \epsilon_{final} = 600 + 60 = 660$ (line 7).

Example 5.A.26. Illustration of Algorithm 5.13.

The input data, given by Example 5.1.3, are: the block ($R = \{1, 2, 3\}$), the schedule ($t_1 = 40$, $t_2 = 110$, $t_3 = 180$), the earliest start ($a_1^e = 40$, $a_2^e = 110$, $a_3^e = 180$) and the latest start ($a_1^l = 180$, $a_2^l = 250$, $a_3^l = 320$).

The block is scheduled at its earliest time, $\hat{t}_1 = 40$, $\hat{t}_2 = 110$, $\hat{t}_3 = 180$, and the maximum delay time is computed $\lambda_d = a_{\delta_1}^l - a_{\delta_1}^e = 180 - 40 = 140$. Then, the delay times that would change the penalization of the services, $E = \{50, 40, 90, 140\}$ and $\sigma = -3$, are obtained using function ***getDelays*** (see Example 5.A.31 for more details). To obtain the maximum delay that reduces the penalization of the block, $\epsilon_{final} = 90$, function ***getBestDelay*** is used (more information in Example 5.A.36). Finally, the block is delayed: $\hat{t}_1 = \hat{t}_1 + \epsilon_{final} = 40 + 90 = 130$, $\hat{t}_2 = \hat{t}_2 + \epsilon_{final} = 110 + 90 = 200$ and $\hat{t}_3 = \hat{t}_3 + \epsilon_{final} = 180 + 90 = 270$ (line 7).

Example 5.A.27. Illustration of Algorithm 5.13.

The input data, given by Example 5.1.3, are: the block ($R = \{4, 5, 6\}$), the schedule ($t_4 = 590$, $t_5 = 660$, $t_6 = 730$), the earliest start ($a_4^e = 460$, $a_5^e = 530$, $a_6^e = 600$) and the latest start ($a_4^l = 590$, $a_5^l = 660$, $a_6^l = 730$).

The block is scheduled at its earliest time, $\hat{t}_4 = 460$, $\hat{t}_5 = 530$, $\hat{t}_6 = 600$, and the maximum delay time is computed $\lambda_d = a_{\delta_4}^l - a_{\delta_4}^e = 590 - 460 = 130$. Then, the delay times that would change the penalization of the services, $E = \{80, 70, 60, 130\}$ and $\sigma = -2$, are obtained using function ***getDelays*** (see Example 5.A.32 for more details). To obtain the maximum delay that reduces the penalization of the block, $\epsilon_{final} = 70$, function ***getBestDelay*** is used (see more information in Example 5.A.37). Finally, the block is delayed: $\hat{t}_4 = \hat{t}_4 + \epsilon_{final} = 460 + 70 = 530$, $\hat{t}_5 = \hat{t}_5 + \epsilon_{final} = 530 + 70 = 600$ and $\hat{t}_6 = \hat{t}_6 + \epsilon_{final} = 600 + 70 = 670$ (line 7).

Example 5.A.28. Illustration of Algorithm 5.13.

The input data, given by Example 5.1.3, are: the block ($R = \{5, 6\}$), the schedule ($t_5 = 660$, $t_6 = 730$), the earliest start ($a_5^e = 530$, $a_6^e = 600$) and the latest start ($a_5^l = 660$, $a_6^l = 730$).

The block is scheduled at its earliest time, $\hat{t}_5 = 530$, $\hat{t}_6 = 600$, and the maximum delay time is computed $\lambda_d = a_{\delta_5}^l - a_{\delta_5}^e = 660 - 530 = 130$. Then, the delay times that would change the penalization of the services, $E = \{70, 60, 130\}$ and $\sigma = -2$, are obtained using function ***getDelays*** (more information in Example 5.A.33). To obtain the maximum delay that reduces the penalization of the block, $\epsilon_{final} = 60$, function ***getBestDelay*** is used (more details in Example 5.A.38). Finally, the block is delayed: $\hat{t}_5 = \hat{t}_5 + \epsilon_{final} = 530 + 60 = 590$ and $\hat{t}_6 = \hat{t}_6 + \epsilon_{final} = 600 + 60 = 660$ (line 7).

5.A.8.1 Get potential delay times for the block: *getDelays*

The function presented in Algorithm 5.14 obtains the list of delay times that would change the penalization of the block.

This method obtains the delay times by iterating through the services and checking if they are scheduled before, within or after their soft time window. If a service is scheduled before its soft time window, there are two options (lines 3 - 10):

- Delaying it until it reaches the time window, which would reduce its penalization.
- Delaying it more than the time, which would maintain the penalization.

If the service is scheduled after the soft time window, delaying it would increase the penalization (lines 11 - 12). If the service is within its soft time window, delaying it until it finishes at the end of its soft time window would maintain the penalization (lines 13 - 16).

Algorithm 5.14: getDelays - Get the possible delay times for the block

```
Data: the block ($\delta$), the maximum delay ($\lambda_d$) and the schedule ($t$)
//Get possible delay times for the block
1  $E \leftarrow \emptyset$, $\sigma \leftarrow 0$
2  for $k \in \delta$ do
3      if $t_k < \underline{\beta}_k$ then
4          $\sigma \leftarrow \sigma - 1$ //The penalization would decrease
5          $d \leftarrow \underline{\beta}_k - t_k$ //Delay time to reach the soft time window
6          if $d \leq \lambda_d$ and $d \notin E$ then
7              $E \leftarrow E \cup \{d\}$
8              $d \leftarrow \bar{\beta}_k - (t_k + \eta_k)$ //Delay time to reach the end of the soft time window
9              if $d \leq \lambda_d$ and $d \notin E$ then
10                 $E \leftarrow E \cup \{d\}$
11     else if $t_k + \eta_k \geq \bar{\beta}_k$ then
12         $\sigma \leftarrow \sigma + 1$ //The penalization would increase
13     else if $t_k + \eta_k < \bar{\beta}_k$ then
14         $d \leftarrow \bar{\beta}_k - (t_k + \eta_k)$ //Delay time to reach the end of the soft time window
15         if $d \leq \lambda_d$ and $d \notin E$ then
16             $E \leftarrow E \cup \{d\}$
17 if $\lambda_d \notin E$ then
18     $E \leftarrow E \cup \{\lambda_d\}$
19 return $E$, $\sigma$
```

Example 5.A.29. Illustration of Algorithm 5.14.

The input data, given by Example 5.A.24, are: the block ($\delta = \{1, 2\}$), the maximum delay ($\lambda_d = 180$) and the schedule ($t_1 = 0$ and $t_2 = 70$).

This method starts by initializing the list of delay times $E = \emptyset$, and the integer that registers the change on the penalization $\sigma = 0$ (line 1). Then, the delay times are obtained iterating though the services:

$k = 1$. In this case $t_1 = 0$ and $\underline{\beta}_1 = 90$, which means that $t_1 < \underline{\beta}_1$ (line 3). Therefore, $\sigma = \sigma - 1 = -1$ (line 4) and $d = \underline{\beta}_1 - t_1 = 90 - 0 = 90$ (line 5). Since $d \leq \lambda_d$ (line 6), d is added to the list, $E = \{90\}$.

$k = 2$. In this case $t_2 = 70$ and $\underline{\beta}_2 = 150$, which means that $t_2 < \underline{\beta}_2$ (line 3). Therefore, $\sigma = \sigma - 1 = -2$ (line 4) and $d = \underline{\beta}_2 - t_2 = 150 - 70 = 80$ (line 5). Since $d \leq \lambda_d$ (line 6), d is added to the list, $E = \{90, 80\}$.

Finally, the maximum delay time is added to the list $E = \{90, 80, 180\}$ (lines 18 - 19).

Example 5.A.30. Illustration of Algorithm 5.14.

The input data, given by Example 5.A.25, are: the block ($\delta = \{3, 4, 5, 6\}$), the maximum delay ($\lambda_d = 60$) and the schedule ($t_3 = 390$, $t_4 = 460$, $t_5 = 530$ and $t_6 = 600$).

This method starts by initializing the list of delay times $E = \emptyset$, and the integer that registers the change on the penalization $\sigma = 0$ (line 1). Then, the delay times are obtained iterating though the services:

$k = 3$. In this case, $t_3 = 390$ and $\bar{\beta}_3 = 480$, which means that $t_3 + \eta_3 \leq \bar{\beta}_3$ (line 6). Therefore, $d = \bar{\beta}_3 - \eta_3 - t_3 = 480 - 60 - 390 = 30$ (line 8), which is added to the list because $d \leq \lambda_d$, $E = \{30\}$.

$k = 4$. In this case, $t_4 = 460$ and $\underline{\beta}_4 = 540$, which means that $t_4 < \underline{\beta}_4$ (line 3). Therefore, $\sigma = \sigma - 1 = -1$ (line 4) and $d = \underline{\beta}_4 - t_4 = 540 - 460 = 80$ (line 5), which is greater than λ_d (line 9), so d is not added to E.

$k = 5$. In this case $t_5 = 530$ and $\bar{\beta}_5 = 660$, this means that $t_5 + \eta_5 \leq \bar{\beta}_5$ (line 6). Therefore, $d = \bar{\beta}_5 - \eta_5 - t_5 = 660 - 60 - 530 = 70$ (line 8), which is greater than λ_d (line 9), so d is not added to E.

$k = 6$. In this case, $t_6 = 600$ and $\underline{\beta}_6 = 660$, which means that $t_6 < \underline{\beta}_6$ (line 3). Therefore, $\sigma = \sigma - 1 = -2$ (line 4) and $d = \underline{\beta}_6 - t_6 = 660 - 600 = 60$ (line 5). Since $d \leq \lambda_d$ (line 6), d is added to the list, $E = \{30, 60\}$.

Example 5.A.31. Illustration of Algorithm 5.14.

The input data, given by Example 5.A.26, are: the block ($\delta = \{1, 2, 3\}$), the maximum delay ($\lambda_d = 140$) and the schedule ($t_1 = 40$, $t_2 = 110$ and $t_3 = 180$).

This method starts by initializing the list of delay times $E = \emptyset$, and the integer that registers the change on the penalization $\sigma = 0$ (line 1). Then, the delay times are obtained iterating though the services:

$k = 1$. In this case, $t_1 = 40$ and $\underline{\beta}_1 = 90$, which means that $t_1 < \underline{\beta}_1$ (line 3). Therefore, $\sigma = \sigma - 1 = -1$ (line 4) and $d = \underline{\beta}_1 - t_1 = 90 - 40 = 50$ (line 5). Since $d \leq \lambda_d$ (line 6), d is added to the list, $E = \{50\}$.

$k = 2$. In this case, $t_2 = 110$ and $\underline{\beta}_2 = 150$, which means that $t_2 < \underline{\beta}_2$ (line 3). Therefore, $\sigma = \sigma - 1 = -2$ (line 4) and $d = \underline{\beta}_2 - t_2 = 150 - 110 = 40$ (line 5). Since $d \leq \lambda_d$ (line 6), d is added the list, $E = \{50, 40\}$.

$k = 3$. In this case, $t_3 = 180$ and $\underline{\beta}_3 = 270$, which means that $t_3 < \underline{\beta}_3$ (line 3). Therefore, $\sigma = \sigma - 1 = -3$ (line 4) and $d = \underline{\beta}_3 - t_3 = 270 - 180 = 90$ (line 5). Since $d \leq \lambda_d$ (line 6), d is added to the list, $E = \{50, 40, 90\}$.

Finally, the maximum delay time is added to the list $E = \{50, 40, 90, 140\}$ (lines 18 - 19).

Example 5.A.32. Illustration of Algorithm 5.14.

The input data, given by Example 5.A.27, are: the block ($\delta = \{4, 5, 6\}$), the maximum delay ($\lambda_d = 130$) and the schedule ($t_4 = 460$, $t_5 = 530$ and $t_6 = 600$).

This method starts by initializing the list of delay times $E = \emptyset$, and the integer that registers the change on the penalization $\sigma = 0$ (line 1). Then, the delay times are obtained iterating though the services:

$k = 4$. In this case, $t_4 = 460$ and $\underline{\beta}_4 = 540$, which means that $t_4 < \underline{\beta}_4$ (line 3). Therefore, $\sigma = \sigma - 1 = -1$ (line 4) and $d = \underline{\beta}_4 - t_4 = 540 - 460 = 80$ (line 5). Since $d \leq \lambda_d$ (line 6), d is added to the list, $E = \{80\}$.

$k = 5$. In this case, $t_5 = 530$ and $\bar{\beta}_5 = 660$, which means that $t_5 + \eta_5 < \bar{\beta}_5$ (line 3). Therefore, $d = \bar{\beta}_5 - (t_5 + \eta_5) = 660 - (530 + 60) = 70$ (line 5). Because $d \leq \lambda_d$ (line 6), d is added to the list, $E = \{80, 70\}$.

$k = 6$. In this case, $t_6 = 600$ and $\underline{\beta}_6 = 660$, which means that $t_6 < \underline{\beta}_6$ (line 3). Therefore, $\sigma = \sigma - 1 = -2$ (line 4) and $d = \underline{\beta}_6 - t_6 = 660 - 600 = 60$ (line 5). Since $d \leq \lambda_d$ (line 6), d is added to the list, $E = \{80, 70, 60\}$.

Finally, the maximum delay time is added to the list $E = \{80, 70, 60, 130\}$ (lines 18 - 19).

Example 5.A.33. Illustration of Algorithm 5.14.

The input data, given by Example 5.A.28, are: the block ($\delta = \{5, 6\}$), the maximum delay ($\lambda_d = 130$) and the schedule ($t_5 = 530$ and $t_6 = 600$).

This method starts by initializing the list of delay times, $E = \emptyset$, and the integer that registers the change on the penalization, $\sigma = 0$ (line 1). Then, the delay times are obtained iterating though the services:

$k = 5$. In this case, $t_5 = 530$ and $\bar{\beta}_5 = 660$, which means that $t_5 + \eta_5 < \bar{\beta}_5$ (line 3). Therefore, $d = \bar{\beta}_5 - (t_5 + \eta_5) = 660 - (530 + 60) = 70$ (line 5). Since $d \leq \lambda_d$ (line 6), d is added to the list, $E = \{70\}$.

$k = 6$. In this case, $t_6 = 600$ and $\underline{\beta}_6 = 660$, which means that $t_6 < \underline{\beta}_6$ (line 3). Therefore, $\sigma = \sigma - 1 = -2$ (line 4) and $d = \underline{\beta}_6 - t_6 = 660 - 600 = 60$ (line 5). Since $d \leq \lambda_d$ (line 6), d is added to the list, $E = \{70, 60\}$.

Finally, the maximum delay time is added to the list $E = \{70, 60, 130\}$ (lines 18 - 19).

5.A.8.2 Get best delay for the block: *getBestDelay*

Algorithm 5.15 is used to obtain the delay time of the block that reduces its penalization as much as possible. For each delay time, the change in the penalization of the block is computed (lines 1 - 7). The desired delay time is the last one that reduces penalization (lines 8 - 9), that is, a larger delay would maintain or increase the penalization. In case the penalization always decreases, the delay time would be the maximum one (lines 10 - 11).

Algorithm 5.15: getBestDelay - Get maximum delay time for the block

```
Data: the block ($\delta$), the schedule ($t$) and the delay times ($E$)
//Delay time for the block
1  for $\epsilon \in sorted(E)$ do
2      $\sigma \leftarrow 0$
3      for $k \in \delta$ do
4          if $t_k + \epsilon < \underline{\beta}_k$ then
5              $\sigma \leftarrow \sigma - 1$ //The penalization decreases
6          else if $t_k + \epsilon + \eta_k \geq \bar{\beta}_k$ then
7              $\sigma \leftarrow \sigma + 1$ //The penalization increases
8      if $\sigma \geq 0$ then
9          $\epsilon_{final} \leftarrow \epsilon$, exit loop
10     else if $\epsilon = \epsilon_{max}$ then
11         $\epsilon_{final} \leftarrow \max\{E\}$
12 return $\epsilon_{final}$
```

Example 5.A.34. Illustration of Algorithm 5.15.

According to Example 5.A.24, the input data are: the block ($\delta = \{1, 2\}$), the schedule ($t_1 = 0$ and $t_2 = 70$) and the delay times ($E = \{90, 80, 180\}$).

For each delay time it is checked if the penalization is reduced:

$\epsilon = 80$. The penalization change is initialized to $\sigma_d = 0$. Then, it is updated for each service:

- $k = 1$. In this case, $t_1 + \epsilon = 0 + 80 = 80$ and $\underline{\beta}_1 = 90$, which means that $t_1 + \epsilon < \underline{\beta}_1$ (line 4). Therefore, the penalization is reduced if the service is delayed and $\sigma_d = \sigma_d - 1 = -1$ (line 5).
- $k = 2$. In this case, $t_2 + \epsilon = 70 + 80 = 150$ and $\underline{\beta}_2 = 150$, which means that $t_2 + \epsilon = \underline{\beta}_2$ (line 4).

$\epsilon = 90$. The penalization change is initialized to $\sigma_d = 0$. Then, it is updated for each service:

- $k = 1$. In this case, $t_1 + \epsilon = 0 + 90 = 90$ and $\underline{\beta}_1 = 90$, which means that $t_1 + \epsilon = \underline{\beta}_1$ and the penalization does not change.
- $k = 2$. In this case, $t_2 + \epsilon = 70 + 90 = 160$ and $\bar{\beta}_2 = 270$, which means that $t_2 + \epsilon + \eta_2 < \underline{\beta}_2$ and the penalization does not change.

Since $\epsilon = \max\{E_d\}$, the delay time is $\epsilon_d = \epsilon = 90$.

Example 5.A.35. Illustration of Algorithm 5.15.

According to Example 5.A.25 the input data are: the block ($\delta = \{3, 4, 5, 6\}$), the schedule ($t_3 = 390$, $t_4 = 460$, $t_5 = 530$ and $t_6 = 600$) and the delay times ($E = \{30, 60\}$).

For each delay time it is checked if the penalization is reduced:

$\epsilon = 30$. The penalization change is initialized to $\sigma_d = 0$. Then, it is updated for each service:

- $k = 3$. In this case, $t_3 + \epsilon + \eta_3 = 390 + 60 + 30 = 480$ and $\bar{\beta}_3 = 480$, which means that $t_3 + \epsilon + \eta_3 = \bar{\beta}_3$ and $\sigma_d = \sigma_d + 1 = 1$.
- $k = 4$. In this case, $t_4 + \epsilon = 460 + 30 = 490$ and $\underline{\beta}_4 = 540$, which means that $t_4 + \epsilon < \underline{\beta}_4$ and $\sigma_d = \sigma_d - 1 = 0$.
- $k = 5$. In this case, $t_5 + \epsilon + \eta_5 = 530 + 30 + 60 = 620$ and $\bar{\beta}_5 = 660$, this means that $t_5 + \epsilon + \eta_5 \leq \bar{\beta}_5$ and the penalization does not change.
- $k = 6$. In this case, $t_6 + \epsilon = 600 + 30 = 630$ and $\underline{\beta}_6 = 660$, which means that $t_6 + \epsilon < \underline{\beta}_6$ and $\sigma_d = \sigma_d - 1 = -1$.

$\epsilon = 60$. The penalization change is initialized to $\sigma_d = 0$. Then, it is updated for each service:

- $k = 3$. In this case, $t_3 + \epsilon + \eta_3 = 390 + 60 + 60 = 510$ and $\bar{\beta}_3 = 480$, which means that $t_3 + \epsilon + \eta_3 > \bar{\beta}_3$ and $\sigma_d = \sigma_d + 1 = 1$.
- $k = 4$. In this case, $t_4 + \epsilon = 460 + 60 = 520$ and $\underline{\beta}_4 = 540$, which means that $t_4 + \epsilon < \underline{\beta}_4$ and $\sigma_d = \sigma_d - 1 = 0$.
- $k = 5$. In this case, $t_5 + \epsilon + \eta_5 = 530 + 60 + 60 = 650$ and $\bar{\beta}_5 = 660$, this means that $t_5 + \epsilon + \eta_5 \leq \bar{\beta}_5$ and the penalization does not change.
- $k = 6$. In this case, $t_6 + \epsilon = 600 + 60 = 660$ and $\underline{\beta}_6 = 660$, which means that $t_6 + \epsilon = \underline{\beta}_6$ and the penalization does not change.

Since $\sigma_d = 0$, the delay time is $\epsilon_d = \epsilon = 60$.

Example 5.A.36. Illustration of Algorithm 5.15.

According to Example 5.A.26 the input data are: the block ($\delta = \{1, 2, 3\}$), the schedule ($t_1 = 40$, $t_2 = 110$ and $t_2 = 180$) and the delay times ($E = \{40, 50, 90, 140\}$).

For each delay time it is checked if the penalization is reduced:

$\epsilon = 40$. The penalization change is initialized to $\sigma_d = 0$. Then, it is updated for each service:

$k = 1$. In this case, $t_1 + \epsilon = 40 + 40 = 80$ and $\underline{\beta}_1 = 90$, which means that $t_1 + \epsilon < \underline{\beta}_1$ (line 4). Therefore, the penalization is reduced if the service is delayed and $\sigma_d = \sigma_d - 1 = -1$ (line 5).

$k = 2$. In this case, $t_2 + \epsilon = 110 + 40 = 150$ and $\underline{\beta}_2 = 150$, which means that $t_2 + \epsilon = \underline{\beta}_2$ and the penalization remains unchanged.

$k = 3$. In this case, $t_3 + \epsilon = 180 + 40 = 220$ and $\underline{\beta}_3 = 270$, which means that $t_3 + \epsilon < \underline{\beta}_3$ (line 4). Therefore, the penalization is reduced if the service is delayed and $\sigma_d = \sigma_d - 1 = -2$ (line 5).

$\epsilon = 50$. The penalization change is initialized to $\sigma_d = 0$. Then, it is updated for each service:

$k = 1$. In this case, $t_1 + \epsilon = 40 + 50 = 90$ and $\underline{\beta}_1 = 90$, which means that $t_1 + \epsilon = \underline{\beta}_1$ and the penalization does not change.

$k = 2$. In this case, $t_2 + \epsilon = 110 + 50 = 160$ and $\bar{\beta}_2 = 270$, which means that $t_2 + \epsilon + \eta_2 k < \bar{\beta}_2$ and the penalization does not change.

$k = 3$. In this case, $t_3 + \epsilon = 180 + 50 = 230$ and $\underline{\beta}_3 = 270$, which means that $t_3 + \epsilon < \underline{\beta}_3$. Therefore, the penalization is reduced if the service is delayed and $\sigma_d = \sigma_d - 1 = -1$ (line 5).

$\epsilon = 90$. The penalization change is initialized to $\sigma_d = 0$. Then, it is updated for each service:

$k = 1$. In this case, $t_1 + \epsilon = 40 + 90 = 130$ and $\bar{\beta}_1 = 240$, which means that $t_1 + \epsilon + \eta_1 < \bar{\beta}_1$ and the penalization does not change.

$k = 2$. In this case, $t_2 + \epsilon = 110 + 90 = 200$ and $\bar{\beta}_2 = 270$, which means that $t_k + \epsilon + \eta_2 < \bar{\beta}_2$ and the penalization does not change.

$k = 3$. In this case, $t_2 + \epsilon = 180 + 90 = 270$ and $\underline{\beta}_2 = 270$, which means that $t_2 + \epsilon = \underline{\beta}_2$ and the penalization does not change.

Since $\sigma_d = 0$, the delay time is $\epsilon_d = \epsilon = 90$.

Example 5.A.37. Illustration of Algorithm 5.15.

According to Example 5.A.27 the input data are: the block ($\delta = \{4, 5, 6\}$), the schedule ($t_4 = 460$, $t_5 = 530$ and $t_6 = 600$) and the delay times ($E = \{60, 70, 80, 130\}$).

For each delay time it is checked if the penalization is reduced:

$\epsilon = 60$. The penalization change is initialized to $\sigma_d = 0$. Then, it is updated for each service:

$k = 4$. In this case, $t_4 + \epsilon = 460 + 60 = 520$ and $\underline{\beta}_4 = 540$, which means that $t_4 + \epsilon < \underline{\beta}_4$ (line 4). Therefore, the penalization is reduced if the service is delayed and $\sigma_d = \sigma_d - 1 = -1$ (line 5).

$k = 5$. In this case, $t_5 + \epsilon = 530 + 60 = 590$ and $\bar{\beta}_5 = 660$, which means that $t_5 + \epsilon + \eta_5 < \bar{\beta}_5$ and the penalization does not change.

$k = 6$. In this case, $t_6 + \epsilon = 600 + 60 = 660$ and $\bar{\beta}_6 = 870$, which means that $t_6 + \epsilon + \eta_6 < \bar{\beta}_6$ and the penalization does not change.

$\epsilon = 70$. The penalization change is initialized to $\sigma_d = 0$. Then, it is updated for each service:

- $k = 4$. In this case, $t_4 + \epsilon = 460 + 70 = 530$ and $\underline{\beta}_4 = 540$, which means that $t_4 + \epsilon < \underline{\beta}_4$ (line 4). Therefore, the penalization is reduced if the service is delayed and $\sigma_d = \sigma_d - 1 = -1$ (line 5).
- $k = 5$. In this case, $t_5 + \epsilon = 530 + 70 = 600$ and $\bar{\beta}_5 = 660$, which means that $t_5 + \epsilon + \eta_5 = \bar{\beta}_5$. Therefore, the penalization is reduced if the service is delayed and $\sigma_d = \sigma_d + 1 = 0$ (line 5).
- $k = 6$. In this case, $t_6 + \epsilon = 600 + 70 = 670$ and $\bar{\beta}_6 = 870$, which means that $t_6 + \epsilon + \eta_6 < \bar{\beta}_6$ and the penalization does not change.

Since $\sigma_d = 1$, the delay time is $\epsilon_d = \epsilon = 70$.

Example 5.A.38. Illustration of Algorithm 5.15.

According to Example 5.A.28 the input data are: the block ($\delta = \{5, 6\}$), the schedule ($t_5 = 530$ and $t_6 = 600$) and the delay times ($E = \{60, 70, 130\}$).

For each delay time it is checked if the penalization is reduced:

$\epsilon = 60$. The penalization change is initialized to $\sigma_d = 0$. Then, it is updated for each service:

- $k = 5$. In this case, $t_5 + \epsilon = 530 + 60 = 590$ and $\bar{\beta}_5 = 660$, which means that $t_5 + \epsilon + \eta_5 < \bar{\beta}_5$ and the penalization does not change.
- $k = 6$. In this case, $t_6 + \epsilon = 600 + 60 = 660$ and $\bar{\beta}_6 = 870$, which means that $t_6 + \epsilon + \eta_6 < \bar{\beta}_6$ and the penalization does not change.

Since $\sigma_d = 0$, the delay time is $\epsilon_d = \epsilon = 60$.

5.A.9 Modify the schedule of some services of the block: *moveServices*

The function presented in Algorithm 5.16 is used to improve the soft time window penalization of a block that cannot be moved as a whole.

The algorithm starts by fixing the earliest and latest starting times of the first and last services of the block (lines 1 - 2), to guarantee that those services will not be moved. Then, the earliest and latest starting times of the remaining services are updated (lines 3 - 6) according to the ones of the first and last services. After that, the block is divided into smaller blocks, according to their soft time windows (line 7). The earliest and latest starting times, considering the soft time windows, are obtained (lines 8 - 11). Finally, the new schedule of the block is computed (line 12).

The functions ***getBlocksStw*** and ***getSchedulePenalization*** have been previously described in 4.A.1 and 4.1.2.

Algorithm 5.16: moveServices - Move the free services to improve penalization

```
   Data: the block ($\delta = \{\delta_1, ..., \delta_r\}$), the schedule ($t$), the earliest starting times ($a^e$) and the
         latest starting times ($a^l$)
   //Get earliest and latest times
1  $\hat{t}^e \leftarrow t^e$, $\hat{t}^l \leftarrow t^l$
   //Fix first and last service, because the whole block cannot be moved
2  $\hat{t}^e_{\delta_1} \leftarrow t_{\delta_1}$, $\hat{t}^l_{\delta_1} \leftarrow t_{\delta_1}$, $\hat{t}^e_{\delta_r} \leftarrow t_{\delta_r}$, $\hat{t}^l_{\delta_r} \leftarrow t_{\delta_r}$
   //Update earliest and latest times of the other services
3  for $k \in \{\delta_2, ..., \delta_r\}$ do
4      $\hat{t}^e_k \leftarrow \max\{\hat{t}^e_k, \hat{t}^e_{k-1} + \eta_{k-1} + \theta_{k-1,k}\}$
5  for $k \in \{\delta_{r-1}, ..., \delta_1\}$ do
6      $\hat{t}^l_k \leftarrow \min\{\hat{t}^l_k, \hat{t}^l_{k+1} - \theta_{k,k+1} - \eta_k\}$
   //Divide the block into smaller blocks according to stw
7  $\hat{\Delta} \leftarrow$ getBlocksStw($\delta$)
   //Get earliest start for the services, according to soft time windows
8  for $j \in \delta$ do
9      $b^e_j \leftarrow \min\{\max\{\underline{\beta}_j, \hat{t}^e_j\}, \hat{t}^l_j\}$
   //Get latest start for the services, according to soft time windows
10 for $j \in \delta$ do
11     $b^l_j \leftarrow \max\{\min\{\bar{\beta}_j - \eta_j, \hat{t}^l_j\}, \hat{t}^e_j\}$
   //Get schedule for the services
12 $\bar{t} \leftarrow$ getSchedulePenalization($\hat{t}^e$, $\hat{t}^l$, $b^e$, $b^l$, $\hat{\Delta}$, $\delta$ )
13 return $\epsilon_{final}$
```

5.A.10 Combine the schedule of the blocks: *getCombinedSchedules*

Algorithm 5.17 is used to combine the schedules of the two blocks in such a way that the final schedule is feasible and it has a break between the blocks of, at least, π_{min}. The first step is to combine the schedules of the two blocks into one single schedule (lines 1 - 5). Then, it checks if the break between the blocks does not reach a duration of π_{min}, in which case the blocks must be moved in order for the break to have that duration (lines 6 - 31). This is done by advancing the first block, and delaying the second one, in such a way that the increment of soft time window penalization is minimum.

Algorithm 5.17: getCombinedSchedules - Combine the schedules of the blocks

```
Data: block 1($\delta_1$), block 2($\delta_2$), the schedule of block 1 ($\hat{t}$), the schedule of block 2 ($\tilde{t}$), the
      route ($R$), the earliest starting times ($a^e$) and the latest starting times ($a^l$)
//Combine the schedules of the blocks
for $j \in R$ do
    if $j \in \delta_1$ then
        $\bar{t}_j \leftarrow \hat{t}_j$
    else
        $\bar{t}_j \leftarrow \tilde{t}_j$
//Check if the break between blocks is greater or equal than $\pi_{min}$
$j_1 \leftarrow$ last service of $\delta_1$, $j_2 \leftarrow$ first service of $\delta_2$
if $\bar{t}_{j_2} - (\bar{t}_{j_1} + \eta_{j_1} + \theta_{j_1,j_2}) < \pi_{min}$ then
    //Get the change in penalization if $\delta_1$ is advanced or $\delta_2$ is delayed
    $\sigma_d \leftarrow$ delayPenalization($\delta_2$, $\bar{t}$)
    $\sigma_a \leftarrow$ advancePenalization($\delta_1$, $\bar{t}$)
    //Check if it is better to start advancing $\delta_1$ or delaying $\delta_2$
    if $\sigma_a < \sigma_b$ then
        $delay \leftarrow false$
    else
        $delay \leftarrow true$
    //Get break between the blocks
    $b \leftarrow \bar{t}_{j_2} - (\bar{t}_{j_1} + \eta_{j_1} + \theta_{j_1,j_2})$
    //Increase the break between the blocks
    while $b < \pi_{min}$ do
        if $delay = true$ then
            //Get delay to have a break of duration $\pi_{min}$
            $md \leftarrow \max\{\min\{\bar{t}_{j_1} + \eta_{j_1} + \theta_{j_1,j_2} + \pi_{min}, a^l_{j_2}\}, a^e_{j_2}\} - \bar{t}_{j_2}$
            if $md > 0$ then
                $\bar{t}$, $d \leftarrow$ delay($\delta_2$, $\bar{t}$, $a^l$, $a^e$, $\sigma_a$, $\sigma_d$, $md$)
                $b \leftarrow \bar{t}_{j_2} - (\bar{t}_{j_1} + \eta_{j_1} + \theta_{j_1,j_2})$
                $delay \leftarrow false$
            else
                $delay \leftarrow false$
        else
            //Get advance to have a break of duration $\pi_{min}$
            $ma \leftarrow \bar{t}_{j_1} - \max\{\min\{\bar{t}_{j_2} - \pi_{min} - \theta_{j_1,j_2} - \eta_{j_1}, a^l_{j_1}\}, a^e_{j_1}\}$
            if $ma > 0$ then
                $\bar{t}$, $a \leftarrow$ advance($\delta_1$, $\bar{t}$, $a^l$, $a^e$, $\sigma_a$, $\sigma_d$, $ma$)
                $b \leftarrow \bar{t}_{j_2} - (\bar{t}_{j_1} + \eta_{j_1} + \theta_{j_1,j_2})$
                $delay \leftarrow true$
            else
                $delay \leftarrow true$
return $\bar{t}$
```

Example 5.A.39. Illustration of Algorithm 5.17.

The input data, obtained in Example 5.1.3, are: the schedule of the first ($\hat{t}_1 = 90$ and $\hat{t}_2 = 160$) and second ($\tilde{t}_3 = 450$, $\tilde{t}_4 = 520$, $\tilde{t}_5 = 590$ and $\tilde{t}_6 = 660$) block.

In this case, the combined schedule is $\bar{t}_1 = 90$, $\bar{t}_2 = 160$, $\bar{t}_3 = 450$, $\bar{t}_4 = 520$, $\bar{t}_5 = 590$ and

$\bar{t}_6 = 660$. The break between the blocks is $\bar{t}_3 - (\bar{t}_2 + \eta_2 + \theta_{2,3}) = 450 - (160 + 60 + 10) = 220 \geq \pi_{min} = 120$ (line 7), which means that the schedule does not need to be modified.

Example 5.A.40. Illustration of Algorithm 5.17.

The input data, obtained in Example 5.1.3, are: the schedule of the first ($\hat{t}_1 = 130$, $\hat{t}_2 = 80$ and $\hat{t}_2 = 270$) and second ($\tilde{t}_4 = 530$, $\tilde{t}_5 = 600$ and $\tilde{t}_6 = 670$) block.

In this case, the combined schedule is $\bar{t}_1 = 130$, $\bar{t}_2 = 80$, $\bar{t}_2 = 270$, $\bar{t}_4 = 530$, $\bar{t}_5 = 600$ and $\bar{t}_6 = 670$. The break between the blocks is $\bar{t}_4 - (\bar{t}_3 + \eta_3 + \theta_{3,4}) = 530 - (270 + 60 + 10) = 190 \geq \pi_{min} = 120$ (line 7), which means that the schedule does not need to be modified.

Example 5.A.41. Illustration of Algorithm 5.17.

The input data, obtained in Example 5.1.3, are: the schedule of the first ($\hat{t}_1 = 180$, $\hat{t}_2 = 250$, $\hat{t}_3 = 320$ and $\hat{t}_4 = 390$) and second ($\tilde{t}_5 = 590$ and $\tilde{t}_6 = 660$) block.

In this case, the combined schedule is $\bar{t}_1 = 180$, $\bar{t}_2 = 250$, $\bar{t}_3 = 320$, $\bar{t}_4 = 390$, $\bar{t}_5 = 590$ and $\bar{t}_6 = 660$. The break between the blocks is $\bar{t}_5 - (\bar{t}_4 + \eta_4 + \theta_{4,5}) = 590 - (390 + 60 + 10) = 130 \geq \pi_{min} = 120$ (line 7), which means that the schedule does not need to be modified.

5.A.10.1 Penalization change of the block when it is delayed: *delayPenalization*

Algorithm 5.18 obtains the change in the penalization if second block is delayed. It starts initializing a counter (line 1) and then, for each service of the block, checks if its penalization would decrease or increase (lines 2 - 6). The penalization will be reduced if the service is scheduled before its soft time window (line 3), in which case the counter is reduced (line 4). The penalization increases if the service is scheduled at the end of its soft time window or after (line 5), so the counter increases (line 6).

Algorithm 5.18: delayPenalization - Get the change in penalization when delaying the block

```
Data: block 2 ($\delta_2$) and the schedule of block 2 ($\tilde{t}$)
//Check if the penalization of $\delta_2$ increases if it is delayed
1 $\sigma_d \leftarrow 0$
2 for $k \in \delta_2$ do
3     if $\tilde{t}_k < \underline{\beta}_k$ then
4         $\sigma_d \leftarrow \sigma_d - 1$ //The penalization is reduced
5     else if $\tilde{t}_k + \eta_k \geq \bar{\beta}_k$ then
6         $\sigma_d \leftarrow \sigma_d + 1$ //The penalization increases
7 return $\sigma_d$
```

5.A.10.2 Penalization change of the block when it is advanced: *advancePenalization*

Algorithm 5.19 computes how the penalization will change when advancing the first block. After initializing a counter (line 1), for each service the algorithm tests whether the penalization will increase or decrease if the block is advanced (lines 2 - 6). If the service ends after its soft time window (line 3) then its penalization will decrease and, therefore, the counter is reduced (line 4). If the service is scheduled at the start of its soft time window, or before it, the penalization will increment (line 5), so the counter increases (line 6).

Algorithm 5.19: advancePenalization - Get the change in penalization if the block is advanced

```
  Data: block 1 ($\delta_1$) and the schedule of block 2 ($\hat{t}$)
  //Check if the penalization of $\delta_1$ increases if it is advanced
1 $\sigma_a \leftarrow 0$
2 for $k \in \delta_1$ do
3     if $\hat{t}_k + \eta_k > \bar{\beta}_k$ then
4         $\sigma_a \leftarrow \sigma_a - 1$ //The penalization is reduced
5     else if $\hat{t}_k \leq \underline{\beta}_k$ then
6         $\sigma_a \leftarrow \sigma_a + 1$ //The penalization increases
7 return $\sigma_a$
```

5.A.10.3 Delay the block: *delay*

Algorithm 5.20 is used to delay the second block in order to increase the break with the first one.

Algorithm 5.20: delay - Delay the block

```
   Data: Block 2 ($\delta_2$), the schedule ($\bar{t}$), the earliest starting times ($a^e$), the latest starting times
         ($a^l$), the penalization change for Block 1 ($\sigma_a$), the penalization change for Block 2 ($\sigma_d$)
         and the maximum delay time ($md$)
   //Get possible delay times for the block
1  $E_d \leftarrow \emptyset$
2  for $k \in \delta_2$ do
       //Delay time so the service it leaves its stw
3      $d \leftarrow \max\{\bar{\beta}_k - (\bar{t}_k + \eta_k), 0\}$
4      if $d < md$ and $d \notin E_d$ then
5          $E_d \leftarrow E_d \cup \{d\}$
   //Add to the list the maximum possible delay time
6  if $md \notin E_d$ then
7      $E_d \leftarrow E_d \cup \{md\}$
   //Delay the block
8  for $\epsilon \in sorted(E_d)$ do
       //Get the change in penalization when delaying the block
9      $\bar{\sigma}_d \leftarrow 0$
10     for $j \in \delta_1$ do
11         if $\bar{t}_j + \epsilon < \underline{\beta}_j$ then
12             $\bar{\sigma}_d \leftarrow \bar{\sigma}_d - 1$
13         else if $\bar{t}_j + \epsilon + \eta_j \geq \bar{\beta}_j$ then
14             $\bar{\sigma}_d \leftarrow \bar{\sigma}_d + 1$
       //Delay the block and check if the algorithm should end
15     if $\bar{\sigma}_d >= \sigma_a$ then
16         $\bar{t}_k \leftarrow \bar{t}_k + \epsilon$, $\forall k \in \delta_2$, $\sigma_d \leftarrow \bar{\sigma}_d$, break
17     else if $\epsilon = md$ then
18         $\bar{t}_k \leftarrow \bar{t}_k + \epsilon$, $\forall k \in \delta_2$, $\sigma_d \leftarrow \bar{\sigma}_d$
19 return $\bar{t}$, $\sigma_d$
```

The algorithm starts obtaining the amount of time that the block needs to be delayed to make each service leave its soft time window (lines 1 - 7). To delay the block the algorithm iterates through the sorted list of times (line 8) and obtains how the penalization would change if the block is delayed (lines 9 - 14). That is, for each service, if the penalization would decrease then the counter is reduced (lines 11 - 12) and, if it would increment, the counter increases (lines 13 - 14). After that, the block is delayed, the change of penalization d is updated and the method checks if the loop can be continued (lines 15 - 18). This is done comparing the increase of penalization when delaying the second block, σ_d, with the one of the first block, σ_a. If the value for the first block is not the biggest one, the algorithm terminates, in any other case it continues.

5.A.10.4 Advance the block: *advance*

Algorithm 5.21 is the one used to advance the first block in order to increase the break with the second one. First, the advance time that makes each service of the block leave its soft time window is obtained (lines 1 - 7). Then, to advance the block, the algorithm iterates through the sorted list of times (line 8) and computes how the penalization would change if the block is advanced (lines 9 - 14). This means that, for each service, if the penalization would decrease then the counter is reduced (lines 11 - 12) and, if it would increment, the counter is increased (lines 13 - 14). After that, the block is advanced, the change of penalization s is updated and the method checks if the loop can be continued (lines 15 - 18). This is done by comparing the increase of penalization when advancing the first block, σ_a, with the one of the second block, σ_b. If the value for the second block is not the biggest one, the algorithm terminates, in any other case it continues.

Algorithm 5.21: advance - Advance the block

```
Data: Block 1 ($\delta_1$), the schedule ($\bar{t}$), the earliest starting times ($a^e$), the latest starting times
      ($a^l$), the penalization change for Block 1 ($\sigma_a$), the penalization change for Block 2 ($\sigma_d$)
      and the maximum advance time ($ma$)
   //Get possible advance times for the block
1  $E_a \leftarrow \emptyset$
2  for $k \in \delta_1$ do
       //Advance time so the service leaves its stw
3      $a \leftarrow \max\{\bar{t}_k - \underline{\beta}_k, 0\}$
4      if $a < ma$ and $a \notin E_a$ then
5          $E_a \leftarrow E_a \cup \{a\}$
6  if $ma \notin E_a$ then
7      $E_a \leftarrow E_a \cup \{ma\}$
   //Advance the block
8  for $\epsilon \in sorted(E_a)$ do
       //Change of penalization when advancing the block
9      $\bar{\sigma}_a \leftarrow 0$
10     for $j \in \delta_1$ do
11         if $\bar{t}_j - \epsilon > \bar{\beta}_j$ then
12             $\bar{\sigma}_a \leftarrow \bar{\sigma}_a - 1$
13         else if $\bar{t}_j - \epsilon \leq \underline{\beta}_j$ then
14             $\bar{\sigma}_a \leftarrow \bar{\sigma}_a + 1$
       //Advance the block and check if the algorithm should end
15     if $\bar{\sigma}_a >= \sigma_d$ then
16         $\bar{t}_k \leftarrow \bar{t}_k - \epsilon$, $\forall k \in \delta_1$, $\sigma_a \leftarrow \bar{\sigma}_a$, break
17     else if $\epsilon = ma$ then
18         $\bar{t}_k \leftarrow \bar{t}_k - \epsilon$, $\forall k \in \delta_1$, $\sigma_a \leftarrow \bar{\sigma}_a$
19 return $\bar{t}$, $\sigma_a$
```

Chapter 6

The biobjective problem

In previous chapters the HCSP was studied in a hierarchical way, prioritizing either the welfare of the users or the cost of the schedule. This chapter focuses on the biobjective version of the HCSP, that is, considering both objectives at the same time to analyze how they interact. Solving the biobjective problem results in multiple different solutions, giving the supervisor a diverse selection of schedules to choose from.

The MILP describing the biobjective problem is the one presented in Chapter 2.1, that is,

$$f_1 = \min\ \omega_1 \sum_{i\in N}\sum_{d\in D}\sum_{j\in S}\sum_{k\in S^1} \lambda_j^i x_{jk}^{id} + \omega_2 \sum_{j\in S}(v_j^{start} + v_j^{end}) \tag{2.1}$$

$$f_2 = \min\ \omega_3 \sum_{i\in N} z^i + \omega_4 \sum_{i\in N}\sum_{d\in D}(t_s^{id} - t_0^{id} - \hat{r}^{id}) \tag{2.2}$$

subject to: (2.3) - (2.29).

This chapter is structured as follows. Section 6.1 focuses on the epsilon constraint method, which is a technique that uses the MILP formulation to solve the biobjective problem. Section 6.2 consists in a detailed description of the metaheuristic algorithm proposed for solving the problem. Finally, Appendix 6.A presents additional information related to the metaheuristic algorithm.

6.1 Epsilon Constraint method

In this section one of the most well known techniques to solve multiobjective problems, the Epsilon Constraint approach (Haimes *et al.* (1971)), is introduced.

Let us consider the general formulation of a multiobjective problem, P, with p objectives:

$$\min\ f_1(\boldsymbol{x}), ..., f_p(\boldsymbol{x}) \tag{6.1}$$

$$\text{st. } \boldsymbol{x} \in F \tag{6.2}$$

Given two feasible solutions $\boldsymbol{x}$ and $\boldsymbol{y}$, it is said that $\boldsymbol{x}$ dominates $\boldsymbol{y}$ (denoted as $\boldsymbol{x} \succ \boldsymbol{y}$) if $f_k(\boldsymbol{x}) \leq f_k(\boldsymbol{y})\ \forall k \in \{1, ..., p\}$ and $f_k(\boldsymbol{x}) < f_k(\boldsymbol{y})$ for at least one $k \in \{1, ..., p\}$. When solving a multiobjective problem the goal is to find the Pareto frontier, which is the set composed by the non dominated solutions.

6.1.1 AUGMECON2 method

The exact Pareto frontier can be obtained using the improved version of the Augmented Epsilon Constraint method (known as AUGMECON2), presented by Mavrotas & Florios (2013). The Epsilon Constraint method, and consequently the AUGMECON2, uses the MILP to calculate the set of non dominated points.

In order for this method to determine the exact Pareto set, the problem must satisfy two conditions: the objective function coefficients have to be integer and the nadir points of the Pareto set must be known (which are composed by the worst objective values of the solutions in the Pareto set). The first condition can always be achieved by multiplying the coefficients of the objective functions by the appropriate power of 10. The second condition is met in the context of the problem under study because it has two objectives, which means that the nadir points can be obtained by calculating the pay-off table (for problems with more than 2 objectives the nadir points must be calculated using other methods proposed in the literature, see Deb *et al.* (2006) to estimate the nadir points using evolutionary approaches).

The pay-off table, PT, of the original problem P is defined as:

$$PT = \begin{pmatrix} f_1(\boldsymbol{x}_1) & \cdots & f_k(\boldsymbol{x}_{1,k}) & \cdots & f_p(\boldsymbol{x}_{1,p}) \\ \vdots & \ddots & & & \vdots \\ f_1(\boldsymbol{x}_{k,1}) & \cdots & f_k(\boldsymbol{x}_k) & \cdots & f_p(\boldsymbol{x}_{k,p}) \\ \vdots & & & \ddots & \vdots \\ f_1(\boldsymbol{x}_{p,1}) & \cdots & f_k(\boldsymbol{x}_{p,k}) & \cdots & f_p(\boldsymbol{x}_p) \end{pmatrix}, \tag{6.3}$$

where $f_k(\boldsymbol{x})$ is the value of the objective function k when evaluating a solution $\boldsymbol{x}$. Two solutions are considered: $\boldsymbol{x}_k$, which is the solution that optimizes objective k, and $\boldsymbol{x}_{k,k'}$, which is the solution of the lexicographic problem that considers k as the first objective and k' as the second one.

From this pay-off table, the range r_k (the difference between the maximum and minimum value that each objective k takes according to the table) and the upper bound ub_k are obtained, with $k = 2, ..., p$. Moreover, each r_k is divided into q_k equal intervals, which results in $q_k + 1$ grid points (g_k).

The AUGMECON2 method, presented in Algorithm 6.1, consists in iteratively solving problem $\hat{P}$, which is:

$$\begin{aligned} \min\ & f_1(\boldsymbol{x}) + \varepsilon(S_2/r_2 + 10^{-1}S_3/r_3 + \cdots + 10^{-(p-2)}S_p/r_p) \\ \text{st.}\ & f_k(\boldsymbol{x}) - S_k = e_k,\ i = 2, \ldots, p \\ & \boldsymbol{x} \in F \end{aligned}$$

where F represents the constraints of the problem, $S_k \in \mathbb{R}$ is the surplus variable and $e_k = ub_k - (i_k \times (r_k/g_k))$ is the right hand side of the new constraint for objective k, ε is a small number usually between 10^{-3} and 10^{-6} and F is the feasible region of the original problem P.

The AUGMECON2 algorithm starts by initializing the grid point counters (i_k) and, then, it iterates through them to solve problem $\hat{P}$. If the solution found is feasible, it is added to the non dominated set, Ω, and the bypass coefficient, $b = \lfloor S_2/(r_k/g_k) \rfloor$, is calculated. This coefficient indicates the number of consecutive iterations that can be skipped, setting $i_2 = i_2 + b$. If the solution of $\hat{P}$ is not feasible, the loop for objective 2 is restarted, $i_2 = 0$, and the method continues with the other loops. Finally, the algorithm stops after it has iterated through all the grid points, that is, after it reaches $i_p = g_p$.

Algorithm 6.1: AUGMECON2

```
Data: the epsilon parameter ($\varepsilon$), the range ($r_k$), the upper bound ($ub_k$) and the number of
      intervals ($g_k$)
$i_k \leftarrow 1\ \forall k = 2, ..., p$
while $i_p \leq g_p$ do
    $\hat{P} \leftarrow$ generateMILP($ub_k, r_k, g_k, i_2, ..., i_p, \varepsilon$)
    $\boldsymbol{x} \leftarrow$ solveMILP($\hat{P}$)
    if $\boldsymbol{x}$ is not feasible then
        $i_2 \leftarrow g_2$
    else
        $\Omega \leftarrow \Omega \cup \{\boldsymbol{x}\}$ //Add the solution to the non dominated set
        $b \leftarrow \lfloor S_2/(r_k/g_k) \rfloor$, $i_2 \leftarrow i_2 + b$ //Get the number of skippable iterations
    if $i_2 < g_2$ then
        $i_2 \leftarrow i_2 + 1$
    else
        $i_2 \leftarrow 0$
        ...
        if $i_{p-1} < g_{p-1}$ then
            $i_{p-1} \leftarrow i_{p-1} + 1$
        else
            $i_{p-1} \leftarrow 0$
            $i_p \leftarrow i_p + 1$
return $\Omega$
```

6.1.1.1 Biobjective version of AUGMECON2

In this section, the AUGMECON2 for the specific case of the biobjective HCSP is described.

Specifically, for the biojective version of the HCSP, the AUGMECON2 method iteratively solves the following problem, $\hat{P}$:

$$\begin{aligned} &\min\ [f_1(\boldsymbol{x}) + \varepsilon(S_2/r_2)] \\ &\text{st.}\ f_2(\boldsymbol{x}) - S_2 = e_2 \\ &\quad \boldsymbol{x} \in F, \end{aligned}$$

where f_1 is equation (2.1), f_2 is equation (2.2) and F represents Constraints (2.3) - (2.29) of the HCSP problem, S_2 is the surplus variable for f_2, ub_2 is the upper bound of f_2, r_2 is the range of f_2, g_2 indicates the number of intervals that divide the range, i_2 is the grid point counter and $e_2 = ub_2 - (i_2 \times (r_2/g_2))$ is the right hand side of the new constraint.

As it was previously explained, to obtain ub_2 and r_2, it is necessary to solve the two lexicographical versions of the problem. It results in two solutions: one that prioritizes the welfare over the cost ($\boldsymbol{x}_{wc}$) and another that prioritizes the cost over the welfare ($\boldsymbol{x}_{cw}$). As it can be seen in Figure 6.1, the upper bound is $ub_2 = f_2(\boldsymbol{x}_{wc})$ and the range is $r_2 = f_2(\boldsymbol{x}_{wc}) - f_2(\boldsymbol{x}_{cw})$. In the example of Figure 6.1 it is also illustrated how the values of e_2 are obtained after dividing the range in $g_2 = 3$ intervals.

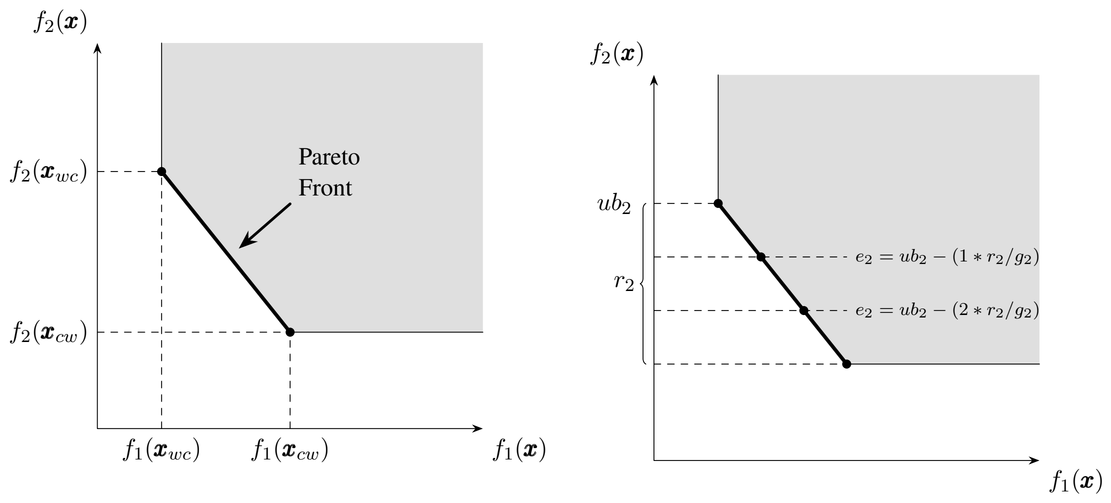

$f_2(\boldsymbol{x})$
$f_2(\boldsymbol{x}_{wc})$
Pareto Front
$f_2(\boldsymbol{x}_{cw})$
$f_1(\boldsymbol{x}_{wc})$
$f_1(\boldsymbol{x}_{cw})$
$f_1(\boldsymbol{x})$
$f_2(\boldsymbol{x})$
ub_2
r_2
$e_2 = ub_2 - (1 * r_2/g_2)$
$e_2 = ub_2 - (2 * r_2/g_2)$
$f_1(\boldsymbol{x})$

Figure 6.1: Example illustrating the AUGMECON2 method.

The biobjective AUGMECON2 method, presented in Algorithm 6.2, initializes the grid point counters, i_2, (line 1) and iterates through them to solve problem $\hat{P}$ (lines 3 - 4). If the solution found is feasible, then it is added to the non dominated set (lines 5 - 6). The bypass coefficient, $b = \lfloor S_2/(r_2/g_2) \rfloor$, indicates the number of consecutive iterations that can be skipped, $i_2 = i_2 + b$ (line 7). If the solution of $\hat{P}$ is not feasible (lines 8 - 9), or if all grid points have been evaluated (lines 10 - 11), the algorithm ends.

Algorithm 6.2: Biobjective AUGMECON2

```
   Data: the epsilon parameter ($\varepsilon$), the range ($r_2$), the upper bound ($ub_2$) and the number of
         intervals ($g_2$)
 1 $i_2 \leftarrow 1$
 2 while $i_2 \leq g_2$ do
 3     $\hat{P} \leftarrow$ generateMILP($ub_2, r_2, g_2, i_2, \varepsilon$) //Generate the problem
 4     $\boldsymbol{x} \leftarrow$ solveMILP($\hat{P}$) //Obtain a solution to the problem
 5     if $\boldsymbol{x}$ is feasible then
 6         $\Omega \leftarrow \Omega \cup \{\boldsymbol{x}\}$ //Add the solution to the non dominated set
 7         $b \leftarrow \lfloor S_2/(r_2/g_2) \rfloor$, $i_2 \leftarrow i_2 + b$ //Get the number of skippable iterations
 8     else
 9         break //The solution is not feasible
10     if $i_2 < g_2$ then
11         $i_2 \leftarrow i_2 + 1$ //Next grid point
12 return $\Omega$
```

6.2 Biobjective metaheuristic algorithm

This section is focused on the algorithm (from now on denoted as BIALNS) designed to solve the biobjective version of the HCSP, that is, to obtain non dominated solutions. A solution of the problem is $\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$ and it is composed by the routes ($\boldsymbol{x}$) and the schedules ($\boldsymbol{t}$). The routes describe the order of services to be performed by each caregiver and the schedules are the starting times of each service. The goal of the algorithm is to obtain the set of non dominated solutions (Ω), so it relies on a set of solutions that are composed by different routes ($\hat{\Omega}$).

Figure 6.2 presents a simple diagram to explain the general scheme of the algorithm designed to solve the biobjective HCSP under study. The algorithm is divided into three steps: in the first one the hierarchical versions of the problem are solved, in the second step solutions composed by different routes are generated and, in the final step, non dominated solutions are obtained.

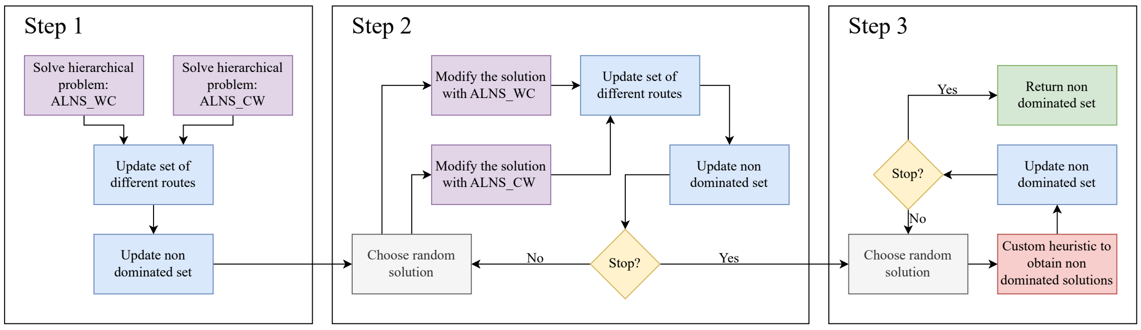

Step 1
Solve hierarchical problem: ALNS_WC
Solve hierarchical problem: ALNS_CW
Update set of different routes
Update non dominated set
Step 2
Modify the solution with ALNS_WC
Update set of different routes
Modify the solution with ALNS_CW
Update non dominated set
Choose random solution
No
Stop?
Yes
Step 3
Yes
Return non dominated set
Stop?
Update non dominated set
No
Choose random solution
Custom heuristic to obtain non dominated solutions

Figure 6.2: Scheme of algorithm BIALNS.

The pseudocode presented in Algorithm 6.3 is the one used to obtain non dominated solutions for the problem, that is, solutions that cannot improve one objective without deteriorating the other one. The three functions involved in the algorithm correspond to the three steps outlined in the diagram.

Algorithm 6.3: BIALNS - Metaheuristic algorithm to obtain non dominated solutions

```
  Data: the set of services ($S$) and the set of caregivers ($N$)
  //Get the set of initial solutions
1 $\Omega, \hat{\Omega} \leftarrow$ initialiseSets($S$, $N$)
  //Get solutions composed by different routes
2 $\Omega, \hat{\Omega}_1, \hat{\Omega}_2 \leftarrow$ getDifferentSolutions($S$, $\Omega$, $\hat{\Omega}$)
  //Get non dominated solutions
3 $\Omega \leftarrow$ getNonDominatedSet($S$, $\Omega$, $\hat{\Omega}_1$, $\hat{\Omega}_2$)
4 return $\Omega$
```

Now the three steps involved in the algorithm (***initialiseSets***, ***getDifferentSolutions*** and ***getNonDominatedSet***) will be thoroughly explained.

6.2.1 Initialise the sets: *initialiseSets*

The non dominated set is initialized using Algorithm 6.4, which is based on the methodology explained in Chapter 3, Section 3.1. First, initial solutions of each lexicographical objective are generated (following the procedure described in Section 3.2), improving them using the ALNS method (lines 1 - 4). After that, the set of non dominated solutions is updated (lines 5 - 7).

Algorithm 6.4: initialiseSets - Get solutions for each lexicographic objective

```
Data: the set of services ($S$), the set of caregivers ($N$), the set of solutions ($\hat{\Omega} = \emptyset$) and the
      set of non dominated solutions ($\Omega = \emptyset$)
  //Get initial solution for the lexicographic welfare-cost
1 $\boldsymbol{\omega}_{wc} \leftarrow$ initialSolution($S$, $N$, $f_{wc}$)
  //ALNS for the lexicographic welfare-cost
2 $\boldsymbol{\omega}_{wc}, \hat{\Omega} \leftarrow$ ALNS($f_{wc}$, $\boldsymbol{\omega}_{wc}$, $\hat{\Omega}$)
  //Get initial solution for the lexicographic cost-welfare
3 $\boldsymbol{\omega}_{cw} \leftarrow$ initialSolution($S$, $N$, $f_{cw}$)
  //ALNS for the lexicographic cost-welfare
4 $\boldsymbol{\omega}_{cw}, \hat{\Omega} \leftarrow$ ALNS($f_{cw}$, $\boldsymbol{\omega}_{cw}$, $\hat{\Omega}$)
  //Get non dominated solutions
5 $\Omega \leftarrow$ updateNonDominatedSet($\Omega$, $\boldsymbol{\omega}_{wc}$)
6 $\Omega \leftarrow$ updateNonDominatedSet($\Omega$, $\boldsymbol{\omega}_{cw}$)
7 $\Omega \leftarrow$ updateNonDominatedSet($\Omega$, $\hat{\boldsymbol{\omega}}$) $\forall \hat{\boldsymbol{\omega}} \in \hat{\Omega}$
8 return $\Omega, \hat{\Omega}$
```

6.2.2 Generate solutions composed by different routes: *getDifferentSolutions*

The function used to generate solutions composed by different routes is described in Algorithm 6.5. The first step is to filter the set of solutions, that is, to remove the solutions that are too far from the non dominated ones (line 1). Then, until a stopping criteria is not met, the solutions are modified to obtain different routes. To this aim, a solution is selected at random (line 3) and it is updated with the ALNS for each lexicographic objective (lines 4 and 6). Then, two new solutions are generated, by updating the schedules of the ones obtained before in order to optimize the other lexicographic objective function (lines 5 and 7). Finally, these new solutions are used to update the non dominated set and the ones composed by different routes (lines 8 - 9).

Algorithm 6.5: getDifferentSolutions - Get set of solutions with different routes

```
Data: the set of services ($S$), the set of non dominated solutions ($\Omega$) and the set of solutions
      ($\hat{\Omega}$)
   //Remove the solutions that are too far from $\Omega$
1  $\hat{\Omega}_1, \hat{\Omega}_2 \leftarrow$ filterSet($\hat{\Omega}$, $\Omega$)
2  while stopping criteria is not met do
       //Get solution to modify
3      $\bar{\boldsymbol{\omega}} \leftarrow$ chooseRandomSolution($\hat{\Omega}_1$, $\hat{\Omega}_2$, $\Omega$)
       //Get solutions that improve each lexicographic objective
4      $\boldsymbol{\omega}_{wc} \leftarrow$ ALNS($S$, $N$, $f_{wc}$, $\bar{\boldsymbol{\omega}}$) //ALNS for the lexicographic welfare-cost
5      $\tilde{\boldsymbol{\omega}}_{cw} \leftarrow$ schedule($\boldsymbol{\omega}_{wc}, f_{cw}$) //Get schedule that optimizes $f_{cw}$
6      $\boldsymbol{\omega}_{cw} \leftarrow$ ALNS($S$, $N$, $f_{cw}$, $\bar{\boldsymbol{\omega}}$) //ALNS for the lexicographic cost-welfare
7      $\tilde{\boldsymbol{\omega}}_{wc} \leftarrow$ schedule($\omega_{cw}, f_{wc}$) //Get schedule that optimizes $f_{wc}$
       //Update the non dominated solutions
8      $\Omega \leftarrow$ updateNonDominatedSet($\Omega$, $\boldsymbol{\omega}$) $\forall \boldsymbol{\omega} \in \{\boldsymbol{\omega}_{wc}, \boldsymbol{\omega}_{cw}, \tilde{\boldsymbol{\omega}}_{cw}, \tilde{\boldsymbol{\omega}}_{wc}\}$
       //Update the sets of solutions
9      $\hat{\Omega}_1, \hat{\Omega}_2 \leftarrow$ updateSetsOfSolutions($\hat{\Omega}_1$, $\hat{\Omega}_2$, $\Omega$, $\omega$) $\forall \boldsymbol{\omega} \in \{\boldsymbol{\omega}_{wc}, \boldsymbol{\omega}_{cw}, \tilde{\boldsymbol{\omega}}_{cw}, \tilde{\boldsymbol{\omega}}_{wc}\}$
10 return $\Omega, \hat{\Omega}_1, \hat{\Omega}_2$
```

6.2.3 Generate non dominated solutions: *getNonDominatedSet*

The function shown in Algorithm 6.6 is devoted to generate non dominated solutions. The method, until a stopping criteria is met (line 1), chooses a solution, route and service at random, following a uniform distribution, (lines 2 - 4) and modifies its schedule to obtain non dominated solutions (lines 4 - 25). To this aim, the service is delayed and/or advanced in order to improve the soft time window penalization (lines 7 - 16) or the cost (lines 17 - 25) of the schedule.

Algorithm 6.6: getNonDominatedSet - Get set of non dominated solutions

```
   Data: the set of services ($S$), the set of non dominated solutions ($\Omega$) and the sets of solutions
         ($\hat{\Omega}_1, \hat{\Omega}_2$)
1  while stopping criteria not met do
       //Get service to move
2      $\bar{\omega} \leftarrow$ chooseRandomSolution($\hat{\Omega}$, $\hat{\Omega}_1$, $\hat{\Omega}_2$) //Get solution to modify
3      $R \leftarrow$ chooseRandomRoute($\bar{\omega}$) //Get route to modify
4      $j \leftarrow$ chooseRandomService($R$) //Get service to move
5      $t^e, t^l \leftarrow$ getInfo($R$) //Get earliest and latest times for the services
       //Get delay and advance times to improve penalization
6      $d, a \leftarrow$ getTimeStw($j$, $R$, $\bar{\omega}$, $t^e$, $t^l$)
       //Delay the service to improve penalization
7      $\hat{R} \leftarrow$ getAffectedServicesDelay($j$, $R$, $\bar{\omega}$, $d$, $F(j,R)$) //Get affected services
8      $\bar{d} \leftarrow$ updateDelay($j$, $R$, $\bar{\omega}$, $d$, $\hat{R}$) //Update maximum delay time
9      $\tilde{\omega} \leftarrow$ randomDelay($j$, $R$, $\bar{\omega}$, 0, $\bar{d}$, $d$, $\hat{R}$) //Delay at random the service
10     $\Omega \leftarrow$ updateNonDominatedSet($\Omega$, $\tilde{\omega}$) //Update the non dominated solutions
11     $\hat{\Omega}_1, \hat{\Omega}_2 \leftarrow$ updateSetsOfSolutions($\hat{\Omega}_1$, $\hat{\Omega}_2$, $\Omega$, $\tilde{\omega}$) //Update the sets of solutions
       //Advance the service to improve penalization
12     $\hat{R} \leftarrow$ getAffectedServicesAdvance($j$, $R$, $\bar{\omega}$, $a$, $P(j,R)$) //Get affected services
13     $\bar{a} \leftarrow$ updateAdvance($j$, $R$, $\bar{\omega}$, $a$, $\hat{R}$) //Update maximum advance time
14     $\tilde{\omega} \leftarrow$ randomAdvance($j$, $R$, $\bar{\omega}$, 0, $\bar{a}$, $a$, $\hat{R}$) //Advance at random the service
15     $\Omega \leftarrow$ updateNonDominatedSet($\Omega$, $\tilde{\omega}$) //Update the non dominated solutions
16     $\hat{\Omega}_1, \hat{\Omega}_2 \leftarrow$ updateSetsOfSolutions($\hat{\Omega}_1$, $\hat{\Omega}_2$, $\Omega$, $\tilde{\omega}$) //Update the sets of solutions
       //Get delay and advance time to improve cost
17     $D, A \leftarrow$ getTimeCost($j$, $R$, $\bar{\omega}$, $t^e$, $t^l$)
18     for $(\underline{d}, \bar{d}) \in D$ do
           //Delay the service
19         $\tilde{\omega} \leftarrow$ randomDelay($j$, $R$, $\bar{\omega}$, $\underline{d}$, $\bar{d}$, $d$, $F(j,R)$) //Delay at random the service
20         $\Omega \leftarrow$ updateNonDominatedSet($\Omega$, $\tilde{\omega}$) //Update the non dominated solutions
21         $\hat{\Omega}_1, \hat{\Omega}_2 \leftarrow$ updateSetsOfSolutions($\hat{\Omega}_1$, $\hat{\Omega}_2$, $\Omega$, $\tilde{\omega}$) //Update the sets of solutions
22     for $(\underline{a}, \bar{a}) \in A$ do
           //Advance the service
23         $\tilde{\omega} \leftarrow$ randomAdvance($j$, $R$, $\bar{\omega}$, $\underline{a}$, $\bar{a}$, $a$, $P(j,R)$) //Advance at random the service
24         $\Omega \leftarrow$ updateNonDominatedSet($\Omega$, $\tilde{\omega}$) //Update the non dominated solutions
25         $\hat{\Omega}_1, \hat{\Omega}_2 \leftarrow$ updateSetsOfSolutions($\hat{\Omega}_1$, $\hat{\Omega}_2$, $\Omega$, $\tilde{\omega}$) //Update the sets of solutions
26 return $\Omega$
```

Figure 6.3 presents a simple diagram to explain the scheme of Algorithm 6.6. After choosing a route and a service at random (lines 2 - 4) the method modifies its schedule in six possible ways:

a. Delay the service to improve the penalization (lines 7 - 9).

b. Advance the service to improve the penalization (lines 12 - 14).

c. Delay the service to reduce the duration of the breaks that occur after it (lines 17 and 19).

d. Advance the service to reduce the duration of the breaks that occur before it (lines 17 and 23).

e. Delay the service to increase the duration of the break that occurs before it, with the aim of making this break reach a duration of at least π_{min} (lines 17 and 19).

f. Advance the service to increase the duration of the break that occurs after it, with the aim of making this break reach a duration of at least π_{min} (lines 17 and 23).

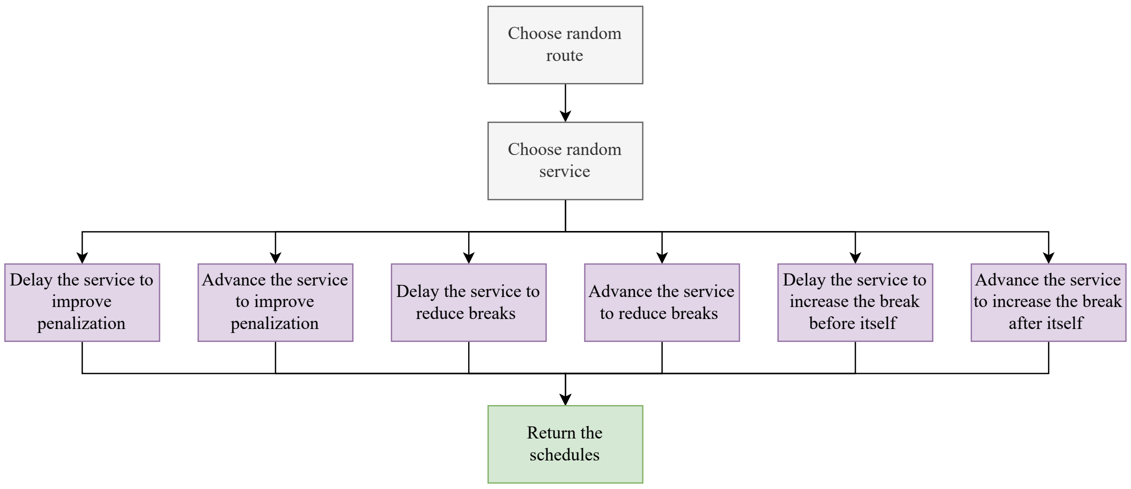

Choose random route
Choose random service
Delay the service to improve penalization
Advance the service to improve penalization
Delay the service to reduce breaks
Advance the service to reduce breaks
Delay the service to increase the break before itself
Advance the service to increase the break after itself
Return the schedules

Figure 6.3: Scheme of the method to obtain non dominated solutions.

Example 6.2.1. Illustration of Algorithm 6.6.

To illustrate Algorithm 6.6, the schedule of a route composed of 6 services will be modified, performing one iteration of the while loop. The schedule and time windows of the route are shown in Figure 6.4 and Table 6.1. For simplicity, all services have a duration of 1 hour, the travel time is ignored and 8:00 is time 0 of the planning horizon. The available working times for the caregiver are $\underline{\gamma} = 0$ and $\bar{\gamma} = 840$.

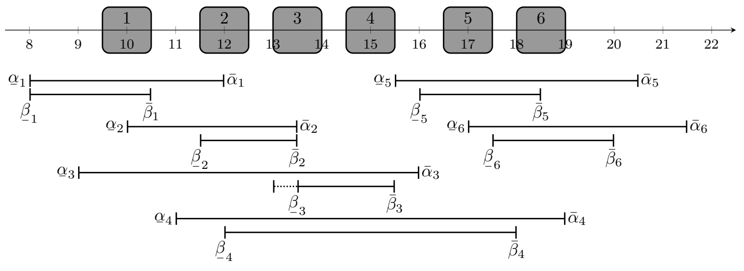


Figure 6.4: Schedule to modify.

j	1	2	3	4	5	6
$\underline{\alpha}_j$	0	120	60	180	450	540
$\bar{\alpha}_j$	240	330	480	660	750	810
$\underline{\beta}_j$	0	210	330	240	480	570
$\bar{\beta}_j$	150	330	450	600	630	720

Table 6.1: Hard and soft time windows of the services (BIALNS).

The current schedule of the services is: $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$. Since there is no break larger than two hours, the cost of the schedule is $f_2 = t_6 + \eta_6 - t_1 = 600 + 60 - 90 = 570$. The soft time window penalization is $f_1 = t_3 + \eta_3 - \bar{\beta}_3 = 300 + 60 - 330 = 30$.

Let us now suppose that Service 4 is the randomly selected service to obtain non dominated solutions. The earliest and latest starting times of the services are obtained with function ***getInfo*** (see details in Example 6.A.1): $t_1^e = 0$, $t_2^e = 120$, $t_3^e = 180$, $t_4^e = 240$, $t_5^e = 450$, $t_6^e = 540$, $t_1^l = 180$, $t_2^l = 270$, $t_3^l = 450$, $t_4^l = 600$, $t_5^l = 720$ and $t_6^l = 780$.

Then, the schedule is modified according to each objective:

Welfare. The penalization is improved applying the function ***getTimeStw*** (line 7), which gets the maximum possible advance and delay times for the services. As a result of applying the function, the maximum delay time is $d = 150$ and the maximum advance time is $a = 150$ (for more information see Example 6.A.2).

Delay. To delay the service, first the list of services that will be affected by the maximum delay is computed, with the function ***getAffectedServicesDelay*** (see Example 6.A.3 for more information), $\hat{R} = \{5, 6\}$ (line 8). Then the function ***updateDelay*** updates the delay time so it guarantees that the penalization of the schedule will not increase, $\bar{d} = 120$ (see details in Example 6.A.5). The interval within which Service 4 can be delayed is presented in Figure 6.5 as a bolded segment in the hard time window of the service.

Finally, the schedule of Service $j = 4$ is delayed 90 minutes with ***randomDelay*** (for more details see Example 6.A.15), resulting in the schedule presented in Figure 6.5: $\tilde{t}_1 = 90$, $\tilde{t}_2 = 210$, $\tilde{t}_3 = 300$, $\tilde{t}_4 = 480$, $\tilde{t}_5 = 540$ and $\tilde{t}_6 = 600$ (line 10).

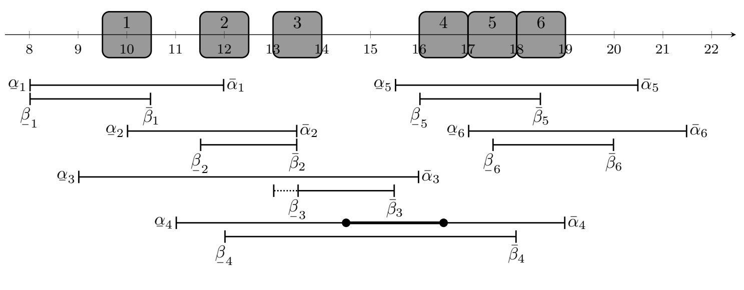


Figure 6.5: Delay Service 4.

The largest break of the schedule is between services 3 and 4, which has a duration of $b = \tilde{t}_4 - (\tilde{t}_3 + \eta_3 + \theta_{3,4}) = 480 - (300 + 60) = 120$. Therefore, the cost of the schedule is $f_2 = \tilde{t}_6 + \eta_6 - \tilde{t}_1 - b = 600 + 60 - 90 - 120 = 450$. The soft time window penalization is $f_1 = \underline{\beta}_3 - \tilde{t}_3 = 330 - 300 = 30$.

Advance. To advance the service, first the list of services that will be affected by the maximum advance is obtained, ***getAffectedServicesAdvance*** (see Example 6.A.7 for more information), $\hat{R} = \{3, 2, 1\}$ (line 12). Function ***updateAdvance*** is used to update the advance time so it guarantees that the penalization of the schedule will not increase, $\bar{a} = 0$, (see more details in Example 6.A.8). The interval within which Service 4 can be advanced is presented in Figure 6.6 as a bolded segment in the hard time window of the service.

Finally, the schedule of Service $j = 4$ is advanced 90 minutes with ***randomAdvance*** (for more information see Example 6.A.18), resulting in the schedule presented in Figure 6.6: $\tilde{t}_1 = 90$, $\tilde{t}_2 = 180$, $\tilde{t}_3 = 240$, $\tilde{t}_4 = 300$, $\tilde{t}_5 = 510$ and $\tilde{t}_6 = 600$ (line 15).

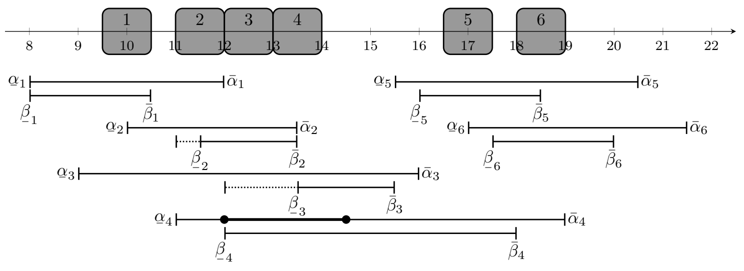


Figure 6.6: Advance Service 4.

The largest break of the schedule is between services 4 and 5, which has a duration of $b = t_5 - (\tilde{t}_4 + \eta_4 + \theta_{4,5}) = 510 - (300 + 60) = 150$. Therefore, the cost of the schedule is $f_2 = \tilde{t}_6 + \eta_6 - \tilde{t}_1 - b = 600 + 60 - 90 - 150 = 420$. The soft time window penalization is $f_1 = (\underline{\beta}_2 - \tilde{t}_2) + (\underline{\beta}_3 - \tilde{t}_3) = (210 - 180) + (330 - 240) = 120$.

Cost. To improve the cost of the schedule, the maximum possible advance and delay times for the service is found using function ***getTimeCost*** (for more details see Example 6.A.10). Therefore, two delay times and two advance times are possible:

Delay: $\underline{d} = 0$ and $\bar{d} = 90$. The schedule of Service $j = 4$ is delayed 60 minutes with ***randomDelay*** (for more information see Example 6.A.16), resulting in the schedule: $\tilde{t}_1 = 90$, $\tilde{t}_2 = 210$, $\tilde{t}_3 = 300$, $\tilde{t}_4 = 450$, $\tilde{t}_5 = 510$ and $\tilde{t}_6 = 600$ (line 18), Figure 6.7. The interval within which Service 4 can be delayed is presented in Figure 6.7 as a bolded segment in the hard time window of the service.

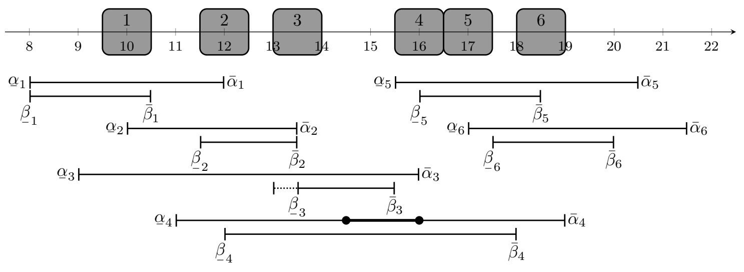


Figure 6.7: Delay Service 4 to reduce breaks.

There is no break larger that two hours, therefore the cost of the schedule is $f_2 = \tilde{t}_6 + \eta_6 - \tilde{t}_1 = 600 + 60 - 90 = 570$. The soft time window penalization is $f_1 = \underline{\beta}_3 - \tilde{t}_3 = 330 - 300 = 30$.

Delay: $\underline{d} = 90$ and $\bar{d} = 210$. The schedule of servService $j = 4$ is delayed 180 minutes with ***randomDelay*** (for more details see Example 6.A.17), resulting in the schedule shown in Figure 6.8: $\tilde{t}_1 = 90$, $\tilde{t}_2 = 210$, $\tilde{t}_3 = 300$, $\tilde{t}_4 = 570$ and $\tilde{t}_5 = 630$, $\tilde{t}_6 = 690$ (line 18).

The interval within which Service 4 can be delayed is presented in Figure 6.8 as a bolded segment in the hard time window of the service. Note that, this interval is obtained considering that the break between services 3 and 4 should have a duration of at least π_{min}, which is represented in the schedule as a bolded segment after the end of Service 3.

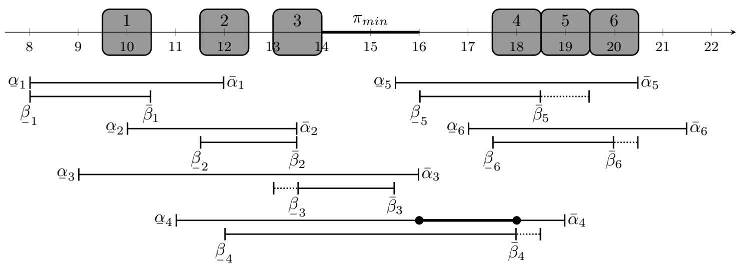


Figure 6.8: Delay Service 4 to increase the previous break.

The largest break of the schedule is between services 3 and 4, which has a duration of $b = t_4 - (\tilde{t}_3 + \eta_3 + \theta_{3,4}) = 570 - (300 + 60) = 210$. Therefore, the cost of the schedule is $f_2 = \tilde{t}_6 + \eta_6 - \tilde{t}_1 - b = 690 + 60 - 90 - 210 = 450$. The soft time window penalization is $f_1 = (\bar{\beta}_3 - \tilde{t}_3) + (\tilde{t}_4 + \eta_4 - \bar{\beta}_4) + (\tilde{t}_5 + \eta_5 - \bar{\beta}_5) + (\tilde{t}_6 + \eta_6 - \bar{\beta}_6) = (330 - 300) + (570 + 60 - 600) + (630 + 60 - 630) + (690 + 60 - 720) = 150$.

Advance: $\underline{a} = 0$ and $\bar{a} = 120$. The schedule of Service $j = 4$ is advanced 30 minutes with ***randomAdvance*** (see Example 6.A.19 for more information), resulting in the schedule shown in Figure 6.9: $\tilde{t}_1 = 90$, $\tilde{t}_2 = 210$, $\tilde{t}_3 = 300$, $\tilde{t}_4 = 360$, $\tilde{t}_5 = 510$ and $\tilde{t}_6 = 600$ (line 21). The interval within which Service 4 can be advanced is presented in Figure 6.9 as a bolded segment in the hard time window of the service.

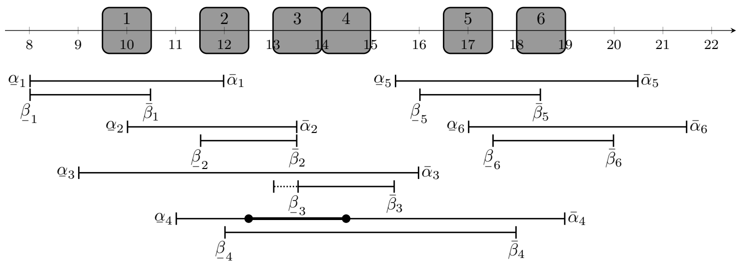


Figure 6.9: Advance Service 4 to reduce breaks.

There is no break larger that two hours, therefore the cost of the schedule is $f_2 = \tilde{t}_6 + \eta_6 - \tilde{t}_1 = 600 + 60 - 90 = 570$. The soft time window penalization is $f_1 = \underline{\beta}_3 - \tilde{t}_3 = 330 - 300 = 30$.

Advance: $\underline{a} = 60$ and $\bar{a} = 150$. The schedule of Service $j = 4$ is advanced 120 minutes with ***randomAdvance*** (for more information see Example 6.A.20), resulting in the schedule shown in Figure 6.10: $\tilde{t}_1 = 90$, $\tilde{t}_2 = 150$, $\tilde{t}_3 = 210$, $\tilde{t}_4 = 270$, $\tilde{t}_5 = 510$ and $\tilde{t}_6 = 600$ (line 21). The interval within which Service 4 can be advanced is presented in Figure 6.10 as a bolded segment in the hard time window of the service. Note that, this interval is obtained considering that the break between services 4 and 5 should have a duration of at least π_{min}, which is represented in the schedule as a bolded segment before the start of Service 5.

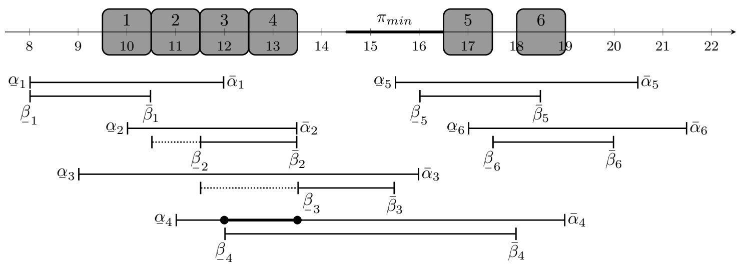

1
2
3
4
5
6
π_{min}
8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
α_1 $\bar{\alpha}_1$ $\underline{\beta}_1$ $\bar{\beta}_1$
α_2 $\bar{\alpha}_2$ $\underline{\beta}_2$ $\bar{\beta}_2$
α_3 $\bar{\alpha}_3$ $\underline{\beta}_3$ $\bar{\beta}_3$
α_4 $\bar{\alpha}_4$ $\underline{\beta}_4$ $\bar{\beta}_4$
α_5 $\bar{\alpha}_5$ $\underline{\beta}_5$ $\bar{\beta}_5$
α_6 $\bar{\alpha}_6$ $\underline{\beta}_6$ $\bar{\beta}_6$

Figure 6.10: Advance Service 4 to increase the next break.

The largest break of the schedule is between services 4 and 5, which has a duration of $b = \tilde{t}_5 - (\tilde{t}_4 + \eta_4 + \theta_{4,5}) = 510 - (270 + 60) = 180$. Therefore, the cost of the schedule is $f_2 = \tilde{t}_6 + \eta_6 - \tilde{t}_1 - b = 600 + 60 - 90 - 180 = 390$. The soft time window penalization is $f_1 = (\bar{\beta}_2 - \tilde{t}_2) + (\bar{\beta}_3 - \tilde{t}_3) = (210 - 150) + (330 - 210) = 180$.

The non dominated solutions are the ones presented in Figure 6.5 (B: $f_1 = 30$, $f_2 = 450$), Figure 6.6 (C: $f_1 = 120$, $f_2 = 420$) and Figure 6.10 (G: $f_1 = 180$, $f_2 = 390$). The dominated solutions are the ones shown in Figure 6.4 (A: $f_1 = 30$, $f_2 = 570$), Figure 6.7 (D: $f_1 = 30$, $f_2 = 570$), Figure 6.8 (E: $f_1 = 150$, $f_2 = 450$) and Figure 6.9 (F: $f_1 = 30$, $f_2 = 570$), because the schedule of Figure 6.5 is better, or equal, in terms of both objectives. Figure 6.11 shows the dominated (in red) and non dominated points (in blue) obtained in this example.

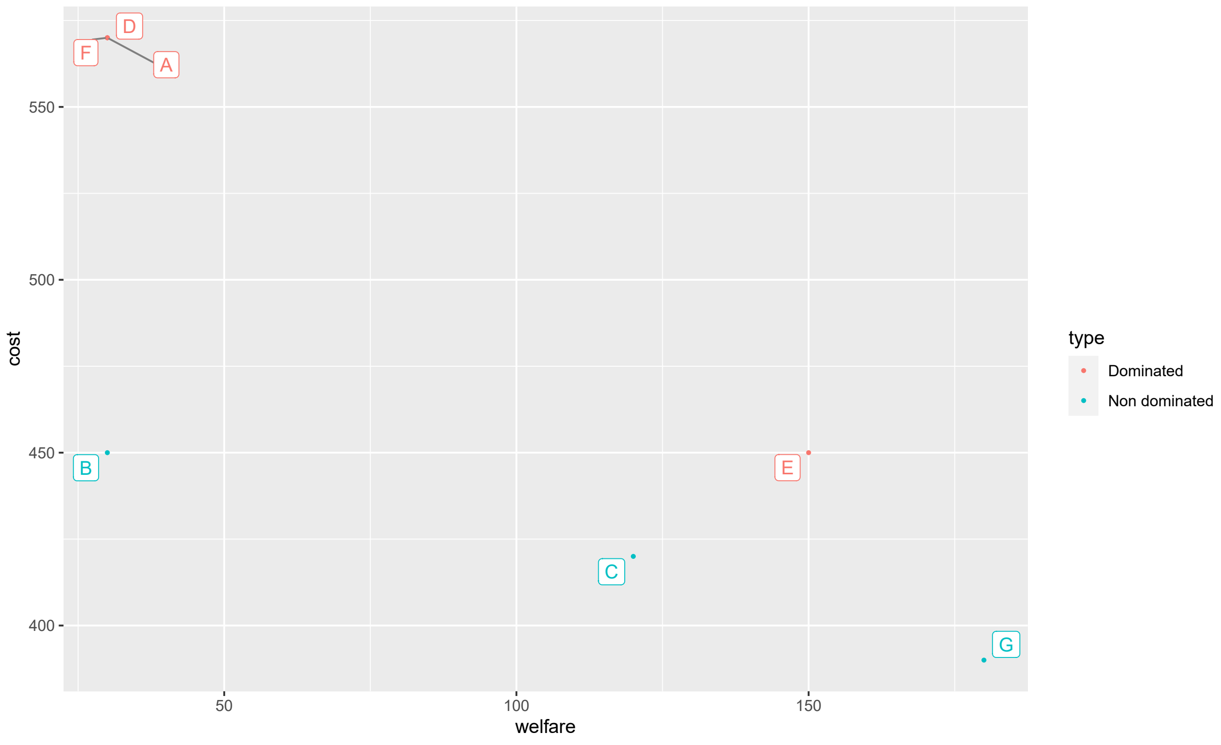


Figure 6.11: Dominated and non dominated points.

Appendix 6.A Auxiliary functions

The functions described in this appendix are used to completely describe the method used to obtain the Pareto frontier.

6.A.1 Adaptive Large Neighborhood Search: *ALNS*

Algorithm 6.7 presents the ALNS version used to improve solutions. This method is the same than the one described in Section 3.1, but now it also stores the solutions that are composed by different routes (line 9).

Algorithm 6.7: ALNS - Adaptive Large Neighborhood search

```
Data: the objective function ($f$), the initial solution ($\boldsymbol{\omega}$), the removal operators ($\Sigma_{rem}$), the
      insertion operators ($\Sigma_{ins}$) and the the set of solutions ($\hat{\Omega}$)
1  $\sigma_{rem} \leftarrow (1, ..., 1)$, $\sigma_{ins} \leftarrow (1, ..., 1)$, $\boldsymbol{\omega}' \leftarrow \boldsymbol{\omega}$
   //Improve the solution
2  while stopping criteria not met do
       //Get removal and insertion operators
3      $\varsigma_{rem} \leftarrow$ chooseRandom($\sigma_{rem}, \Sigma_{rem}$)
4      $\varsigma_{ins} \leftarrow$ chooseRandom($\sigma_{ins}, \Sigma_{ins}$)
       //Obtain new solution
5      $\bar{\boldsymbol{\omega}} \leftarrow$ destroySolution($\boldsymbol{\omega}$, $\varsigma_{rem}$, $f$)
6      $\boldsymbol{\omega}^* \leftarrow$ repairSolution($\bar{\boldsymbol{\omega}}$, $\varsigma_{ins}$, $f$)
       //Update best solution
7      if $f(\boldsymbol{\omega}^*) < f(\boldsymbol{\omega}')$ then
8          $\boldsymbol{\omega}' \leftarrow \boldsymbol{\omega}^*$
       //Update the set of multiple routes
9      $\hat{\Omega} \leftarrow$ updateMultipleRoutes($\hat{\Omega}$, $\boldsymbol{\omega}^*$)
       //Update current solution
10     $\boldsymbol{\omega} \leftarrow$ acceptanceCriteria($\boldsymbol{\omega}^*$, $\boldsymbol{\omega}'$)
       //Update the weights of the operators
11     $\sigma_{rem} \leftarrow$ updateWeights($\sigma_{rem}$, $f$, $\boldsymbol{\omega}'$, $\boldsymbol{\omega}^*$), $\sigma_{ins} \leftarrow$ updateWeights($\sigma_{ins}$, $f$, $\boldsymbol{\omega}'$, $\boldsymbol{\omega}^*$)
12 return $\boldsymbol{\omega}', \hat{\Omega}$
```

6.A.1.1 Update the set of solutions based on different routes: *updateMultipleRoutes*

Algorithm 6.8: updateMultipleRoutes - Update the set that contains solutions with different routes

```
Data: the set of solutions ($\hat{\Omega}$) and the new solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$)
1  add $\leftarrow$ true
2  for $\hat{\boldsymbol{\omega}} \in \hat{\Omega}$ do
       //Check if the routes are equal
3      if $\boldsymbol{x} = \hat{\boldsymbol{x}}$ then
4          add $\leftarrow$ false
5          break
   //Add the route to the set
6  if add = true then
7      $\hat{\Omega} \leftarrow \hat{\Omega} \cup \{\boldsymbol{\omega}\}$
8  return $\hat{\Omega}$
```

Algorithm 6.8 describes the function used to update the set that stores solutions composed by different routes. To check if a new solution needs to be added to the set, the algorithm iterates through the solutions and verifies if the routes are equal (lines 1 - 4). The new solution is added to the set if it is composed by routes different from those already in the set (lines 5 - 6).

6.A.2 Update the non dominated solutions: *updateNonDominatedSet*

Algorithm 6.9 studies the dominance relation between the new solutions and the ones in the set (line 1 - 9).

There are three possible scenarios:

- If the solutions are equal, the new solution cannot be added to the set (lines 4 - 5).
- In case the new solution is dominated by any one of the set (lines 6 - 7), the solution will not be added to the set.
- If the new solution dominates any one of the set (lines 8 - 9), it is added to the set (line 13) and all the dominated solutions are removed (lines 11 - 12).

Algorithm 6.9: updateNonDominatedSet - Update the set that contains solutions with different routes

```
   Data: the set of non dominated solutions ($\Omega$) and the new solution ($\bar{\boldsymbol{\omega}}$)
1  $d \leftarrow \emptyset$ //Solutions dominated by the new one
2  $stop \leftarrow false$ //Stop if the new solution is dominated
   //Check the dominance relation of the new solution with the ones on the set
3  for $\boldsymbol{\omega} \in \Omega$ do
4      if $\boldsymbol{\omega} = \bar{\boldsymbol{\omega}}$ then
           //The solutions are equal
5          $stop \leftarrow true$, break
6      else if $(f_1(\boldsymbol{\omega}) \leq f_1(\bar{\boldsymbol{\omega}})$ and $f_2(\boldsymbol{\omega}) \leq f_2(\bar{\boldsymbol{\omega}}))$ and $(f_1(\boldsymbol{\omega}) < f_1(\bar{\boldsymbol{\omega}})$ or $f_2(\boldsymbol{\omega}) < f_2(\bar{\boldsymbol{\omega}}))$ then
           //The new solution is dominated
7          $stop \leftarrow true$, break
8      else if $(f_1(\bar{\boldsymbol{\omega}}) \leq f_1(\boldsymbol{\omega})$ and $f_2(\bar{\boldsymbol{\omega}}) \leq f_2(\boldsymbol{\omega}))$ and $(f_1(\bar{\boldsymbol{\omega}}) < f_1(\boldsymbol{\omega})$ or $f_2(\bar{\boldsymbol{\omega}}) < f_2(\boldsymbol{\omega}))$ then
           //The solution of the set is dominated by the new one
9          $d \leftarrow d \cup \{\boldsymbol{\omega}\}$
   //Update the non dominated set
10 if $stop = false$ then
11     if $d \neq \emptyset$ then
           //Remove dominated solutions from the set
12         $\Omega \leftarrow \Omega \setminus \{d\}$
       //Add the new solution to the set
13     $\Omega \leftarrow \Omega \cup \{\bar{\boldsymbol{\omega}}\}$
14 return $\hat{\Omega}$
```

6.A.3 Filter the set of solutions: *filterSet*

Algorithm 6.10 filters the set of solutions in order to only keep those that are close to the non dominated ones. To do this, for each solution of $\hat{\Omega}$, the schedules that optimize the two different lexicographic functions are obtained (line 3). Then, if the distance between any of the three solutions (original, optimizing welfare over cost or optimizing cost over welfare) and the set of non dominated solutions is lower than a certain small value, υ_1, the solutions are added to the set $\hat{\Omega}_1$ (lines 4 - 5). In case the distance is lower than a moderate value, υ_2, the solutions are added to $\hat{\Omega}_2$ (lines 6 - 7).

Algorithm 6.10: filterSet - Keep solutions close to the non dominated ones

```
Data: the set of solutions composed by different routes ($\hat{\Omega}$) and the set of non dominated
      solutions ($\Omega$)
1 $\hat{\Omega}_1, \hat{\Omega}_2 \leftarrow \emptyset$ //Initialize sets
2 for $\hat{\omega} \in \hat{\Omega}$ do
      //Get schedules of $\hat{\omega}$ that optimise each lexicographic objective function
3     $\hat{\omega}_{wc} \leftarrow$ schedule($\hat{\omega}, f_{wc}$), $\hat{\omega}_{cw} \leftarrow$ schedule($\hat{\omega}, f_{cw}$)
4     if distance($\hat{\omega}, \Omega$) $< \upsilon_1$ or distance($\hat{\omega}_{wc}, \Omega$) $< \upsilon_1$ or distance($\hat{\omega}_{cw}, \Omega$) $< \upsilon_1$ then
          //If the distance with $\Omega$ is small
5         $\hat{\Omega}_1 \leftarrow \hat{\Omega}_1 \cup \{\hat{\omega}, \hat{\omega}_{wc}, \hat{\omega}_{cw}\}$
6     else if distance($\hat{\omega}, \Omega$) $< \upsilon_2$ or distance($\hat{\omega}_{wc}, \Omega$) $< \upsilon_2$ or distance($\hat{\omega}_{cw}, \Omega$) $< \upsilon_2$ then
          //If the distance with $\Omega$ is moderate
7         $\hat{\Omega}_2 \leftarrow \hat{\Omega}_2 \cup \{\hat{\omega}, \hat{\omega}_{wc}, \hat{\omega}_{cw}\}$
8 return $\hat{\Omega}_1, \hat{\Omega}_2$
```

6.A.4 Update the schedule of a solution: *schedule*

To obtain the schedule of a solution, so that the welfare is prioritized over the cost (or the cost is prioritized over the welfare) the methodology described in Chapter 3 Subsection 3.5 is used. The only difference is that, in this case, the new schedule must be obtained for every route that composes the solution.

6.A.5 Update the set of solutions: *updateSetsOfSolutions*

Algorithm 6.11 updates the sets of solutions to add those that are close to the non dominated ones. If the distance between the solution and the non dominated set is lower than a certain small value, υ_1, the solution is added to $\hat{\Omega}_1$ (lines 1 - 2). In case the distance is lower than a moderate value, υ_2, the solution is added to $\hat{\Omega}_2$ (lines 3 - 4).

Algorithm 6.11: updateSetsOfSolutions - Add solutions close to the non dominated ones

```
Data: the sets of solutions ($\hat{\Omega}_1, \hat{\Omega}_2$), the non dominated solutions ($\Omega$) and the new solution
      ($\omega$)
1 if distance($\omega, \Omega$) $< \upsilon_1$ then
      //If the distance with $\Omega$ is small
2     $\hat{\Omega}_1 \leftarrow \hat{\Omega}_1 \cup \{\omega\}$
3 else if distance($\omega, \Omega$) $< \upsilon_2$ then
      //If the distance with $\Omega$ is moderate
4     $\hat{\Omega}_2 \leftarrow \hat{\Omega}_2 \cup \{\omega\}$
5 return $\hat{\Omega}_1, \hat{\Omega}_2$
```

6.A.6 Get earliest and latest starting times: *getEarliestLatest*

This function is the one described in Section 5.1.1.

Example 6.A.1. Illustration of Algorithm 5.2.

The earliest starting times, for the route presented in Example 6.2.1, according to hard time windows are:

$j = 1.\ t^e_1 = \max\{\underline{\alpha}_1, \underline{\gamma}\} = \max\{0, 0\} = 0.$

$j = 2.\ t^e_2 = \max\{\underline{\alpha}_2, t^e_1 + \eta_1 + \theta_{1,2}\} = \max\{120, 0 + 60\} = 120.$

$j = 3.\ t^e_3 = \max\{\underline{\alpha}_3, t^e_2 + \eta_2 + \theta_{2,3}\} = \max\{60, 120 + 60\} = 180.$

$j = 4.\ t^e_4 = \max\{\underline{\alpha}_4, t^e_3 + \eta_3 + \theta_{3,4}\} = \max\{180, 180 + 60\} = 240.$

$j = 5.\ t^e_5 = \max\{\underline{\alpha}_5, t^e_4 + \eta_4 + \theta_{4,5}\} = \max\{450, 240 + 60\} = 450.$

$j = 6.\ t^e_6 = \max\{\underline{\alpha}_6, t^e_5 + \eta_5 + \theta_{5,6}\} = \max\{540, 450 + 60\} = 540.$

The latest starting times according to hard time windows are:

$j = 6.\ t^l_6 = \min\{\bar{\alpha}_6 - \eta_6, \bar{\gamma} - \eta_6\} = \min\{840 - 60, 840 - 60\} = 780.$

$j = 5.\ t^l_5 = \min\{\bar{\alpha}_5 - \eta_5, t^l_6 - \theta_{5,6} - \eta_5\} = \min\{780 - 60, 780 - 60\} = 720.$

$j = 4.\ t^l_4 = \min\{\bar{\alpha}_4 - \eta_4, t^l_5 - \theta_{4,5} - \eta_4\} = \min\{660 - 60, 780 - 60\} = 600.$

$j = 3.\ t^l_3 = \min\{\bar{\alpha}_3 - \eta_3, t^l_4 - \theta_{3,4} - \eta_3\} = \min\{510 - 60, 600 - 60\} = 450.$

$j = 2.\ t^l_2 = \min\{\bar{\alpha}_2 - \eta_2, t^l_3 - \theta_{2,3} - \eta_2\} = \min\{330 - 60, 450 - 60\} = 270.$

$j = 1.\ t^l_1 = \min\{\bar{\alpha}_1 - \eta_1, t^l_2 - \theta_{1,2} - \eta_1\} = \min\{240 - 60, 270 - 60\} = 180.$

6.A.7 Obtain the maximum time the service can be delayed or advanced according to soft time windows: *getTimeStw*

Algorithm 6.12 computes the maximum time a service can be advanced or delayed, according to the soft time window penalization.

Algorithm 6.12: getTimeStw - Get maximum advance and delay time of the service according to stw

```
Data: the service ($j$), the route ($R$), the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$), the earliest starting times ($t^e$)
      and the latest starting times ($t^l$)
$a \leftarrow 0$, $d \leftarrow 0$ //Initialize advance and delay times
if $t_j < \underline{\beta}_j$ then
    //The service is before its time window
    $d \leftarrow \min\{\bar{\beta}_j - \eta_j - t_j, t^l_j - t_j\}$
else if $t_j > \bar{\beta}_j - \eta_j$ then
    //The service is after its time window
    $a \leftarrow \min\{t_j - \underline{\beta}_j, t_j - t^e_j\}$
else
    if $\min\{\bar{\beta}_j - \eta_j - t_j, t^l_j - t_j\} \neq 0$ then
        //The service is within its soft time window, but can be delayed
        $d \leftarrow \min\{\bar{\beta}_j - \eta_j - t_j, t^l_j - t_j\}$
    if $\min\{t_j - \underline{\beta}_j, t_j - t^e_j\} \neq 0$ then
        //The service is within its soft time window, but can be advanced
        $a \leftarrow \min\{t_j - \underline{\beta}_j, t_j - t^e_j\}$
return $d, a$
```

Three possible options are considered:

- If the service is scheduled before its soft time window (line 2), the delay is the time needed to finish the service at the same time as its soft time window (line 3).
- If the service ends after its soft time window (line 4), the advance time is the one needed for the service to start at its soft time window (line 5).
- In case the service is within its soft time window, the delay is the time needed to end the service when its soft time window finishes (lines 7 - 8). The advance is the time needed to start the service at its soft time window (lines 9 - 10).

Example 6.A.2. Illustration of Algorithm 6.12.

The data given by Example 6.2.1 are: the service to move ($j = 4$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the schedule to modify ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$), the earliest starting times (t^e) and the latest starting times (t^l).

In this case, Service $j = 4$ is scheduled within its soft time window, $t_4 = 390 \in [\underline{\beta}_4, \bar{\beta}_4 - \eta_4] = [240, 540]$. Therefore, the maximum delay time is $d = \min\{\bar{\beta}_4 - \eta_4 - t_4, t^l_4 - t_4\} = \min\{600 - 60 - 390, 600 - 390\} = 150$ (line 8). The maximum advance time is $a = \min\{t_4 - \underline{\beta}_4, t_4 - t^e_4\} = \min\{390 - 240, 390 - 240\} = 150$ (line 10).

6.A.8 Get services affected by the delay time: *getAffectedServicesDelay*

Algorithm 6.13: getAffectedServicesDelay - Get services affected by a potential delay of j

```
Data: the service ($j$), the route ($R$), the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$), the delay time ($d$) and the
      followers of $j$ ($\bar{R}$)
1 $\hat{R} \leftarrow \emptyset$
  //Delay the service
2 $t_j \leftarrow t_j + d$
3 for $k \in \bar{R}$ do
4     if $t_k < t_{k-1} + \eta_{k-1} + \theta_{k-1,k}$ then
          //The start of the service is affected by the delay
5         $t_k \leftarrow t_{k-1} + \eta_{k-1} + \theta_{k-1,k}$
6         $\hat{R} \leftarrow \hat{R} \cup \{k\}$
7     else
8         break
9 return $\hat{R}$
```

Algorithm 6.13 obtains the services that will be affected by the delay of a given service. The method begins delaying service j (line 2) and then, for each of its followers, checks if they will be affected by the delay of j (lines 3 - 8).

Example 6.A.3. Illustration of Algorithm 6.13.

The data provided by Example 6.2.1 are: the service to move ($j = 4$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the schedule to modify ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$), the delay time ($d = 150$) and the followers of 4 ($\bar{R} = \{5, 6\}$).

The algorithm starts initializing the set of affected services, $\hat{R} = \emptyset$, and delaying service 4, $t_4 = t_4 + d = 390 + 150 = 540$. Then, the method checks if the followers are affected by the delay of 4:

$k = 5$. In this case, $t_5 = 510$ and $t_4 + \eta_4 + \theta_{4,5} = 540 + 60 = 600$, which means that 5 is affected by the delay (line 4). Therefore, $t_5 = t_4 + \eta_4 + \theta_{4,5} = 600$ and $\hat{R} = \hat{R} \cup \{6\} = \{5\}$ (lines 5 - 6).

$k = 6$. In this case, $t_6 = 600$ and $t_5 + \eta_5 + \theta_{5,6} = 600 + 60 = 660$, which means that 6 is affected by the delay (line 4). Therefore, $t_6 = t_5 + \eta_5 + \theta_{5,6} = 660$ and $\hat{R} = \hat{R} \cup \{5\} = \{5, 6\}$ (lines 5 - 6).

Example 6.A.4. Illustration of Algorithm 6.13.

The data provided by Example 6.2.1 are: the service to move ($j = 4$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the schedule to modify ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$), the delay time ($d = 210$) and the followers of 4 ($\bar{R} = \{5, 6\}$).

The algorithm starts initializing the set of affected services, $\hat{R} = \emptyset$, and delaying service 4, $t_4 = t_4 + d = 390 + 210 = 600$. Then, the method checks if the followers are affected by the delay of 4:

$k = 5$. In this case, $t_5 = 510$ and $t_4 + \eta_4 + \theta_{4,5} = 600 + 60 = 660$, which means that 5 is affected by the delay (line 4). Therefore, $t_5 = t_4 + \eta_4 + \theta_{4,5} = 660$ and $\hat{R} = \hat{R} \cup \{6\} = \{5\}$ (lines 5 - 6).

$k = 6$. In this case, $t_6 = 600$ and $t_5 + \eta_5 + \theta_{5,6} = 660 + 60 = 720$, which means that 6 is affected by the delay (line 4). Therefore, $t_6 = t_5 + \eta_5 + \theta_{5,6} = 720$ and $\hat{R} = \hat{R} \cup \{5\} = \{5, 6\}$ (lines 5 - 6).

6.A.9 Delay time so the penalization does not increase: *updateDelay*

Algorithm 6.14 updates the delay time of the service, in order to guarantee that by delaying the service the penalization of the route will not be increased. First, the possible delay times for the services are obtained (lines 1 - 3). Then, for each possible delay time (line 4), the algorithm computes the penalization of the original (lines 6 - 10) and new (lines 11 - 16) schedules of the service, which are used to obtain the change in the penalization (line 17). After that, for each follower, their change of penalization is obtained (lines 17 - 32). Finally, the delay time is updated if the penalization does not increase with the delay (lines 35 - 36).

Algorithm 6.14: updateDelay - Get maximum delay to not increase penalization

```
Data: the service ($j$), the route ($R$), the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$), the delay time ($d$) and the
      affected services ($\hat{R}$)
//Possible delay times of $j$
1  $D \leftarrow$ delayTimes($j$, $R$, $\bar{\boldsymbol{\omega}}$, $d$, $\hat{R}$)
2  if $d \notin D$ then
3      $D \leftarrow D \cup \{d\}$
   //Get maximum delay time that does not increase penalization
4  $\bar{d} \leftarrow 0$
5  for $\hat{d} \in sorted(D)$ do
6      $\sigma \leftarrow 0$ //Change in the penalization
7      $\sigma_o \leftarrow 0$ //Old penalization
8      if $t_j < \underline{\beta}_j$ then
9          $\sigma_o \leftarrow \underline{\beta}_j - t_j$ //The service is before its stw start
10     else if $t_j + \eta_j \geq \bar{\beta}_j$ then
11         $\sigma_o \leftarrow t_j + \eta_j - \bar{\beta}_j$ //The service ends after its stw ends
12     $\hat{t}_j \leftarrow t_j + \hat{d}$ //Delay the service
13     $\sigma_n \leftarrow 0$ //New penalization
14     if $\hat{t}_j < \underline{\beta}_j$ then
15         $\sigma_n \leftarrow \underline{\beta}_j - \hat{t}_j$ //The delayed service is before its stw start
16     else if $\hat{t}_j + \eta_j \geq \bar{\beta}_j$ then
17         $\sigma_n \leftarrow \hat{t}_j + \eta_j - \bar{\beta}_j$ //The delayed service is before its stw start
18     $\sigma \leftarrow \sigma + (\sigma_n - \sigma_o)$
       //Delay the affected services
19     for $k \in \hat{R}$ do
20         $\bar{t}_k \leftarrow \hat{t}_{k-1} + \eta_{k-1} + \theta_{k-1,k}$ //Earliest start of $k$ according its predecessors
21         if $t_k < \bar{t}_k$ then
               //If $k$ is affected by the delay
22             $\hat{t}_k \leftarrow \bar{t}_k$
23             $\sigma_o \leftarrow 0$ //Old penalization
24             if $t_k < \underline{\beta}_k$ then
25                 $\sigma_o \leftarrow \underline{\beta}_k - t_k$ //The service is before its stw start
26             else if $t_k + \eta_k \geq \bar{\beta}_k$ then
27                 $\sigma_o \leftarrow t_k + \eta_k - \bar{\beta}_k$ //The service ends after its stw ends
28             $\sigma_n \leftarrow 0$ //New penalization
29             if $\hat{t}_k < \underline{\beta}_k$ then
30                 $\sigma_n \leftarrow \underline{\beta}_k - \hat{t}_k$ //The delayed service is before its stw start
31             else if $\hat{t}_k + \eta_k \geq \bar{\beta}_k$ then
32                 $\sigma_n \leftarrow \hat{t}_k + \eta_k - \bar{\beta}_k$ //The delayed service is before its stw start
33             $\sigma \leftarrow \sigma + (\sigma_n - \sigma_o)$
34         else
35             break
36     if $\sigma \leq 0$ then
37         $\bar{d} \leftarrow \hat{d}$ //If the penalization did not increase
38     else
39         break
40 return $\bar{d}$
```

Example 6.A.5. Illustration of Algorithm 6.14.

The input data given by Example 6.2.1 are: the service to move ($j = 4$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the schedule to modify ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 390$ and $t_6 = 600$), the delay time ($d = 150$) and the affected services ($\bar{R} = \{5, 6\}$).

The first step is to obtain the list of possible delay times using function ***delayTimes*** (see Example 6.A.6 for more details), $D = \{120, 150\}$. Then, the algorithm iterates through the delay times to get the largest one that does not increase the penalization:

$\hat{d} = 120$. Compute how the penalization of each service changes when delaying 4 120 minutes:

$j = 4$. The original schedule of Service 4 is $t_4 = 390 \in [\underline{\beta}_4, \bar{\beta}_4 - \eta_4] = [240, 540]$, that is, its original penalization is $\sigma_o = 0$ (lines 7 - 11). The delayed schedule is $\hat{t}_4 = t_4 + \hat{d} = 510 \in [\underline{\beta}_4, \bar{\beta}_4 - \eta_4] = [240, 540]$, which means that its new penalization is $\sigma_n = 0$ (lines 13 - 17). Therefore, there is no change in the penalization of Service 4, $\sigma = 0$ (line 18).

$k = 5$. The delayed schedule of Service 5 is $\bar{t}_5 = \hat{t}_4 + \eta_4 + \theta_{4,5} = 510 + 60 = 570$ and $t_5 = 510 < \bar{t}_5$. Therefore, $\hat{t}_5 = \bar{t}_5 = 570$ (lines 21 - 21). The original schedule of 5 is $t_5 = 510 \in [\underline{\beta}_5, \bar{\beta}_5 - \eta_5] = [480, 570]$, that is, its original penalization is $\sigma_o = 0$ (lines 22 - 27). Since $\hat{t}_5 = 570 \in [\underline{\beta}_5, \bar{\beta}_5 - \eta_5] = [480, 570]$, its new penalization is $\sigma_n = 0$ (lines 28 - 32). Which means that there is no change in the penalization of Service 5, $\sigma = 0$ (line 33).

$k = 6$. The delayed schedule of Service 6 is $\bar{t}_6 = \hat{t}_5 + \eta_5 + \theta_{5,6} = 570 + 60 = 630$ and $t_6 = 600 < \bar{t}_6$. Therefore, $\hat{t}_6 = \bar{t}_6 = 630$ (lines 21 - 21). The original schedule of 6 is $t_6 = 600 \in [\underline{\beta}_6, \bar{\beta}_6 - \eta_6] = [570, 660]$, that is, its original penalization is $\sigma_o = 0$ (lines 22 - 27). Since $\hat{t}_6 = 630 \in [\underline{\beta}_6, \bar{\beta}_6 - \eta_6] = [570, 660]$, its new penalization is $\sigma_n = 0$ (lines 28 - 32). Which means that there is no change in the penalization of Service 6, $\sigma = 0$ (line 33).

Thus, $\sigma \leq 0$ and the maximum delay time is updated to $\bar{d} = \hat{d} = 120$ (lines 36 - 37), as it can be seen in Figure 6.12.

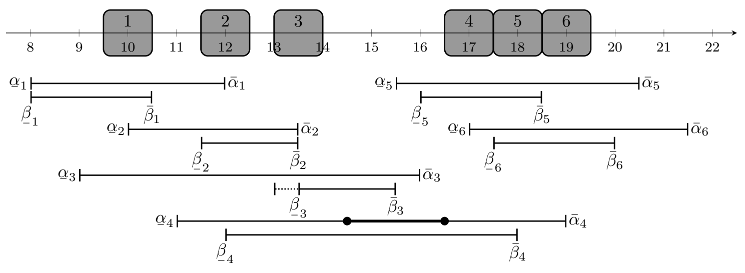


Figure 6.12: Delay of 120 minutes.

$\hat{d} = 150$. Compute how the penalization of each service changes when delaying 4 150 minutes:

$j = 4$. The original schedule of Service 4 is $t_4 = 390 \in [\underline{\beta}_4, \bar{\beta}_4 - \eta_4] = [240, 540]$, that is, its original penalization is $\sigma_o = 0$ (lines 7 - 11). The delayed schedule is $\hat{t}_4 = t_4 + \hat{d} = 540 \in [\underline{\beta}_4, \bar{\beta}_4 - \eta_4] = [240, 540]$, which means that its new penalization is $\sigma_n = 0$ (lines 13 - 17). Therefore, there is no change in the penalization of Service 4, $\sigma = 0$ (line 18).

$k = 5$. The delayed schedule of Service 5 is $\bar{t}_5 = \hat{t}_4 + \eta_4 + \theta_{4,5} = 540 + 60 = 600$ and $t_5 = 510 < \bar{t}_5$, meaning that $\hat{t}_5 = \bar{t}_5 = 600$ (lines 21 - 21). The original schedule of 5 is $t_5 = 510 \in [\underline{\beta}_5, \bar{\beta}_5 - \eta_5] = [480, 570]$, that is, its original penalization is $\sigma_o = 0$ (lines

22 - 27). Since $\hat{t}_5 = 600 > \bar{\beta}_5 - \eta_5 = 570$ its new penalization is $\sigma_n = \hat{t}_5 + \eta_5 - \bar{\beta}_5 = 600 + 60 - 630 = 30$ (lines 31 - 32). Therefore, the change in the penalization of Service 5 is $\sigma = 30$ (line 33).

$k = 6$. The delayed schedule of Service 6 is $\bar{t}_6 = \hat{t}_5 + \eta_5 + \theta_{5,6} = 600 + 60 = 660$ and $t_6 = 600 < \bar{t}_6$, so $\hat{t}_6 = \bar{t}_6 = 660$ (lines 21 - 21). The original schedule of 6 is $t_6 = 600 \in [\underline{\beta}_6, \bar{\beta}_6 - \eta_6] = [570, 660]$, that is, its original penalization is $\sigma_o = 0$ (lines 22 - 27). Since $\hat{t}_6 = 660 \in [\underline{\beta}_6, \bar{\beta}_6 - \eta_6] = [570, 660]$ its new penalization is $\sigma_n = 0$ (lines 38 - 32). Therefore, there is no change in the penalization of Service 6, $\sigma = 0$ (line 33).

Thus $\sigma > 0$ and the maximum delay time is not updated, $\bar{d} = 120$, (lines 36 - 39) as it can be seen in Figure 6.13.

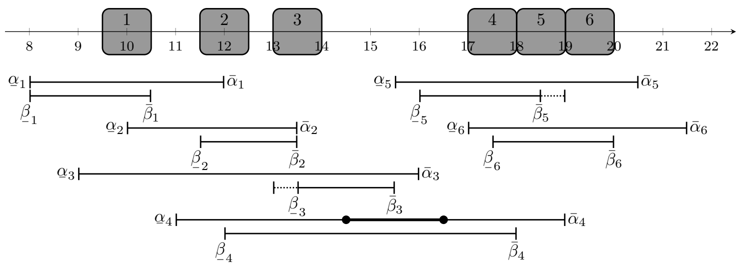


Figure 6.13: Delay of 150 minutes.

6.A.9.1 Get list of potential delay times: *delayTimes*

To obtain the delay times that can result in a change on the penalization Algorithm 6.15 is used. For the given service, the method obtains the delay time necessary for it to start at the beginning, and finish at the end, of its soft time window (lines 2 - 6). The same is done for each follower affected by the delay (lines 8 - 37). Two possibilities are considered:

- If the follower is scheduled within its soft time window, the delay[1] time needed for it to end when the window finishes is computed (lines 8 - 17).
- In case the service starts before the soft time window, the delay time necessary for the service to start at its soft time window is obtained (lines 19 - 28). In addition, the algorithm also gets the delay time that would cause the service to end at the same time as its soft time window (lines 28 - 37).

[1]Note that the delay times obtained are those needed to delay the given service j so that the penalization of its followers changes.

Algorithm 6.15: delayTimes - Get possible delay times to not increase penalization

```
Data: the service ($j$), the route ($R$), the solution ($\boldsymbol{\omega} = (\boldsymbol{x},\boldsymbol{t})$), the delay time ($d$) and the set
      of affected services ($\hat{R}$)
$D \leftarrow \emptyset$
if $t_j + d \geq \underline{\beta}_j$ then
    //The delayed service is after its stw start
    if $\underline{\beta}_j - t_j > 0$ then
        $D \leftarrow D \cup \{\underline{\beta}_j - t_j\}$
    if $t_j + d \geq \bar{\beta}_j - \eta_j$ and $\bar{\beta}_j - \eta_j - t_j > 0$ then
        //The delayed service is after its stw end
        $D \leftarrow D \cup \{\bar{\beta}_j - \eta_j - t_j\}$
for $k \in \hat{R}$ do
    if $\underline{\beta}_k \leq t_k \leq \bar{\beta}_k - \eta_k$ then
        //Get delay time needed for the service leaves its stw
        $\epsilon_1 \leftarrow t_k - t_j$
        //Service time and travel time between $j$ and $k$
        $\epsilon_2 \leftarrow 0$
        for $l \in \{j, ..., k-1\}$ do
            $\epsilon_2 \leftarrow \epsilon_2 + \eta_l + \theta_{l,l+1}$
        $\epsilon_3 \leftarrow \epsilon_1 - \epsilon_2$ //Free time between services $j$ and $k$
        $\epsilon_4 \leftarrow \bar{\beta}_k - \eta_k - t_k$ //Delay time needed for $k$ to end at its stw
        $\epsilon_5 \leftarrow \epsilon_4 + \epsilon_3$ //Delay time of $j$ needed to have $k$ ends at its stw
        if $0 < \epsilon_5 \leq d$ and $\epsilon_5 \notin D$ then
            $D \leftarrow D \cup \{\epsilon_5\}$
    else if $t_k < \underline{\beta}_k$ then
        //Get delay time needed for the service enters its stw
        $\epsilon_1 \leftarrow t_k - t_j$
        //Service time and travel time between $j$ and $k$
        $\epsilon_2 \leftarrow 0$
        for $l \in \{j, ..., k-1\}$ do
            $\epsilon_2 \leftarrow \epsilon_2 + \eta_l + \theta_{l,l+1}$
        $\epsilon_3 \leftarrow \epsilon_1 - \epsilon_2$ //Free time between services $j$ and $k$
        $\epsilon_4 \leftarrow \underline{\beta}_k - t_k$ //Delay time needed for $k$ to enter its stw
        $\epsilon_5 \leftarrow \epsilon_4 + \epsilon_3$ //Delay time of $j$ needed to have $k$ enters its stw
        if $0 < \epsilon_5 \leq d$ then
            if $\epsilon_5 \notin D$ then
                $D \leftarrow D \cup \{\epsilon_5\}$
            $\epsilon_4 \leftarrow \bar{\beta}_k - \eta_k - t_k$ //Delay time needed for $k$ to end at its stw
            $\epsilon_5 \leftarrow \epsilon_4 + \epsilon_3$ //Delay time of $j$ needed to have $k$ ends at its stw
            if $0 < \epsilon_5 \leq d$ and $\epsilon_5 \notin D$ then
                $D \leftarrow D \cup \{\epsilon_5\}$
return $D$
```

Example 6.A.6. Illustration of Algorithm 6.15.

According to Example 6.A.5, the input data are: the service to move ($j = 4$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the schedule to modify ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$), the delay time ($d = 150$) and the affected services ($\bar{R} = \{5, 6\}$).

First, the list of delay times $D = \emptyset$ is initialized. Since $t_4 + d = 390 + 150 = 540 = \bar{\beta}_4 - \eta_4 =$

$600 - 60 = 540$ (line 5) the delay time of Service 4 is $D = D \cup \{\bar{\beta}_4 - \eta_4 - t_4\} = \{540 - 390\} = 150$. Then, for each of the affected services, the delay times that could change their penalization are computed:

$k = 5$. In this case, $t_5 = 510 \in [\underline{\beta}_5, \bar{\beta}_5 - \eta_5] = [480, 570]$ (line 8), which means that the delay time of $j = 4$ so $k = 5$ ends at its soft time window must be obtained:

- The time between the schedules of 4 and 5 is $\epsilon_1 = t_5 - t_4 = 510 - 390 = 120$ (line 9).
- The service and travel times between 4 and 5 is $\epsilon_2 = \eta_4 + \theta_{4,5} = 60$ (lines 9 - 12).
- The free time between 4 and 5 is $\epsilon_3 = \epsilon_1 - \epsilon_2 = 120 - 60 = 60$ (line 13).
- The delay time of 5 so it end when its soft time window finishes is $\epsilon_4 = \bar{\beta}_5 - \eta_5 - t_5 = 630 - 60 - 510 = 60$.
- The delay time of 4 so Service 5 ends with its soft time window is $\epsilon_5 = \epsilon_4 + \epsilon_3 = 60 + 60 = 120$.

Finally, $\epsilon_5 = 120$ is added to the list, $D = D \cup \{\epsilon_5\} = \{150, 120\}$ (lines 16 - 17).

$k = 6$. In this case, $t_6 = 600 \in [\underline{\beta}_6, \bar{\beta}_6 - \eta_6] = [570, 660]$ (line 8) which means that the delay time of $j = 4$ so $k = 6$ ends at its soft time window must be obtained:

- The time between the schedules of 4 and 6 is $\epsilon_1 = t_6 - t_j = 600 - 390 = 210$ (line 9).
- The service and travel times between 4 and 6 is $\epsilon_2 = \eta_4 + \theta_{4,5} + \eta_5 + \theta_{5,6} = 120$ (lines 9 - 12).
- The free time between 4 and 6 is $\epsilon_3 = \epsilon_1 - \epsilon_2 = 210 - 120 = 90$ (line 13).
- The delay time of 6 so it end when its soft time window finishes is $\epsilon_4 = \bar{\beta}_6 - \eta_6 - t_6 k = 720 - 60 - 600 = 60$.
- The delay time of 4 so Service 6 ends with its soft time window is $\epsilon_5 = \epsilon_4 + \epsilon_3 = 60 + 90 = 150$.

Therefore, ϵ_5 is not added to the list, because 150 is already on it.

6.A.10 Get services affected by the advance time: *getAffectedServicesAdvance*

The services that will be affected by the advance of a given service are obtained using Algorithm 6.16.

Algorithm 6.16: getAffectedServicesAdvance - Get services affected by a potential advance of j

```
Data: the service ($j$), the route ($R$), the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$), the advance time ($a$) and the
      predecesors of $j$ ($\bar{R}$)
1 $\hat{R} \leftarrow \emptyset$
2 $t_j \leftarrow t_j - a$ //Advance service $j$
3 for $k \in reversed(\bar{R})$ do
4     if $t_k > t_{k+1} - \theta_{k,k+1} - \eta_k$ then
          //If the new start of $k$ is affected by the advance of $j$
5         $t_k \leftarrow t_{k+1} - \theta_{k,k+1} - \eta_k$
6         $\hat{R} \leftarrow \hat{R} \cup \{k\}$
7     else
8         break
9 return $\hat{R}$
```

First, the service is advanced (line 2) and then, it is check if its predecessors are affected by the advance (lines 3 - 8). This is done updating the starting time of the service according to the advanced start of its follower (lines 4 - 5) and, if the new starting time is different than the original one, the service is added to the list (line 6). In case the new start of the service is not affected, the method terminates (lines 7 - 8).

Example 6.A.7. Illustration of Algorithm 6.16.

According to Example 6.2.1, the available data are: the service to move ($j = 4$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the schedule to modify ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$), the advance time ($d = 150$) and the predecessors of 4 ($\bar{R} = \{1, 2, 3\}$).

The set of affected services is initialized, $\hat{R} = \emptyset$, and Service $j = 4$ is advanced, $t_4 = t_4 - d = 390 - 150 = 240$. After that, the method checks if the predecessors are affected by the advance of 4:

$k = 3$. In this case, $t_3 = 300$ and $t_4 - \eta_3 - \theta_{3,4} = 240 - 60 = 180$, which means that 3 is affected by the advance (line 4). Therefore, $t_3 = t_4 - \eta_3 - \theta_{3,4} = 180$ and $\hat{R} = \hat{R} \cup \{3\} = \{3\}$ (lines 5 - 6).

$k = 2$. In this case, $t_2 = 210$ and $t_3 - \eta_2 - \theta_{2,3} = 180 - 60 = 120$, which means that 2 is affected by the advance (line 4). Therefore, $t_2 = t_3 - \eta_2 - \theta_{2,3} = 120$ and $\hat{R} = \hat{R} \cup \{2\} = \{3, 2\}$ (lines 5 - 6).

$k = 1$. In this case, $t_1 = 90$ and $t_2 - \eta_{1k} - \theta_{1,2} = 120 - 60 = 60$, which means that 1 is affected by the advance (line 4). Therefore, $t_1 = t_2 - \eta_1 - \theta_{1,2} = 60$ and $\hat{R} = \hat{R} \cup \{1\} = \{3, 2, 1\}$ (lines 5 - 6).

6.A.11 Obtain advance time so the penalization does not increase: *updateAdvance*

Algorithm 6.17 updates the advance time of the service to guarantee that the penalization of the route will not increase when advancing. First, all possible advance times for the services are computed (lines 1 - 3). Then, for each advance time (line 4), the method obtains their penalization before (lines 6 - 10), and after (lines 11 - 16), advancing them. These two values are used to obtain how the penalization changes (line 17). After that, if a service is affected by the advance of its following service (lines 20 - 21), it is advanced and its change of penalization is computed (lines 22 - 32). In case the service is not affected by the advance, the loop ends (lines 33 - 34). Finally, the advance time is updated if the penalization does not increase when the services are advanced (lines 35 - 36). Otherwise, the algorithm terminates (lines 37 - 38).

Algorithm 6.17: updateAdvance - Get maximum advance to not increase penalization

```
Data: the service ($j$), the route ($R$), the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$), the advance time ($a$) and the
      affected services ($\hat{R}$)
//Possible advance times of $j$
1  $A \leftarrow$ advanceTimes($j$, $R$, $\bar{\boldsymbol{\omega}}$, $a$, $\hat{R}$)
2  if $a \notin A$ then
3      $A \leftarrow A \cup \{a\}$
   //Get maximum delay time that does not increase penalization
4  $\bar{a} \leftarrow 0$
5  for $\hat{a} \in sorted(A)$ do
6      $\sigma \leftarrow 0$
7      $\hat{t}_j \leftarrow t_j - \hat{a}$ //Advance the services
8      $\sigma_o \leftarrow 0$ //Old penalization
9      if $t_j < \underline{\beta}_j$ then
10         $\sigma_o \leftarrow \underline{\beta}_j - t_j$ //The service is before its stw start
11     else if $t_j + \eta_j \geq \bar{\beta}_j$ then
12         $\sigma_o \leftarrow t_j + \eta_j - \bar{\beta}_j$ //The service ends after its stw ends
13     $\sigma_n \leftarrow 0$ //New penalization
14     if $\hat{t}_j < \underline{\beta}_j$ then
15         $\sigma_n \leftarrow \underline{\beta}_j - \hat{t}_j$ //The advanced service is before its stw start
16     else if $\hat{t}_j + \eta_j \geq \bar{\beta}_j$ then
17         $\sigma_n \leftarrow \hat{t}_j + \eta_j - \bar{\beta}_j$ //The advanced service is before its stw start
18     $\sigma \leftarrow \sigma + (\sigma_n - \sigma_o)$
19     for $k \in \hat{R}$ do
20         $\bar{t}_k \leftarrow \hat{t}_{k+1} - \eta_k - \theta_{k,k+1}$ //Latest start of $k$ according to its followers
21         if $t_k > \bar{t}_k$ then
               //If $k$ is affected by the advance
22             $\hat{t}_k \leftarrow \bar{t}_k$
23             $\sigma_o \leftarrow 0$ //Old penalization
24             if $t_k < \underline{\beta}_k$ then
25                 $\sigma_o \leftarrow \underline{\beta}_k - t_k$ //The service is before its stw start
26             else if $t_k + \eta_k \geq \bar{\beta}_k$ then
27                 $\sigma_o \leftarrow t_k + \eta_k - \bar{\beta}_k$ //The service ends after its stw ends
28             $\sigma_n \leftarrow 0$ //New penalization
29             if $\hat{t}_k < \underline{\beta}_k$ then
30                 $\sigma_n \leftarrow \underline{\beta}_k - \hat{t}_k$ //The advanced service is before its stw start
31             else if $\hat{t}_k + \eta_k j \geq \bar{\beta}_k$ then
32                 $\sigma_n \leftarrow \hat{t}_k + \eta_k - \bar{\beta}_k$ //The advanced service is before its stw start
33             $\sigma \leftarrow \sigma + (\sigma_n - \sigma_o)$
34         else
35             break
36     if $\sigma \leq 0$ then
           //If by advancing the penalization did not increase
37         $\bar{a} \leftarrow \hat{a}$
38     else
39         break
40 return $\bar{a}$
```

Example 6.A.8. Illustration of Algorithm 6.17.

The data given by Example 6.2.1 are: the service to move ($j = 4$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the schedule to modify ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$), the advance time ($d = 150$) and the affected services ($\bar{R} = \{3, 2, 1\}$).

The first step is to obtain the list of possible advance times using function ***advanceTimes*** (Example 6.A.9), $A = \{150\}$. Then, iterating through the advance times, the largest one that does not increase the penalization is computed:

$\hat{a} = 150$. Obtain how the penalization of each service changes when Service 4 is advanced 150 minutes:

$j = 4$. The original schedule of Service 4 is $t_4 = 390 \in [\underline{\beta}_4, \bar{\beta}_4 - \eta_4] = [240, 540]$, that is, its original penalization is $\sigma_o = 0$ (lines 7 - 11). The advanced schedule is $\hat{t}_4 = t_4 + \hat{d} = 240 \in [\underline{\beta}_4, \bar{\beta}_4 - \eta_4] = [240, 540]$, which means that its new penalization is $\sigma_n = 0$ (lines 13 - 17). Therefore, there is no change in the penalization of Service 4, $\sigma = 0$ (line 18).

$k = 3$. The advanced schedule of Service 3 is $\bar{t}_3 = \hat{t}_4 - \eta_3 - \theta_{3,4} = 240 - 60 = 180$ and $t_3 = 300 > \bar{t}_3$, which means that $\hat{t}_3 = \bar{t}_3 = 180$ (line 22). The original schedule of Service 3 is $t_3 = 300 < \underline{\beta}_3 = 330$, that is, its original penalization is $\sigma_o = \underline{\beta}_3 - t_3 = 330 - 300 = 30$ (line 24 - 25). Since $\hat{t}_3 = 180 < \underline{\beta}_3 = 330$, its new penalization is $\sigma_n = \underline{\beta}_3 - \hat{t}_3 = 330 - 180 = 150$ (lines 29 - 30). Therefore, the change in the penalization of Service 3 is $\sigma = \sigma_n - \sigma_0 = 150 - 30 = 120$ (line 33).

$k = 2$. The advanced schedule of Service 2 is $\bar{t}_2 = \hat{t}_3 - \eta_2 - \theta_{2,3} = 180 - 60 = 120$ and $t_2 = 210 > \bar{t}_2$, which means that $\hat{t}_2 = \bar{t}_2 = 120$ (line 22). The original schedule of Service 2 is $t_2 = 210 = \underline{\beta}_2$, that is, its original penalization is $\sigma_o = 0$ (line 24 - 25). Since $\hat{t}_2 = 120 < \underline{\beta}_2 = 210$, its new penalization is $\sigma_n = \underline{\beta}_2 - \hat{t}_2 = 210 - 120 = 90$ (lines 29 - 30). Therefore, the total change in the penalization is $\sigma = \sigma + (\sigma_n - \sigma_0) = 120 + (90 - 0) = 210$ (line 33).

$k = 1$. The advance schedule of Service 1 is $\bar{t}_1 = \hat{t}_2 - \eta_1 - \theta_{1,2} = 120 - 60 = 60$ and $t_1 = 90 > \bar{t}_1$, which means that $\hat{t}_1 = \bar{t}_1 = 60$ (line 22). The original schedule of Service 1 is $t_1 = 90 \in [\underline{\beta}_1, \bar{\beta}_1 - \eta_1] = [0, 90]$, that is, its original penalization is $\sigma_o = 0$ (line 23 - 27). Since $\hat{t}_k = 60 \in [\underline{\beta}_1, \bar{\beta}_1 - \eta_1] = [0, 90]$, its new penalization is $\sigma_n = 0$ (lines 28 - 32). Therefore, there is no change in the penalization of Service 1, which means that $\sigma = 210$ (line 33).

In this example $\sigma > 0$ so the maximum delay time is not updated, $\bar{a} = 0$, (lines 36 - 39), as can be seen in Figure 6.14.

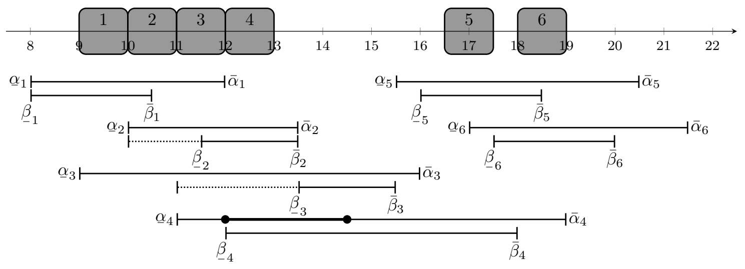


Figure 6.14: Advance of 150 hours.

6.A.11.1 Get list of potential advance times: *advanceTimes*

Algorithm 6.18 computes the advance times that can result in a change on the penalization.

Algorithm 6.18: advanceTimes - Get possible advance times to improve penalization

```
Data: the service ($j$), the route ($R$), the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$), the advance time ($a$) and the
      affected services ($\hat{R}$)
1  $A \leftarrow \emptyset$
2  if $t_j - a \leq \bar{\beta}_j - \eta_j$ then
       //The advanced service is before its stw end
3      if $t_j - (\bar{\beta}_j - \eta_j) > 0$ then
4          $A \leftarrow A \cup \{t_j - (\bar{\beta}_j - \eta_j)\}$
5      if $t_j - a \leq \underline{\beta}_j$ and $t_j - \underline{\beta}_j > 0$ then
           //The advanced service is after its stw end
6          $A \leftarrow A \cup \{t_j - \underline{\beta}_j\}$
7  for $k \in \hat{R}$ do
8      if $\underline{\beta}_k \leq t_k \leq \bar{\beta}_k - \eta_k$ then
           //Get advance time of the service so it leaves its stw
9          $\epsilon_1 \leftarrow t_j - t_k$
           //Service time and travel time between $j$ and $k$
10         $\epsilon_2 \leftarrow 0$
11         for $l \in \{j, ..., k-1\}$ do
12             $\epsilon_2 \leftarrow \epsilon_2 + \eta_l + \theta_{l,l+1}$
13         $\epsilon_3 \leftarrow \epsilon_1 - \epsilon_2$ //Free time between services $j$ and $k$
14         $\epsilon_4 \leftarrow t_k - \underline{\beta}_k$ //Advance time so $k$ to leave its stw
15         $\epsilon_5 \leftarrow \epsilon_4 + \epsilon_3$ //Advance time of $j$ to have $k$ leave its stw
16         if $0 < \epsilon_5 < a$ and $\epsilon_5 \notin A$ then
17             $A \leftarrow A \cup \{\epsilon_5\}$
18     else if $t_k > \bar{\beta}_k - \eta_k$ then
           //Get advance time of the service so it enters its stw
19         $\epsilon_1 \leftarrow t_j - t_k$
20         $\epsilon_2 \leftarrow 0$
21         for $l \in \{j, ..., k-1\}$ do
22             $\epsilon_2 \leftarrow \epsilon_2 + \eta_l + \theta_{l,l+1}$
23         $\epsilon_3 \leftarrow \epsilon_1 - \epsilon_2$ //Free time between services $j$ and $k$
24         $\epsilon_4 \leftarrow t_k - (\bar{\beta}_k - \eta_k)$ //Advance time so $k$ to enter its stw
25         $\epsilon_5 \leftarrow \epsilon_4 + \epsilon_3$ //Advance time of $j$ to have $k$ enter its stw
26         if $0 < \epsilon_5 < a$ then
27             if $\epsilon_5 \notin A$ then
28                 $A \leftarrow A \cup \{\epsilon_5\}$
               //Get advance time of the service so it leaves its stw
29             $\epsilon_4 \leftarrow t_k - \underline{\beta}_k$ //Advance time so $k$ to leave its stw
30             $\epsilon_5 \leftarrow \epsilon_4 + \epsilon_3$ //Advance time of $j$ to have $k$ leave its stw
31             if $0 < \epsilon_5 < a$ and $\epsilon_5 \notin A$ then
32                 $A \leftarrow A \cup \{\epsilon_5\}$
33 return $A$
```

For the given service, two advance times are obtained: the one needed for it to end at the same time as its soft time window, and the time needed for the service to start at the beginning of the soft time window (lines 2 - 6). After that, the same is done for each predecessor affected by the advance (lines 7 - 32). Two possibilities are considered:

- If the predecessor is scheduled within its soft time window, the advance[2] time needed for it to start when the window begins is computed (lines 8 - 17).
- In case the service ends after the soft time window, the advance time needed for the service to end when its soft time window finishes is obtained (lines 19 - 25). Furthermore, the advance time needed for the service to start at the same time as its soft time window is also computed (lines 26 - 32).

Example 6.A.9. Illustration of Algorithm 6.18.

The available data, given by Example 6.A.8, are: the service to move ($j = 4$), the route ($R = \{1,2,3,4,5,6\}$), the schedule to modify ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$), the advance time ($d = 150$) and the affected services ($\bar{R} = \{3,2,1\}$).

First, the list of advance times is initialized $A = \emptyset$. Since $t_4 - d = 390 - 150 = 240 = \underline{\beta}_4$ (line 5) the advance time is $A = A \cup \{t_4 - \bar{\beta}_4\} = \{390 - 240\} = \{150\}$.

Then, for each of the affected services, the advance times that could change their penalization are computed:

$k = 3$. In this case, $t_3 = 300 < \underline{\beta}_3 = 330$, which means that the service cannot be advanced without increasing its penalization.

$k = 2$. In this case, $t_2 = 210 = \underline{\beta}_2$, which means that the service cannot be advanced without increasing, its penalization.

$k = 1$. In this case $t_1 = 90 \in [\underline{\beta}_1, \bar{\beta}_1 - \eta_1] = [60, 120]$ (line 8). Therefore, the advance time of $j = 4$ so Service $k = 1$ starts at its soft time window is obtained:

- The time between the schedules of services 4 and 1 is $\epsilon_1 = t_4 - t_1 = 390 - 90 = 300$ (line 9).
- The service and travel times between services 4 and 1 is $\epsilon_2 = \eta_1 + \theta_{1,2} + \eta_2 + \theta_{2,3} + \eta_3 + \theta_{3,4} = 180$ (lines 9 - 12).
- The free time between services 4 and 1 is $\epsilon_3 = \epsilon_1 - \epsilon_2 = 300 - 180 = 120$ (line 13).
- Advance time of Service 1 so it begins when its soft time window starts is $\epsilon_4 = t_1 - \underline{\beta}_1 = 90 - 0 = 90$.
- Delay time of Service 4 so Service 1 ends with its soft time window is $\epsilon_5 = \epsilon_4 + \epsilon_3 = 90 + 120 = 210$.

In this case ϵ_5 is not added to the list because $210 > 150$.

6.A.12 Obtain the maximum time the service can be delayed or advanced according to the cost: *getTimeCost*

Algorithm 6.19 obtains the minimum and maximum times the service can be moved in order to improve the cost of the schedule. First, the delay times that would either reduce the breaks after the service (line 1) or make the break before the service reach a duration of π_{min} (line 2) are computed. Then, the advance time that reduces the breaks before the service (line 4) or makes the break after the service have a duration of π_{min} (line 5) is obtained.

[2]Note that the advance times obtained are those needed to move the given service j so that the penalization of its predecessors changes.

Algorithm 6.19: getTimeCost - Get possible advance and delay times for j to improve the cost

```
Data: the service ($j$), the route ($R$), the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$), the earliest starting times ($t^e$)
      and the latest starting times ($t^l$)
  //Get delay times
1 $\underline{d}_1, \bar{d}_1 \leftarrow$ delayReduceBreak($j$, $R$, $\boldsymbol{\omega}$, $t^l$)
2 $\underline{d}_2, \bar{d}_2 \leftarrow$ delayIncreaseBreak($j$, $R$, $\boldsymbol{\omega}$, $t^e$)
3 $D \leftarrow \{(\underline{d}_1, \bar{d}_1), (\underline{d}_2, \bar{d}_2)\}$
  //Get advance times
4 $\underline{a}_1, \bar{a}_1 \leftarrow$ advanceReduceBreak($j$, $R$, $\boldsymbol{\omega}$, $t^l$)
5 $\underline{a}_2, \bar{a}_2 \leftarrow$ advanceIncreaseBreak($j$, $R$, $\boldsymbol{\omega}$, $t^e$)
6 $A \leftarrow \{(\underline{a}_1, \bar{a}_1), (\underline{a}_2, \bar{a}_2)\}$
7 return $D, A$
```

Example 6.A.10. Illustration of Algorithm 6.19.

According to Example 6.2.1, the available data are: the service ($j = 4$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the schedule ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$) and the earliest (t^e) and latest starting times (t^l).

Two different delay times are obtained, depending on whether the break would increase or decrease:

Reduce break. The maximum and minimum times Service 4 can be delayed to reduce the breaks after it are $\underline{d}_1 = 0$ and $\bar{d}_1 = 90$, which are obtained using function ***delayReduceBreak*** (Example 6.A.11).

Increase break. The maximum and minimum times Service 4 can be delayed to increase the break before it are $\underline{d}_2 = 90$ and $\bar{d}_2 = 210$, which are obtained using function ***delayIncreaseBreak*** (Example 6.A.12).

Two different advance times are computed, depending on whether the break would increase or decrease:

Reduce break. The maximum and minimum times Service 4 can be advanced to reduce the breaks before it are $\underline{a}_1 = 0$ and $\bar{a}_1 = 120$, which are obtained using function ***advanceReduceBreak*** (Example 6.A.13).

Increase break. The maximum and minimum times Service 4 can be advanced to increase the break after it are $\underline{a}_2 = 60$ and $\bar{a}_2 = 150$, which are obtained using function ***delayIncreaseBreak*** (Example 6.A.14).

6.A.12.1 Get delay time needed to reduce breaks: *delayReduceBreak*

The function described in Algorithm 6.20 checks if the breaks can be reduced by delaying the service. First, the maximum delay time for the service is computed (line 1). Then, the list of services that will be affected by that delay is obtained (line 2). After that, iterating through the affected services (line 5), the algorithm checks if there is a break between them (line 7), in which case the delay time is updated (lines 8 - 16).

Algorithm 6.20: delayReduceBreak - Get delay time to reduce the breaks after the service

```
Data: the service ($j$), the route ($R$), the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$) and the latest starting times ($t^l$)
//Get delay times
1  $d \leftarrow t^l_j - t_j$
2  $\hat{R} \leftarrow$ getAffectedServicesDelay($j$, $R$, $\boldsymbol{\omega}$, $d$, $F(j,R)$) //Get services affected by the delay
3  $\hat{R} \leftarrow \{j\} \cup \hat{R}$ //Add the service to the set
4  $\underline{d}, \bar{d} \leftarrow 0$
   //Check if there is a break between the affected services
5  for $k \in \hat{R}$ do
6      if $k+1 \in R$ then
7          $b \leftarrow t_{k+1} - (t_k + \eta_k + \theta_{k,k+1})$
8          if $0 < b < \pi_{min}$ then
               //Get delay time so the service reaches $k+1$
9              $\epsilon_1 \leftarrow t_{k+1} - t_j$ //Time between $j$ and $k+1$
               //Service time and travel time between $j$ and $k+1$
10             $\epsilon_2 \leftarrow 0$
11             for $l \in \{j, ..., k\}$ do
12                 $\epsilon_2 \leftarrow \epsilon_2 + \eta_l + \theta_{l,l+1}$
13             $\epsilon_3 \leftarrow \epsilon_1 - \epsilon_2$ //Free time between services $j$ and $k+1$
14             $\bar{d} \leftarrow \epsilon_3$
15         else if $b \geq \pi_{min}$ then
16             break
17 return $\underline{d}, \bar{d}$
```

Example 6.A.11. Illustration of Algorithm 6.20.

The data, provided by Example 6.A.10, are: the service ($j = 4$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the schedule ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$, $t_6 = 600$) and the earliest (t^e) and latest starting times (t^l).

The maximum time that the service can be delayed is $d = t^l_4 - t_4 = 600 - 390 = 210$ (line 1) and the services that will be affected by this delay are $\hat{R} = \{5, 6\}$ (Example 6.A.4). Now Service 4 is added to the affected services, $\hat{R} = \{4, 5, 6\}$ and the maximum and minimum delay times are initialized, $\underline{d}, \bar{d} = 0$ (lines 3 - 4).

After that, the algorithm checks if there is a break after the services of $\hat{R}$:

$k = 4$. The break between 4 and 5 is $b = t_5 - (t_4 + \eta_4 + \theta_{4,5}) = 510 - (390 + 60) = 60$ (line 7). Therefore, $b < \pi_{min}$ (line 8), which means that the delay time of 4 needed to reach 5 must be computed:

- The time between the schedules of 4 and 5 is $\epsilon_1 = t_5 - t_4 = 510 - 390 = 120$ (line 9).
- The service and travel times between services 4 and 5 is $\epsilon_2 = \eta_4 + \theta_{4,5} = 60$ (lines 9 - 12).
- The free time between services 4 and 5 is $\epsilon_3 = \epsilon_1 - \epsilon_2 = 120 - 60 = 60$ (line 13).

The delay is set as $\bar{d} = \epsilon_3 = 60$.

$k = 5$. The break between 5 and 6 is $b = t_5 - (t_5 + \eta_5 + \theta_{5,5}) = 600 - (510 + 60) = 30$ (line 7). Therefore, $b < \pi_{min}$ (line 8), which means that the delay time of 44 needed to have service 5 reach 6 must be computed:

- The time between the schedules of 4 and 6 is $\epsilon_1 = t_6 - t_4 = 600 - 390 = 210$ (line 9).

- The service and travel times between services 4 and 6 is $\epsilon_2 = \eta_4 + \theta_{4,5} + \eta_5 + \theta_{5,6} = 120$ (lines 9 - 12).
- The free time between services 4 and 6 is $\epsilon_3 = \epsilon_1 - \epsilon_2 = 210 - 120 = 90$ (line 13).

The delay is set as $\bar{d} = \epsilon_3 = 90$, as it can be seen in Figure 6.15.

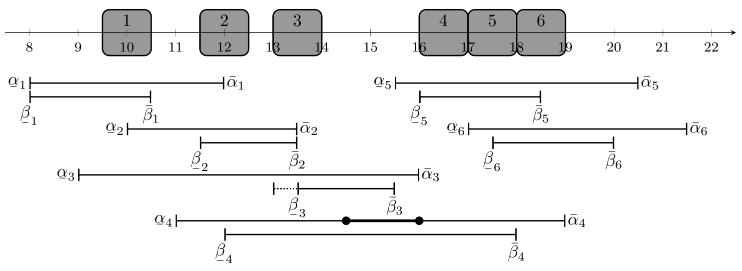


Figure 6.15: Delay to reduce breaks.

6.A.12.2 Get delay time needed to increase the break: *delayIncreaseBreak*

Algorithm 6.20 checks if the break before the service can reach a duration of π_{min} by delaying the service. After obtaining the maximum time the service can be delayed (line 1), it is necessary to check if there is another break in the route with a duration of π_{min} (lines 3 - 7). If there is no other big break, the algorithm tests if setting the start of the service at its latest time results in a break of, at least, π_{min} (lines 8 - 9). In that case, the delay time is set as the maximum one (line 10). Finally, the minimum delay time that guarantees the break will have a duration of π_{min}, is computed (lines 11 - 13).

Algorithm 6.21: delayIncreaseBreak - Get delay time to increase the break before the service

```
Data: the service ($j$), the route ($R$), the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$) and the latest starting times ($t^l$)
   //Get delay times
1  $d \leftarrow t^l_j - t_j$
2  $\underline{d}, \bar{d} \leftarrow 0$
   //Check if there is a big break in the route
3  $b \leftarrow false$
4  for $k \in R$ do
5      if $k \neq 1$ and $k \neq j$ then
6          if $t_k - (t_{k-1} + \eta_{k-1} + \theta_{k-1,k}) \geq \pi_{min}$ then
7              $b \leftarrow true$
   //Check if the break before $j$ can be increased
8  if $j \neq 1$ and $b = false$ then
9      if $t^l_j - (t_{j-1} + \eta_{j-1} + \theta_{j-1,j}) \geq \pi_{min}$ then
           //If the break before $j$ can cave a duration of $\pi_{min}$
10         $\bar{d} \leftarrow d$
           //End of $j-1$ + travel to $j$
11         $\bar{t}_j \leftarrow t_{j-1} + \eta_{j-1} + \theta_{j-1,j}$
12         if $\bar{t}_j + \pi_{min} > t_j$ then
               //Minimum delay of $j$ to get a break of $\pi_{min}$
13             $\underline{d} \leftarrow \bar{t}_j + \pi_{min} - t_j$
14 return $\underline{d}, \bar{d}$
```

Example 6.A.12. Illustration of Algorithm 6.21.

The data, provided by Example 6.A.10, are: the service ($j = 4$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the schedule ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$) and the earliest (t^e) and latest starting times (t^l).

The maximum time that the service can be delayed is $d = t^l_4 - t_4 = 600 - 390 = 210$ and the maximum and minimum delay times are initialized to $\underline{d}, \bar{d} = 0$ (lines 1 - 2).

The first step is to check if there is a break with a duration greater than or equal to π_{min}:

$k = 2$. The break between 1 and 2 is $t_2 - (t_1 + \eta_1 + \theta_{1,2}) = 210 - (90 + 60) = 60$ which does not reach a duration of π_{min} (line 6).

$k = 3$. The break between 2 and 3 is $t_3 - (t_2 + \eta_2 + \theta_{2,3}) = 300 - (210 + 60) = 30$ which does not reach a duration of π_{min} (line 6).

$k = 5$. The break between 4 and 5 is $t_5 - (t_4 + \eta_4 + \theta_{4,5}) = 510 - (390 + 60) = 60$ which does not reach a duration of π_{min} (line 6).

$k = 6$. The break between 5 and 6 is $t_6 - (t_5 + \eta_5 + \theta_{5,6}) = 600 - (510 + 60) = 30$ which does not reach a duration of π_{min} (line 6).

To see if the break between services 3 and 4 can reach a duration of π_{min}, the method schedules Service 4 at its latest time, which results in a break with a duration of $t^l_4 - (t_3 + \eta_3 + \theta_{3,4}) = 600 - (330 + 60) = 240 \geq \pi_{min}$ (line 9). Therefore, the delay time is updated, $\bar{d} = d = 210$, and the minimum delay is set as the time needed to have a break of π_{min} between services 3 and 4:

- The time when the break starts is $\bar{t}_4 = t_3 + \eta_3 + \theta_{3,4} = 300 + 60 = 360$ (line 11).
- The starting time of Service 4, to guarantee a break duration of π_{min}, is $\bar{t}_4 + \pi_{min} = 360 + 120 = 480$. Because this starting time is later than the current schedule of Service 4, $t_4 = 390 < 480$ (line 12), a minimum delay time is needed: $\underline{d} = \bar{t}_4 + \pi_{min} - t_4 = 360 + 120 - 390 = 90$ (line 13).

The maximum and minimum delay times are shown in Figure 6.16.

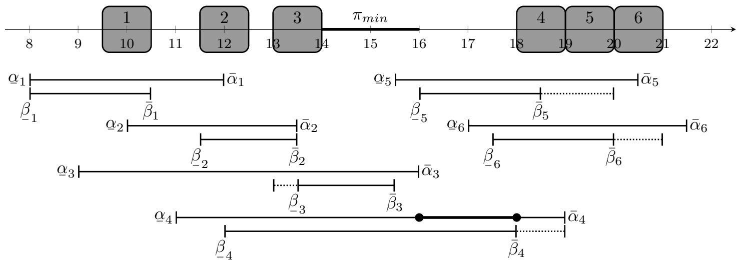


Figure 6.16: Delay to increase the break before Service 4.

6.A.12.3 Get advance time needed to reduce breaks: *advanceReduceBreak*

Algorithm 6.22 checks if the breaks can be reduced when advancing the service. First, the maximum time the service can be advanced (line 1) and the list of services affected by that advance (line 2) are obtained. After that, the algorithm iterates through the affected services (line 5) to check if there is a break between them (line 7), in which case the advance time is updated (lines 8 - 16).

Algorithm 6.22: advanceReduceBreak - Get advance time to reduce the breaks after the service

```
Data: the service ($j$), the route ($R$), the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$) and the earliest starting times ($t^e$)
//Get advance times
1  $a \leftarrow t_j - t_j^e$
2  $\hat{R} \leftarrow$ getAffectedServicesAdvance($j$, $R$, $\boldsymbol{\omega}$, $d$, $P(j,R)$) //Get services affected by advance
3  $\hat{R} \leftarrow \{j\} \cup \hat{R}$ //Add the service to the set
4  $\underline{a}, \bar{a} \leftarrow 0$
//Check if there is a break between the affected services
5  for $k \in \hat{R}$ do
6      if $k+1 \in R$ then
7          $b \leftarrow t_k - (t_{k+1} + \eta_{k+1} + \theta_{k+1,k})$
8          if $0 < b < \pi_{min}$ then
               //Advance time so service $j$ reaches $k+1$
9              $\epsilon_1 \leftarrow t_j - t_{k+1}$ //Time between $j$ and $k+1$
               //Service time and travel time between $j$ and $k+1$
10             $\epsilon_2 \leftarrow 0$
11             for $l \in \{j, ..., k\}$ do
12                 $\epsilon_2 \leftarrow \epsilon_2 + \eta_l + \theta_{l,l+1}$
13             $\epsilon_3 \leftarrow \epsilon_1 - \epsilon_2$ //Free time between services $j$ and $k+1$
14             $\bar{a} \leftarrow \epsilon_3$
15         else if $b \geq \pi_{min}$ then
16             break
17 return $\underline{a}, \bar{a}$
```

Example 6.A.13. Illustration of Algorithm 6.22.

The data, provided by Example 6.A.10, are: the service ($j = 4$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the schedule ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$) and the earliest (t^e) and latest starting times (t^l).

The maximum time that the service can be advanced is $a = t_4 - t_4^e = 390 - 240 = 150$ (line 1) and the services that will be affected by this advance are $\hat{R} = \{3, 2, 1\}$ (Example 6.A.7). Now $j = 4$ is added to the affected services, $\hat{R} = \{4, 3, 2, 1\}$, and the maximum and minimum advance times are initialized to $\underline{a}, \bar{a} = 0$ (lines 3 - 4).

After that, the algorithm checks if there is a break between the services of $\hat{R}$:

$k = 4$. The break between services 4 and 3 is $b = t_4 - (t_3 + \eta_3 + \theta_{3,4}) = 390 - (300 + 60) = 30$ (line 7). In this case, $b < \pi_{min}$ (line 8), so the advance time of 4 so it reaches 3 must be computed:

- The time between the schedules of services 4 and 3 is $\epsilon_1 = t_4 - t_3 = 390 - 300 = 90$ (line 9).
- The service and travel times between 4 and 3 is $\epsilon_2 = \eta_3 + \theta_{3,4} = 60$ (lines 9 - 12).
- The free time between services 4 and 3 is $\epsilon_3 = \epsilon_1 - \epsilon_2 = 90 - 60 = 30$ (line 13).

The advance time is set as $\bar{a} = \epsilon_3 = 30$.

$k = 3$. The break between services 3 and 2 is $b = t_3 - (t_2 + \eta_2 + \theta_{2,3}) = 300 - (210 + 60) = 30$ (line 7). In this case, $b < \pi_{min}$ (line 8), so the advance time of 4 so service 3 reaches 2 must be computed:

- The time between the schedules of services 4 and 2 is $\epsilon_1 = t_4 - t_2 = 390 - 210 = 180$ (line 9).
- The service and travel times between 4 and 2 is $\epsilon_2 = \eta_2 + \theta_{2,3} + \eta_3 + \theta_{3,4} = 120$ (lines 9 - 12).
- The free time between services 4 and 2 is $\epsilon_3 = \epsilon_1 - \epsilon_2 = 180 - 120 = 60$ (line 13).

The advance time is set as $\bar{a} = \epsilon_3 = 60$.

$k = 2$. The break between 2 and 1 is $b = t_2 - (t_1 + \eta_1 + \theta_{1,2}) = 210 - (90 + 60) = 60$ (line 7). In this case, $b < \pi_{min}$ (line 8), so the advance time of 4 so Service 2 reaches 1 must be computed:

- The time between the schedules of services 4 and 1 is $\epsilon_1 = t_4 - t_1 = 390 - 90 = 300$ (line 9).
- The service and travel times between 4 and 1 is $\epsilon_2 = \eta_1 + \theta_{1,2} + \eta_2 + \theta_{2,3} + \eta_3 + \theta_{3,4} = 180$ (lines 9 - 12).
- The free time between services 4 and 1 is $\epsilon_3 = \epsilon_1 - \epsilon_2 = 300 - 180 = 120$ (line 13).

The advance time is set as $\bar{a} = \epsilon_3 = 120$, as it can be seen in Figure 6.17.

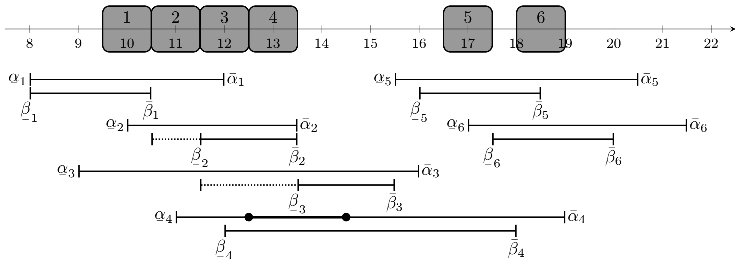


Figure 6.17: Advance to reduce breaks before Service 4.

6.A.12.4 Get advance time needed to increase the break: *advanceIncreaseBreak*

Algorithm 6.23 is used to see if the break after the service can reach a duration of π_{min} when advancing the service. After obtaining the maximum advance time (line 1), the algorithm checks if there is another break in the route with a duration of π_{min} (lines 3 - 7). If there is no other big break, the algorithm tests if setting the start of the service at its earliest time results in a break of, at least, π_{min} (lines 8 - 9). In that case, the advance time is set as the maximum one (line 10). Finally, the minimum advance time, that guarantees the break will have a duration of π_{min}, is computed (lines 11 - 13).

Algorithm 6.23: advanceIncreaseBreak - Get advance time to increase the break after the service

```
Data: the service ($j$), the route ($R$), the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$) and the earliest starting times ($t^e$)
//Get advance times
1  $d \leftarrow t_j - t^e_j$
2  $\underline{a}, \bar{a} \leftarrow 0$
//Check if there is a big break in the route
3  $b \leftarrow false$
4  for $k \in R$ do
5      if $k \neq 1$ and $k \neq j + 1$ then
6          if $t_k - (t_{k-1} + \eta_{k-1} + \theta_{k-1,k}) \geq \pi_{min}$ then
7              $b \leftarrow true$
//Check if the break after $j$ can be increased
8  if $j \neq r$ and $b = false$ then
9      if $t_{j+1} - (t^e_j + \eta_j + \theta_{j,j+1}) \geq \pi_{min}$ then
           //If the break after $j$ can cave a duration of $\pi_{min}$
10         $\bar{a} \leftarrow a$
           //Start of $j$ according to $j + 1$ to have a break of $\pi_{min}$
11         $\bar{t}_j \leftarrow t_{j+1} - \pi_{min} - \theta_{j,j+1} - \eta_j$
12         if $\bar{t}_j < t_j$ then
               //Minimum advance of $j$ to get a break of $\pi_{min}$
13             $\underline{a} \leftarrow t_j - \bar{t}_j$
14 return $\underline{a}, \bar{a}$
```

Example 6.A.14. Illustration of Algorithm 6.23.

The data, provided by Example 6.A.10, are: the service ($j = 4$), the route ($R = \{1, 2, 3, 4, 5, 6\}$), the schedule ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$, $t_6 = 600$) and the earliest (t^e) and latest starting times (t^l).

The maximum time that the service can be advanced is $a = t_4 - t^e_4 = 390 - 240 = 150$, the maximum and minimum advance times are initialized to $\underline{a}, \bar{a} = 0$ (lines 1 - 2).

Then, the algorithm checks if there is a break with duration greater then or equal to π_{min} on the route:

$k = 2$. The break between 1 and 2 is $t_2 - (t_1 + \eta_1 + \theta_{1,2}) = 210 - (90 + 60) = 60$, which does not reach a duration of π_{min} (line 6).

$k = 3$. The break between 2 and 3 is $t_3 - (t_2 + \eta_2 + \theta_{2,3}) = 300 - (210 + 60) = 30$, which does not reach a duration of π_{min} (line 6).

$k = 5$. The break between 4 and 5 is $t_5 - (t_4 + \eta_4 + \theta_{4,5}) = 510 - (390 + 60) = 60$, which does not reach a duration of π_{min} (line 6).

$k = 6$. The break between 5 and 6 is $t_6 - (t_5 + \eta_5 + \theta_{5,6}) = 600 - (510 + 60) = 30$, which does not reach a duration of π_{min} (line 6).

To check if the break after Service 4 can reach a duration of π_{min}, the algorithm schedules it at its earliest time. This results in a break with 5 with duration of $t_5 - (t^e_4 + \eta_4 + \theta_{4,5}) = 510 - (180 + 60) = 270 \geq \pi_{min}$ (line 9). Therefore, the advance time is updated to $\bar{a} = a = 150$. Finally, the minimum advance, to find a break of π_{min} between 4 and 5, is computed:

- The start of Service 4 so the break with Service 5 has a duration of π_{min} is $\bar{t}_4 = t_5 - \pi_{min} - \theta_{4,5} - \eta_4 = 510 - 120 - 60 = 330$ (line 11).
- Since this time is before the current schedule of 4, $t_4 = 390 > 330$ (line 12), a minimum advance time is needed: $\underline{a} = t_4 - \bar{t}_4 = 390 - 330 = 60$ (line 13).

The maximum and minimum advance times are shown in Figure 6.18.

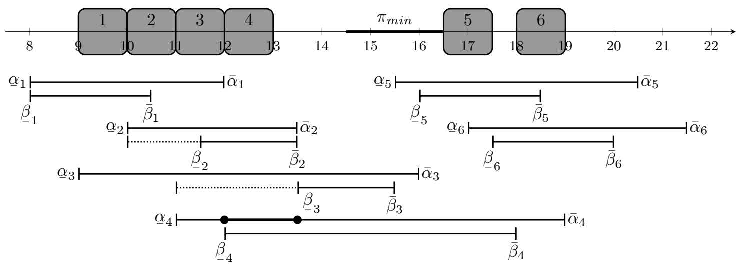


Figure 6.18: Advance to increase the break after service 4.

6.A.13 Delay the service a random amount of time: *randomDelay*

The function shown in Algorithm 6.24 delays the service. First, the number of minutes that the service will be delayed is randomly obtained following a uniform distribution (lines 2 - 5). Then, the service and its affected followers are delayed (lines 6 - 12).

Algorithm 6.24: randomDelay - Delay service j at random

```
Data: the service ($j$), the route ($R$), the solution ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$), the minimum delay time ($\underline{d}$),
      the delay time ($\bar{d}$), the maximum delay time ($d$)and the affected services ($\hat{R}$)
1  $\tilde{t} \leftarrow t$, $\tilde{x} \leftarrow x$
   //Select a delay time
2  if $\hat{R} \neq \emptyset$ and $\bar{d} > 0$ then
3      $d' \leftarrow$ chooseRandom($\underline{d}$, $\bar{d}$)
4  else
5      $d' \leftarrow$ chooseRandom($\underline{d}$, $d$)
   //Delay the service
6  $\tilde{t}_j \leftarrow t_j + d'$
7  for $k \in \hat{R}$ do
8      $\bar{t}_k \leftarrow \tilde{t}_{k-1} + \eta_{k-1} + \theta_{k-1,k}$ //Earliest start of $k$ according to the delay
9      if $\tilde{t}_k < \bar{t}_k$ then
           //If $k$ is affected by the delay
10         $\tilde{t}_k \leftarrow \bar{t}_k$
11     else
12         break
13 return $\tilde{\boldsymbol{\omega}} = (\tilde{\boldsymbol{t}}, \tilde{\boldsymbol{x}})$
```

Example 6.A.15. Illustration of Algorithm 6.24.

The data given by Example 6.2.1 are: the service ($j = 4$), the schedule ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 300$, $t_5 = 510$ and $t_6 = 600$), the minimum delay time ($\underline{d} = 0$), the updated delay time ($\bar{d} = 600$), the maximum delay time ($d = 150$) and the list of affected services ($\hat{R} = \{5, 6\}$).

First, the new schedule is initialized, $\tilde{t} = t$. Then, a random number between $\underline{d} = 0$ and $\bar{d} = 120$ is chosen, in this case, $d' = 90$ (lines 2 - 3). Finally, Service 4 is delayed, $\tilde{t}_4 = t_4 + d' = 390 + 90 = 480$, as well as its followers:

$k = 5$. The earliest start of 5 according to the new schedule of 4 is $\bar{t}_5 = \tilde{t}_4 + \eta_4 + \theta_{4,5} = 480 + 60 + 0 = 540$. Therefore, $\tilde{t}_5 = 510 < \bar{t}_5$ and its starting time is updated $\tilde{t}_5 = \bar{t}_5 = 540$ (line 10).

$k = 6$. The earliest start of 6 according to the new schedule of 5 is $\bar{t}_6 = \tilde{t}_5 + \eta_5 + \theta_{5,6} = 540 + 60 + 0 = 600$. Therefore, $\tilde{t}_6 = 600 = \bar{t}_6$ and its starting time is not updated.

Therefore, the modified schedule is $\tilde{t}_1 = 90$, $\tilde{t}_2 = 210$, $\tilde{t}_3 = 300$, $\tilde{t}_4 = 480$, $\tilde{t}_5 = 540$ and $\tilde{t}_6 = 600$.

Example 6.A.16. Illustration of Algorithm 6.24.

The data given by Example 6.2.1 are: the service ($j = 4$), the schedule ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$), the minimum delay time ($\underline{d} = 0$), the updated delay time ($\bar{d} = 90$), the maximum delay time ($d = 150$) and the list of affected services ($\hat{R} = \{5, 6\}$).

First, the new schedule is initialized, $\tilde{t} = t$. Then, a random number between $\underline{d} = 0$ and $\bar{d} = 90$ is chosen, in this case, $d' = 60$ (lines 2 - 3). Finally, Service 4 is delayed, $\tilde{t}_4 = t_4 + d' = 390 + 60 = 450$, as well as its followers:

$k = 5$. The earliest start of 5 according to the new schedule of 4 is $\bar{t}_5 = \tilde{t}_4 + \eta_4 + \theta_{4,5} = 450 + 60 + 0 = 510$. Therefore, $\tilde{t}_5 = 510 = \bar{t}_5$, its starting time is not updated and the loop terminates (lines 11 - 12).

Therefore, the modified schedule is $\tilde{t}_1 = 90$, $\tilde{t}_2 = 210$, $\tilde{t}_3 = 300$, $\tilde{t}_4 = 450$, $\tilde{t}_5 = 510$ and $\tilde{t}_6 = 600$.

Example 6.A.17. Illustration of Algorithm 6.24.

The data given by Example 6.2.1 are: the service ($j = 4$), the schedule ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$), the minimum delay time ($\underline{d} = 90$), the updated delay time ($\bar{d} = 210$), the maximum delay time ($d = 210$) and the list of affected services ($\hat{R} = \{5, 6\}$).

First, the new schedule is initialized, $\tilde{t} = t$. Then, a random number between $\underline{d} = 0$ and $\bar{d} = 210$ is chosen. In this case, $d' = 180$ (lines 2 - 3). Finally, Service 4 is delayed, $\tilde{t}_4 = t_4 + d' = 390 + 180 = 570$, as well as its followers:

$k = 5$. The earliest start of 5 according to the new schedule of 4 is $\bar{t}_5 = \tilde{t}_4 + \eta_4 + \theta_{4,5} = 570 + 60 + 0 = 630$. Therefore, $\tilde{t}_5 = 510 < \bar{t}_5$ and its starting time is updated $\tilde{t}_5 = \bar{t}_5 = 630$ (line 10).

$k = 6$. The earliest start of 6 according to the new schedule of 5 is $\bar{t}_6 = \tilde{t}_5 + \eta_5 + \theta_{5,6} = 630 + 60 + 0 = 690$. Therefore, $\tilde{t}_6 = 600 < \bar{t}_6$ and its starting time is updated $\tilde{t}_6 = \bar{t}_6 = 690$ (line 10).

Therefore, the modified schedule is $\tilde{t}_1 = 90$, $\tilde{t}_2 = 210$, $\tilde{t}_3 = 300$, $\tilde{t}_4 = 570$, $\tilde{t}_5 = 630$ and $\tilde{t}_6 = 690$.

6.A.14 Advance the service a random amount of time: *randomAdvance*

The function shown in Algorithm 6.25 is the one used to advance the service. First, a random number between 0 and the updated advance time (lines 2 - 3), or the maximum possible advance (lines 4 - 5) is obtained (following a uniform distribution). Then, the service is advanced that amount of time (line 6). Finally, if the predecessors are affected (lines 7 - 8), their schedule is advance (lines 9 - 10), in other case the algorithm terminates (lines 11 - 12).

Algorithm 6.25: randomAdvance - Advance service j a random amount of time

```
Data: the service ($j$), the route ($R$), the solution ($\boldsymbol{\omega} = (\boldsymbol{x},\boldsymbol{t})$), the minimum advance time ($\underline{a}$),
      the advance time ($\bar{a}$), the maximum advance time ($a$) and the affected services ($\hat{R}$)
1  $\tilde{t} \leftarrow t$, $\tilde{x} \leftarrow x$
   //Select advance time
2  if $\hat{R} \neq \emptyset$ and $\bar{a} > 0$ then
3      $a' \leftarrow$ chooseRandom($\underline{a}$, $\bar{a}$)
4  else
5      $a' \leftarrow$ chooseRandom($\underline{a}$, $a$)
   //Advance the service
6  $\tilde{t}_j \leftarrow t_j - a'$
7  for $k \in \hat{R}$ do
8      $\bar{t}_k \leftarrow \tilde{t}_{k+1} - \theta_{k,k+1} - \eta_k$ //Latest start of $k$ according to the advance
9      if $\tilde{t}_k < \bar{t}_k$ then
           //If $k$ is affected by the advance
10         $\tilde{t}_k \leftarrow \bar{t}_k$
11     else
12         break
13 return $\tilde{\boldsymbol{\omega}} = (\tilde{\boldsymbol{t}},\tilde{\boldsymbol{x}})$
```

Example 6.A.18. Illustration of Algorithm 6.25.

The data obtained in Example 6.2.1 are: the service ($j = 4$), the schedule ($\boldsymbol{\omega} = (\boldsymbol{x},\boldsymbol{t})$, $t_1 = 90$, $t_2 = 180$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$), the minimum advance time ($\underline{a} = 0$), the updated advance time ($\bar{a} = 0$), the maximum advance time ($a = 150$) and the list of affected services ($\hat{R} = \{3, 2, 1\}$).

First, the new schedule is initialized, $\tilde{t} = t$. Then, a random number between $\underline{a} = 0$ and $a = 150$ is chosen following a uniform distribution. In this case, $a' = 90$ (lines 4 - 5). Finally, Service 4 is advanced, $\tilde{t}_4 = t_4 - d' = 390 - 90 = 300$, as well as its predecessors:

$k = 3$. The latest start of 3 according to the new schedule of 4 is $\bar{t}_3 = \tilde{t}_4 - \eta_4 - \theta_{3,4} = 300 - 60 - 0 = 240$. Therefore, $\tilde{t}_3 = 300 > \bar{t}_3$, its starting time is updated $\tilde{t}_3 = \bar{t}_3 = 240$ (lines 9 - 10).

$k = 2$. The latest start of 2 according to the new schedule of 3 is $\bar{t}_2 = \tilde{t}_3 - \eta_2 - \theta_{2,3} = 240 - 60 - 0 = 180$. Therefore, $\tilde{t}_2 = 210 > \bar{t}_2$ and its starting time is updated $\tilde{t}_2 = \bar{t}_2 = 180$ (lines 9 - 10).

$k = 1$. The latest start of 1 according to the new schedule of 2 is $\bar{t}_1 = \tilde{t}_2 - \eta_1 - \theta_{1,2} = 180 - 60 - 0 = 120$. Therefore, $\tilde{t}_1 = 90 < \bar{t}_1$ and its starting time is not updated (lines 11 - 12).

Thus, the modified schedule is $\tilde{t}_1 = 90$, $\tilde{t}_2 = 180$, $\tilde{t}_3 = 240$, $\tilde{t}_4 = 300$, $\tilde{t}_5 = 510$ and $\tilde{t}_6 = 600$.

Example 6.A.19. Illustration of Algorithm 6.25.

The data obtained in Example 6.2.1 are: the service ($j = 4$), the schedule ($\boldsymbol{\omega} = (\boldsymbol{x},\boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$), the minimum advance time ($\underline{a} = 0$), the updated advance time ($\bar{a} = 120$), the maximum advance time ($a = 150$) and the list of affected services ($\hat{R} = \{3, 2, 1\}$).

First, the new schedule is initialized, $\tilde{t} = t$. Then, a random number between $\underline{a} = 0$ and $\bar{a} = 120$ is chosen according to a uniform distribution. In this case, $a' = 30$ (lines 4 - 5). Finally, Service 4 is advanced, $\tilde{t}_4 = t_4 - d' = 390 - 30 = 360$, as well as its predecessors:

$k = 3$. The latest start of 3 according to the new schedule of 4 is $\bar{t}_3 = \tilde{t}_4 - \eta_3 - \theta_{3,4} = 360 - 60 - 0 = 300$. Therefore, $\tilde{t}_3 = 300 = \bar{t}_3$, its starting time is not updated and the loop terminates (lines 11 - 12).

Thus, the modified schedule is $\tilde{t}_1 = 90$, $\tilde{t}_2 = 210$, $\tilde{t}_3 = 300$, $\tilde{t}_4 = 360$, $\tilde{t}_5 = 510$ and $\tilde{t}_6 = 600$.

Example 6.A.20. Illustration of Algorithm 6.25.

The data obtained in Example 6.2.1 are: the service ($j = 4$), the schedule ($\boldsymbol{\omega} = (\boldsymbol{x}, \boldsymbol{t})$, $t_1 = 90$, $t_2 = 210$, $t_3 = 300$, $t_4 = 390$, $t_5 = 510$ and $t_6 = 600$), the minimum advance time ($\underline{a} = 60$), the updated advance time ($\bar{a} = 150$), the maximum advance time ($a = 150$) and the list of affected services ($\hat{R} = \{3, 2, 1\}$).

First. the new schedule is initialized, $\tilde{t} = t$. Then, a random number between $\underline{a} = 60$ and $\bar{a} = 150$ is chosen following a uniform distribution. In this case, $a' = 120$ (lines 4 - 5). Finally, Service 4 is advanced, $\tilde{t}_4 = t_4 - d' = 390 - 120 = 270$, as well as its predecessors:

$k = 3$. The latest start of 3 according to the new schedule of 4 is $\bar{t}_3 = \tilde{t}_4 - \eta_3 - \theta_{3,4} = 270 - 60 - 0 = 210$. Therefore, $\tilde{t}_3 = 300 > \bar{t}_3$ and its starting time is updated $\tilde{t}_3 = \bar{t}_3 = 210$ (lines 9 - 10).

$k = 2$. The latest start of 2 according to the new schedule of 3 is $\bar{t}_2 = \tilde{t}_3 - \eta_2 - \theta_{2,3} = 210 - 60 - 0 = 150$. Therefore, $\tilde{t}_2 = 210 > \bar{t}_2$ and its starting time is updated $\tilde{t}_2 = \bar{t}_2 = 150$ (lines 9 - 10).

$k = 1$. The latest start of 1 according to the new schedule of 2 is $\bar{t}_1 = \tilde{t}_2 - \eta_1 - \theta_{1,2} = 150 - 60 - 0 = 90$. Therefore, $\tilde{t}_1 = 90 = \bar{t}_1$ and its starting time is not updated (lines 11 - 12).

Thus, the modified schedule is $\tilde{t}_1 = 90$, $\tilde{t}_2 = 150$, $\tilde{t}_3 = 210$, $\tilde{t}_4 = 270$, $\tilde{t}_5 = 510$ and $\tilde{t}_6 = 600$.

Chapter 7

Computational results

This chapter presents the computational study to analyze the performance of the methods described in Chapters 3, 4, 5 and 6.

The metaheuristic algorithms and the AUGMECON2 method have been implemented in Python 3.7 (Van Rossum & Drake (2009)), whereas the MILP problem has been solved with Gurobi 9.1.1 (Gurobi Optimization, LLC (2021)) via its Python interface. All the experiments were run in an Intel(R) Xeon(R) Gold 6146 CPU 3.20GHz processor, with 16GB of RAM, 2 cores and 100GB of HDD, located in Centre for Information and Communications Technology Research (CITIC).

The chapter is structured as follows. Section 7.1 describes the data used to evaluate the algorithms developed to tackle the HCSP. In Section 7.2 the results related to the metaheuristic approach that prioritizes the welfare over the cost are presented (ALNS_WC). In a similar manner, Section 7.3 presents the results of the approach that prioritizes the cost over the welfare (ALNS_CW). Finally, Section 7.5 shows the results of the biobjective version of the problem (BIALNS).

7.1 Data

The performance of the methods will be tested with two different datasets: the Solomon instances and real data provided by the company. The Solomon instances are used to compare the performance of the algorithms with respect to the resolution of the MILP formulation, but with the main focus of analyzing the parameters of the metaheuristic algorithms. On the other hand, the real instances are used to compare the solutions given by the algorithms with the real schedules employed by the company.

7.1.1 Solomon instances

The instances considered in this computational study are based on the ones presented in Solomon (1987), which are 100-service euclidean problems. The characteristics provided in those instances are: services to carry out, locations, duration, hard time windows, caregivers availability and total working times. Furthermore, some additional data required by the HCSP were randomly generated: services-caregivers affinity levels and soft time windows of the services.

The Solomon instances are classified into three groups, depending on the geographical distribution of the services: clustered, random and a combination of both. Apart from that, instances can have services with wide hard time windows and large caregiver working time or tight hard time windows and small caregiver working time. Therefore, the combination of these characteristics has made the dataset to be one of the most used in the literature.

Table 7.1 contains a basic description of 5 types of Solomon instances, in terms of the number of services (S) and the number of caregivers (N). For each type, 10 different instances were randomly generated, specifying: minimum and maximum duration of the services, minimum and maximum length of the hard time windows (htw) and soft time windows (stw), minimum and maximum travel times between the services and size of the problems in terms of the MILP formulation (number of variables $\times$ number of constraints). In addition, the affinity levels were randomly set to 3 or 5.

S	N	duration		length of htw		length of stw		travel time		size (vars $\times$ constr)
		min	max	min	max	min	max	min	max	
10	2	10	90	65	3380	11	93	1	75	496×950
15	2	10	90	83	1343	12	93	1	89	1036×2010
25	4	10	90	53	896	12	93	1	84	5382×10560
50	6	10	90	49	1226	11	93	1	96	31048×61490
100	14	10	90	20	1225	12	93	1	101	284512×566210

Table 7.1: Data instances.

7.1.2 Real Data

Table 7.2 shows the data of 15 real instances, taken by the schedule of the company during consecutive weeks between years 2016 and 2017. For these instances, the number of services vary from 633 to 950 and their duration is between 30 and 180 minutes. The available caregivers for each week vary from 35 to 39 and the affinity levels between users and caregivers are 3, 4 or 5.

week	S	N	duration		length of htw		length of stw		travel time		size (vars $\times$ constr)
			min	max	min	max	min	max	min	max	
1	865	39	30	168	90	630	60	450	0	22	$409242965 \times 817811191$
2	880	38	30	168	90	630	60	450	0	22	$412686700 \times 824704562$
3	895	35	30	168	90	630	60	450	0	22	$393163615 \times 785700695$
4	863	37	30	168	90	630	60	450	0	22	$386464469 \times 772290281$
5	633	38	30	160	90	630	60	450	0	22	$213674848 \times 426868578$
6	822	37	30	160	90	630	60	450	0	22	$350646500 \times 700684683$
7	894	36	30	160	90	630	60	450	0	22	$403494396 \times 806345076$
8	760	39	30	168	90	630	60	450	0	22	$315995510 \times 631398181$
9	808	36	30	168	90	630	60	450	0	22	$329657720 \times 658733644$
10	911	37	30	160	90	630	60	450	0	22	$430610597 \times 860547017$
11	870	39	30	160	90	630	60	450	0	22	$413983620 \times 827288601$
12	840	39	30	180	90	630	60	450	0	22	$385949190 \times 771243141$
13	950	38	30	180	90	630	60	480	0	22	$480891900 \times 961061762$
14	925	38	30	180	90	630	60	480	0	22	$455934400 \times 911165762$
15	939	38	30	180	90	630	60	480	0	22	$469828672 \times 938943666$

Table 7.2: Real data instances.

7.2 Hierarchical approach: welfare-cost

This section deals with the computational study to evaluate the good performance of the method which combines the ALNS metaheuristic for the routes with the heuristic for the schedules described in Section 4.1, denoted by ALNS_WC.

7.2.1 Gurobi results

The MILP has been solved using Gurobi with a time limit of 12 hours. The results are presented in Table 7.3. Note that, although Gurobi finds a feasible solution for 80% of the instances, it only ensures the global optimality ("-" in the gap column means that it closes it) in 67.5% of them.

instance	feasible	opt	gap	secs	instance	feasible	opt	gap	secs
10_01	✓	✓	-	0.56	10_06	✓	✓	-	0.34
10_02	✓	✓	-	0.60	10_07	✓	✓	-	0.58
10_03	✓	✓	-	0.77	10_08	✓	✓	-	0.64
10_04	✓	✓	-	63.93	10_09	✓	✓	-	1.24
10_05	✓	✓	-	4.48	10_10	✓	✓	-	160.16
15_01	✓	✓	-	0.22	15_06	✓	✓	-	0.67
15_02	✓	✓	-	0.27	15_07	✓	✓	-	2169.01
15_03	✓	✓	-	0.28	15_08	✓	✓	-	14.92
15_04	✓	✓	-	0.20	15_09	✓	✓	-	12.69
15_05	✓	✓	-	0.12	15_10	✓	✓	-	21.92
25_01	✓	✓	-	41.37	25_06	✓	✓	-	29.84
25_02	✓	✓	-	2009.30	25_07	✓	?	1.76	limit
25_03	✓	✓	-	37.02	25_08	-	-	-	-
25_04	✓	?	2.915	limit	25_09	✓	✓	-	28.26
25_05	-	-	-	-	25_10	✓	✓	-	158.94
50_01	-	-	-	-	50_06	✓	?	2.80	limit
50_02	✓	?	49.05	limit	50_07	-	-	-	-
50_03	✓	✓	-	410.92	50_08	✓	?	0.008	limit
50_04	✓	?	0.01	limit	50_09	✓	?	0.005	limit
50_05	✓	?	0.01	limit	50_10	✓	?	0.01	limit
100_01	✓	?	0.008	limit	100_06	-	-	-	-
100_02	-	-	-	-	100_07	-	-	-	-
100_03	✓	?	346.04	limit	100_08	-	-	-	-
100_04	-	-	-	-	100_09	-	-	-	-
100_05	✓	?	269.05	limit	100_10	✓	?	0.003	limit

Table 7.3: Gurobi computational results (prioritizing welfare over cost).

7.2.2 ALNS_WC results

Regarding the configuration of the ALNS_WC algorithm, the parameters considered are the number of iterations (50, 100, 150, 200, 250, 500, 750 and 1000) and the proportion of solution to be destroyed, from now on p, (25%, 50%, 75% and 100%). Further, an automatic version that dynamically updates this parameter is employed, denoted as *auto_p*, which is a method that starts with a proportion of p and decreases after each iteration of the ALNS (*auto*_25%, *auto*_50%, *auto*_75% and *auto*_100%). Each combination of parameters has been run 5 times with a time limit of 1 hour.

7.2.2.1 Gurobi vs ALNS_WC

Let x and x' be two solutions, the Relative Percentage Deviation (RPD) of x from x', is defined as

$$RPD(x, x') = \frac{f(x) - f(x')}{f(x')} \times 100.$$

In this case, the RPD is used to obtain the mean deviation between the solutions given by the algorithms and the ones obtained using the Gurobi solver.

Tables 7.4 and 7.5 present, for the instances with 10 and 15 services, the mean RPD with respect to the first and second objective, respectively. This means that Table 7.4 shows the RPD value for the first objective and, if this value is 0 (for an instance), the RPD value for the second objective is shown in Table 7.5. In this case, the algorithm was run for 1000 iterations, considering the different values of p.

According to the results, the behavior of the ALNS_WC in small instances is better for the configurations with a larger value of p (75%, *auto*_75%, 100% and *auto*_100%). Therefore, it can be deduced that in small instances it is advisable to focus on major changes in the solution that allow the solution space to be adequately explored. In fact, for p = *auto*_100%, the RPD shows that the algorithm finds the same solutions as Gurobi for all instances.

	25%	50%	75%	100%	*auto*_25%	*auto*_50%	*auto*_75%	*auto*_100%
10_01	0	0	0	0	0	0	0	0
10_02	0	0	0	0	0	0	0	0
10_03	0	0	0	0	0	0	0	0
10_04	9.61e-4	0	0	0	0	1.18e-3	0	0
10_05	5.68e-3	0	0	0	0	0	0	0
10_06	0	0	0	0	0	0	0	0
10_07	0	0	0	0	0	0	0	0
10_08	0	0	0	0	0	0	0	0
10_09	0	0	0	0	0	0	0	0
10_10	2.19e-3	2.19e-3	2.19e-3	0	1.66e-2	0	0	0
15_01	0	0	0	0	0	0	0	0
15_02	0	0	0	0	0	0	0	0
15_03	0	0	0	0	0	0	0	0
15_04	0	0	0	0	0	0	0	0
15_05	0	0	0	0	0	0	0	0
15_06	0	0	0	0	0	0	0	0
15_07	0	0	0	0	0	0	0	0
15_08	7.42e-4	0	4.94e-4	0	7.42e-4	9.89e-4	2.47e-4	0
15_09	0	0	0	0	4.34e-5	0	0	0
15_10	0	0	0	0	0	0	0	0

Table 7.4: Mean RPD values comparing ALNS_WC with Gurobi (first objective).

	25%	50%	75%	100%	*auto*_25%	*auto*_50%	*auto*_75%	*auto*_100%
10_01	0	0	0	0	0	0	0	0
10_02	4.94e-2	0	0	0	1.48e-1	4.94e-2	0	0
10_03	0	0	0	0	0	0	0	0
10_04	-	0	0	0	0	-	0	0
10_05	-	0	0	0	0	0	0	0
10_06	0	0	0	0	0	0	0	0
10_07	0	0	0	0	0	0	0	0
10_08	2.08	0	0	0	1.39	0	0	0
10_09	3.98e-1	0	0	0	0	0	0	0
10_10	-	-	-	0	-	0	0	0
15_01	0	0	0	0	0	0	0	0
15_02	0	0	0	0	0	0	0	0
15_03	0	0	0	0	0	0	0	0
15_04	0	0	0	0	0	0	0	0
15_05	0	0	0	0	0	0	0	0
15_06	0	0	0	0	0	0	0	0
15_07	0	0	0	0	0	0	0	0
15_08	-	0	-	0	-	-	-	0
15_09	0	0	0	0	-	0	0	0
15_10	0	0	0	0	0	0	0	0

Table 7.5: Mean RPD values comparing ALNS_WC with Gurobi (second objective).

	25%	50%	75%	100%	*auto*_25%	*auto*_50%	*auto*_75%	*auto*_100%
10_01	0	0	0	0	0	0	0	0
10_02	0	0	0	0	0	0	0	0
10_03	0	0	0	0	0	0	0	0
10_04	9.61e-4	0	0	0	0	1.18e-3	0	0
10_05	5.68e-3	0	0	0	0	0	0	0
10_06	0	0	0	0	0	0	0	0
10_07	0	0	0	0	0	0	0	0
10_08	0	0	0	0	0	0	0	0
10_09	0	0	0	0	0	0	0	0
10_10	2.19e-3	2.19e-3	2.19e-3	0	1.66e-2	0	0	0
15_01	0	0	0	0	0	0	0	0
15_02	0	0	0	0	0	0	0	0
15_03	0	0	0	0	0	0	0	0
15_04	0	0	0	0	0	0	0	0
15_05	0	0	0	0	0	0	0	0
15_06	0	0	0	0	0	0	0	0
15_07	0	0	0	0	0	0	0	0
15_08	7.42e-4	0	4.94e-4	0	7.42e-4	9.89e-4	2.47e-4	0
15_09	0	0	0	0	4.34e-5	0	0	0
15_10	0	0	0	0	0	0	0	0
25_01	2.83e-3	0	0	0	3.92e-3	0	0	0
25_02	8.34e-3	6.45e-3	6.45e-3	6.45e-3	3.02e-3	9.09e-4	7.27e-4	6.45e-3
25_03	2.15	0	0	0	0	0	0	0
25_04	4.67	3.79	5.19	2.58	4.83	3.11	1.72	2.58
25_05	9.89e-1	6.57e-1	6.74e-1	6.52e-1	3.58	1.96	3.35e-1	6.52e-1
25_06	1.22e-3	9.32e-4	7.29e-4	8.13e-4	6.44e-4	0	0	8.13e-4
25_07	1.29	3.24e-1	9.72e-1	6.43e-1	5.82	2.57	1.29	6.43e-1
25_08	1.33	1.97	1.32	1.63	1.96	1.96	1.64	1.63
25_09	2.03e-5	0	0	0	4.07e-5	2.03e-5	2.03e-5	0
25_10	3.69e-4	0	0	0	0	1.62e-3	0	0
50_01	2.89e-3	3.16e-3	4.66e-3	5.28e-3	6.49e-4	8.59e-4	6.17e-4	2.22e-3
50_02	6.52e-1	6.62e-1	5.26e-1	1.47	7.68e-3	1.68e-1	8.07e-1	3.45e-1
50_03	0	0	0	0	0	0	0	0
50_04	6.07e-4	1.01e-3	8.04e-4	1.44e-3	1.09e-3	1.17e-3	1.49e-4	2.51e-4
50_05	4.91e-4	4.95e-4	7.70e-4	1.16e-3	7.70e-4	3.97e-4	2.08e-4	4.54e-4
50_06	6.72e-5	2.69e-5	2.02e-4	3.97e-4	6.72e-5	0	2.02e-5	6.72e-6
50_07	1.61e-3	8.10e-4	6.29e-4	1.40e-3	8.02e-4	1.03e-3	3.62e-4	8.02e-4
50_08	3.22e-4	1.82e-4	4.41e-4	1.12e-3	1.18e-3	7.84e-4	3.08e-4	2.66e-4
50_09	4.03e-4	9.34e-4	1.84e-3	3.15e-3	1.67e-3	9.41e-4	7.40e-4	8.87e-4
50_10	9.63e-4	1.06e-3	2.30e-3	3.59e-3	1.53e-3	2.53e-4	6.82e-4	3.67e-4
100_01	8.91e-1	1.29	1.93	2.98	4.06e-1	6.43e-1	1.21	1.93
100_02	8.33e-2	9.70e-1	1.65	2.21	4.82e-1	5.65e-1	1.13	1.81
100_03	0	0	0	0	0	0	0	0
100_04	2.57	8.46	7.30	8.08	7.76e-1	3.73	5.44	5.56
100_05	0	0	0	0	0	0	0	0
100_06	1.16e-4	2.44e-4	-	-	8.93e-5	2.30e-4	2.72e-4	1.65e-4
100_07	3.22e-4	5.41e-4	-	-	3.19e-4	3.21e-4	4.23e-4	3.75e-4
100_08	2.30e-5	2.01e-5	-	-	1.04e-5	0	2.76e-5	3.45e-5
100_09	2.62e-4	-	-	-	1.13e-4	4.28e-4	4.09e-4	3.40e-4
100_10	6.78e-6	0	2.37e-5	5.08e-5	0	0	0	0

Table 7.6: RPD values comparing ALNS_WC with the best solution found.

Considering all the instances, it is important to highlight that the ALNS_WC is able to find solutions of the same quality as Gurobi. Therefore, to explore the stability of the algorithm in each instance, the relative deviation of the solutions of the ALNS_WC with respect to the best solution (that can be found by Gurobi or by the algorithm) is studied. The mean RPD value (first objective) for all instances, comparing the solutions obtained by the algorithm with the best solution found, is shown in Table 7.6.

The best RPD values are obtained using different configurations of p, depending on the size of the instance considered. The results reported in Table 7.6 show that the ALNS_WC is very stable for all the instances considered. For instances with 10 and 15 services, big values of p report the best solution. When the number of services increases, the configurations with smaller values of p are the most suitable. This trend is clearly reflected in the instances with 100 services, where $p \in \{25\%, auto\_25\%\}$ leads to acceptable solutions in all cases. However, for some of these instances (4 out of 10), the configurations with $p \in \{75\%, 100\%\}$ do not even provide solutions in the set time.

7.2.2.2 Parameter analysis of ALNS_WC

Figures 7.1, 7.2 and 7.3 present the evolution of the success rate (the proportion of times where the best solution is found) and the computational time (in seconds) along the iterations.

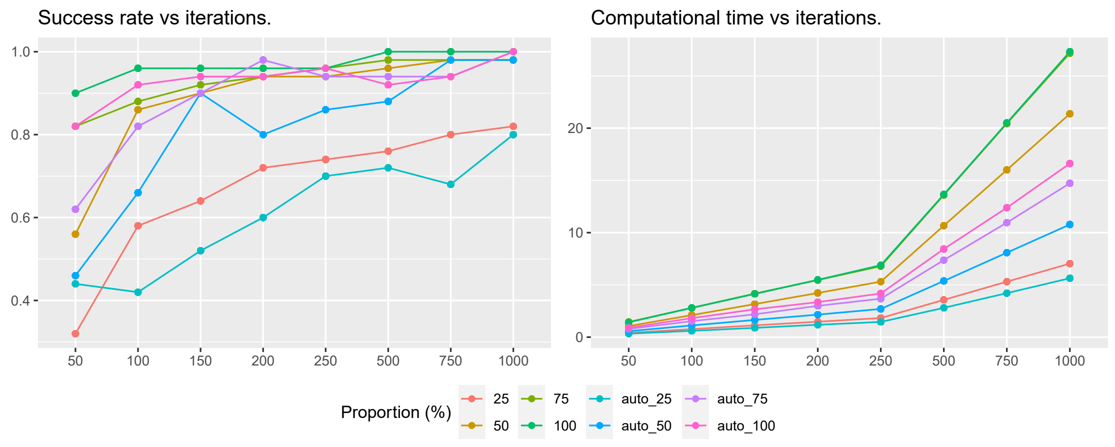


Figure 7.1: Computational results for instances with 10 services (ALNS_WC).

Figure 7.1 is related to instances with 10 services, where configurations of $p \in \{auto\_75\%, auto\_100\%\}$ present really good behavior according to the success rate and the computational time. It can be seen that $p = 100\%$ also shows a good performance in terms of the success rate, but its computational time is worse than the ones of $p \in \{auto\_75\%, auto\_100\%\}$. It is clear that, for these instances, the worst configurations are the ones of $p \in \{25\%, auto\_25\%\}$.

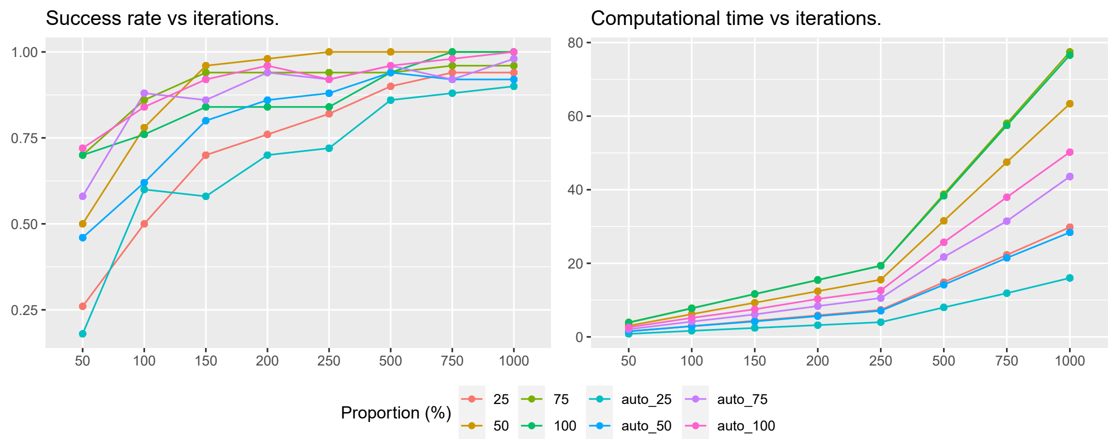


Figure 7.2: Computational results for instances with 15 services (ALNS_WC).

With regard to instances with 15 services, Figure 7.2 shows that any $p \in \{50\%, 100\%, auto\_100\%\}$ is a good choice according to the success rate, although $p = auto\_100\%$ is faster. Figure 7.3 indicates that, for instances with 25 services, $p = auto\_75\%$ outperforms the other configurations of p in terms of success rate and computational time.

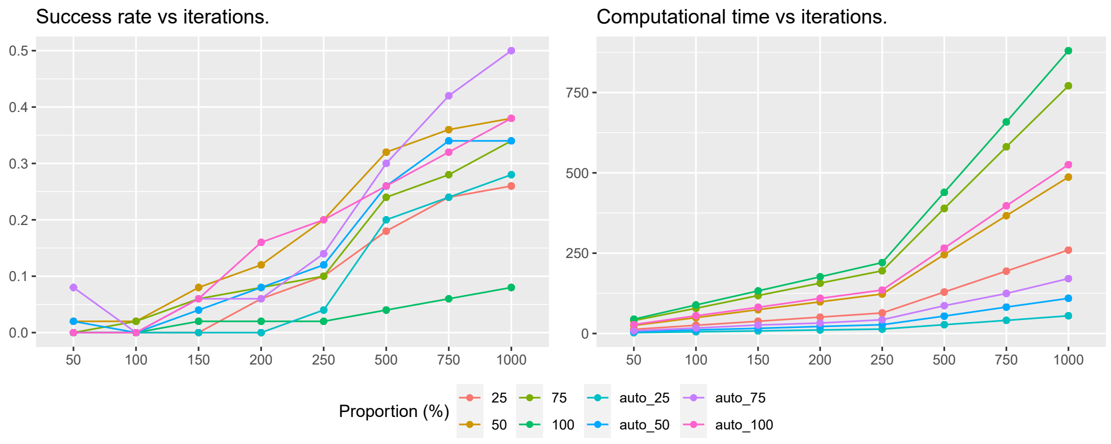


Figure 7.3: Computational results for instances with 25 services (ALNS_WC).

Figures 7.4 and 7.5 evaluate the mean RPD and computational time along the iterations for instances with 50 and 100 services.

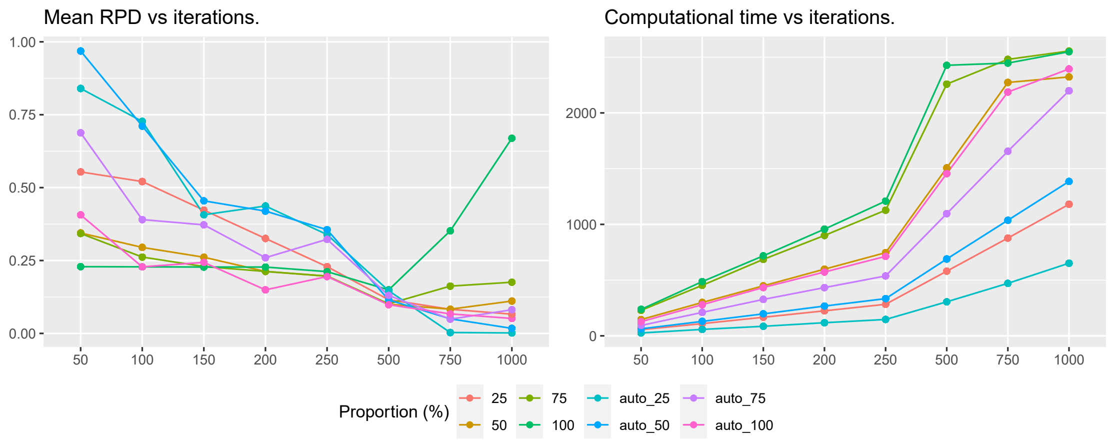


Figure 7.4: Computational results for instances with 50 services (ALNS_WC).

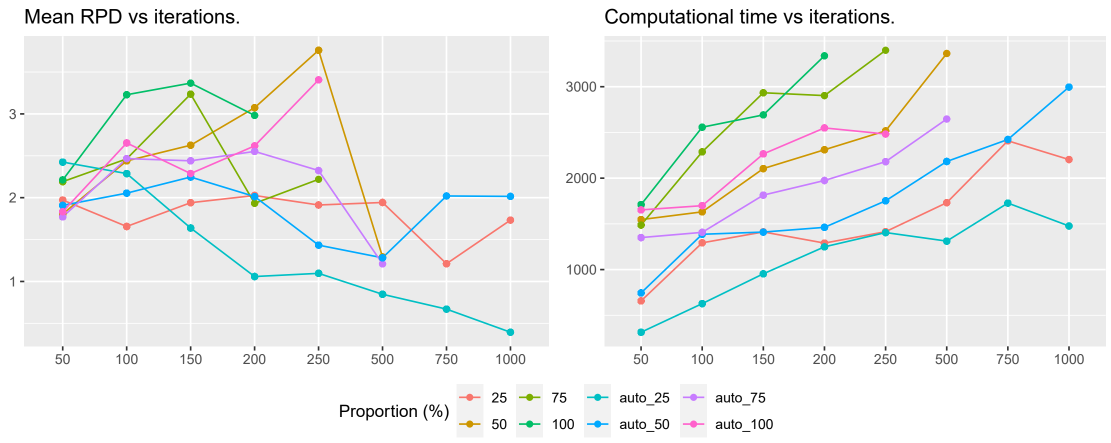


Figure 7.5: Computational results for instances with 100 services (ALNS_WC).

As far as instances with 50 services, $p = auto\_25\%$ is the best configuration in terms of the mean RPD and computational time (see Figure 7.4). For instances with 100 services (see Figure 7.5) only three configurations, any $p \in \{25\%, auto\_25\%, auto\_50\%\}$, are able to solve the instances for 750 and 1000 iterations in the given time limit. From these configurations, the best one (according to the mean RPD and computational time) is $p = auto\_25\%$.

Note that, as the number of services increases, it is desirable to reduce the initial value of the proportion in the automatic configurations. In this way, the lowest initial values for *auto* behave better in the largest instances, obtaining fast computational times.

7.2.3 Constraint programming results

The Constraint Programming method and the heuristic algorithm presented in Chapter 4 (Algorithm 4.1) are used to find the best schedule once a route (a set of ordered services assigned to a caregiver) is given, prioritizing the welfare over the cost. Although Algorithm 4.1 does not guarantee that an optimal solution will be obtained in all cases, the computational experiments showed that high quality solutions can be found in a short computational time. Therefore, to evaluate the behavior of the heuristic algorithm, the two methods are used to obtain the schedules of the same set of routes. Apart from that, in order to compare the performance of the ALNS with CP, the same computational experiments than for ALNS_WC (with the same configuration of parameters) are run. Regarding the implementation, the resulting CP scheduling problems are solved using the CPSAT solver from the Python interface of the Google OR-Tools (see Perron & Furnon (2022)). From now on, the version of the ALNS algorithm with Constraint Programming is denoted as ALNS_CPSAT.

7.2.3.1 Heuristic scheduling algorithm vs CPSAT

To carry out the comparison between Algorithm 4.1 and CP a set of 3000000 routes was randomly generated, out of which 1108371 were feasible (in terms of hard time windows of services and caregivers). For each feasible route, its schedule was obtained using Constraint Programming (CPSAT) and Algorithm 4.1. In 99.995% of the routes the algorithm finds the optimal schedule whereas CPSAT always achieves optimality. This means that the solution given by the heuristic algorithm was not the optimal in only 54 cases.

Figure 7.6 shows a boxplot of the differences between objective values (soft time window penalization and cost) of the schedules found by the algorithm and CPSAT. It can be seen that the most common scenario is that the objective values are equal, with only a few outliers where there are differences between the values obtained by the algorithm and those provided by CPSAT. Comparing the two objectives, the differences in the penalization are smaller than those of the cost, which makes sense as a hierarchical approach that prioritizes the welfare (soft time window penalization when working with only routes) over the cost is being considered.

In terms of computational time, the mean time the heuristic algorithm needs to obtain the schedule of a route is 0.00457 seconds, meanwhile for CPSAT is 0.03358 seconds. Therefore, it can be concluded that the heuristic algorithm, despite not being an exact method, obtains very good results in short computational times.

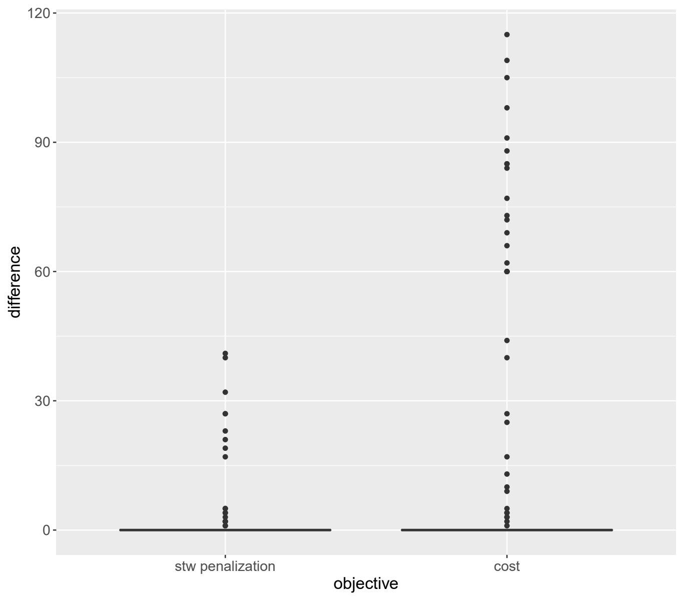


Figure 7.6: Objective function value differences (CPSAT vs Algorithm 4.1).

7.2.3.2 Gurobi vs ALNS_CPSAT

The results of ALNS_CPSAT have been compared with the solutions given by Gurobi over the Solomon instances of 10 and 15 services. Tables 7.7 and 7.8 show the mean RPD with respect to the first and second objectives, respectively. As it can be seen, the performance of the ALNS_CPSAT is similar to the one obtained for ALNS_WC (see Tables 7.4 and 7.4). For this set of instances, the ALNS_CPSAT behaves better for the configurations with a larger value of p, and for each of the instances there exist several configurations of the parameters finding a better than or equal to Gurobi solution.

	25%	50%	75%	100%	*auto*_25%	*auto*_50%	*auto*_75%	*auto*_100%
10_01	0	0	0	0	0	0	0	0
10_02	0	0	0	0	0	0	0	0
10_03	0	0	0	0	0	0	0	0
10_04	0	0	0	0	4.0e-4	0	0	0
10_05	0	0	0	0	0	0	0	0
10_06	0	0	0	0	0	0	0	0
10_07	0	0	0	0	0	0	0	0
10_08	0	0	0	0	0	0	0	0
10_09	0	0	0	0	0	0	0	0
10_10	0	4.0e-3	2.0e-3	0	0	4.0e-3	0	0
15_01	0	0	0	0	0	0	0	0
15_02	0	0	0	0	0	0	0	0
15_03	0	0	0	0	0	0	0	0
15_04	0	0	0	0	0	0	0	0
15_05	0	0	0	0	0	0	0	0
15_06	0	0	0	0	0	0	0	0
15_07	0	0	0	0	0	0	0	0
15_08	2.0e-4	2.0e-4	0	0	2.0e-4	2.0e-4	2.0e-4	0
15_09	4.3e-5	0	0	0	0	0	0	0
15_10	0	0	0	0	0	0	0	0

Table 7.7: Mean RPD values comparing ALNS_CPSAT with Gurobi (first objective).

	25%	50%	75%	100%	*auto*_25%	*auto*_50%	*auto*_75%	*auto*_100%
10_01	0	0	0	0	0	0	0	0
10_02	0.09	0	0	0	0	0	0	0
10_03	0	0	0	0	0	0	0	0
10_04	0	0	0	0	-	0	0	0
10_05	0	0	0	0	0	0	0	0
10_06	0	0	0	0	0	0	0	0
10_07	0	0	0	0	0	0	0	0
10_08	0.69	0	0	0	0.69	0	0	0
10_09	0	0	0	0	0.39	0	0	0
10_10	0	-	-	0	0	-	0	0
15_01	0	0	0	0	0	0	0	0
15_02	0	0	0	0	0	0	0	0
15_03	0	0	0	0	0	0	0	0
15_04	0	0	0	0	0	0	0	0
15_05	0	0	0	0	0	0	0	0
15_06	0	0	0	0	0	0	0	0
15_07	0	0	0	0	0	0	0	0
15_08	-	-	0	0	-	-	-	0
15_09	-	0	0	0	0	0	0	0
15_10	0	0	0	0	0	0	0	0

Table 7.8: Mean RPD values comparing ALNS_CPSAT with Gurobi (second objective).

7.2.3.3 ALNS_WC vs ALNS_CPSAT

It is also interesting to compare the ALNS_CPSAT with the ALNS_WC. To this aim, for each instance (and for each configuration of the parameters), the worst solutions (in terms of the first objective function) obtained by both algorithms are compared[1].

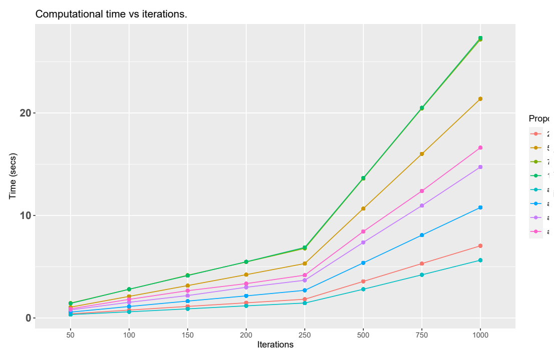
	25%	50%	75%	100%	*auto*_25%	*auto*_50%	*auto*_75%	*auto*_100%
10_01	0	0	0	0	0	0	0	0
10_02	0	0	0	0	0	0	0	0
10_03	0	0	0	0	0	0	0	0
10_04	2.4e-5	0	0	0	1.1e-5	0	0	0
10_05	2.8e-4	0	0	0	0	0	0	0
10_06	0	0	0	0	0	0	0	0
10_07	0	0	0	0	0	0	0	0
10_08	0	0	0	0	0	0	0	0
10_09	0	0	0	0	0	0	0	0
10_10	1.1e-4	0	0	0	2.8e-4	-1.0e-4	0	0
15_01	0	0	0	0	0	0	0	0
15_02	0	0	0	0	0	0	0	0
15_03	0	0	0	0	0	0	0	0
15_04	0	0	0	0	0	0	0	0
15_05	0	0	0	0	0	0	0	0
15_06	0	0	0	0	0	0	0	0
15_07	0	0	0	0	0	0	0	0
15_08	0	-1.2e-5	1.2e-5	0	0	0	0	0
15_09	-2.1e-6	0	0	0	2.1e-6	0	0	0
15_10	0	0	0	0	0	0	0	0

Table 7.9: Comparing the worst solution obtained by ALNS_CPSAT and ALNS_WC.

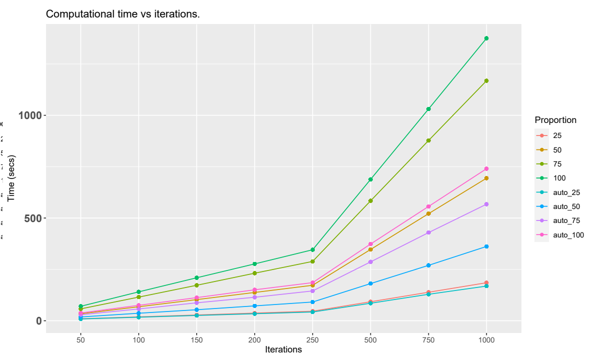
[1]Notice that the worst solutions are being compared because the best solutions found by the two methods are the same for all configurations.

The results are presented in Table 7.9, where it can be seen that both algorithms behave similarly[2], with only really small differences probably due to the randomness inside the removal/insertion operators.

Figures 7.7 and 7.8 show the computational time of ALNS_WC (on the left) and ALNS_CPSAT (on the right) for the instances with 10 and 15 services, respectively. Although the trend of the computational time over the iterations is similar for both algorithms (for the different configurations of parameter p), it is clear that ALNS_WC is much more faster than ALNS_CPSAT[3].

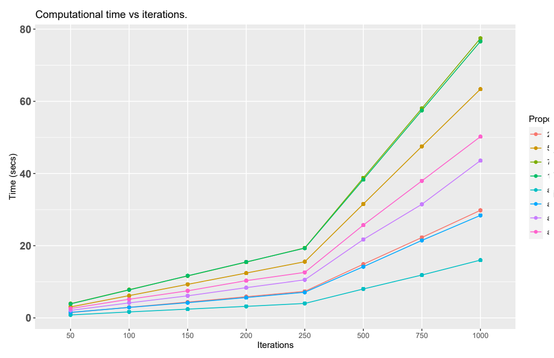


(a) ALNS_WC computational time.

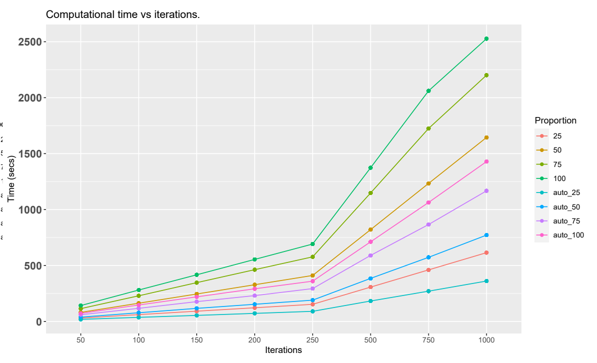


(b) ALNS_CPSAT computational time.

Figure 7.7: Computational time for instances with 10 services.



(a) ALNS_WC computational time.



(b) ALNS_CPSAT computational time.

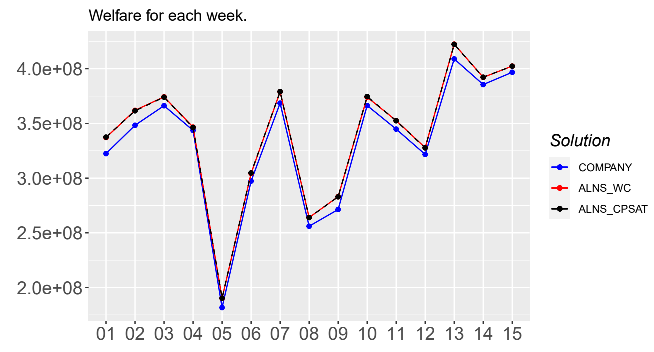
Figure 7.8: Computational time for instances with 15 services.

Finally, the ALNS_CPSAT was also used to solve the real case study. Figure 7.9 shows a comparison of the solutions obtained by the different approaches. Again, the solution of ALNS_WC and ALNS_CPSAT are really similar, finding better results than those employed by the company. Further, the ALNS_CPSAT is more time consuming than ALNS_WC (see Figure 7.10).

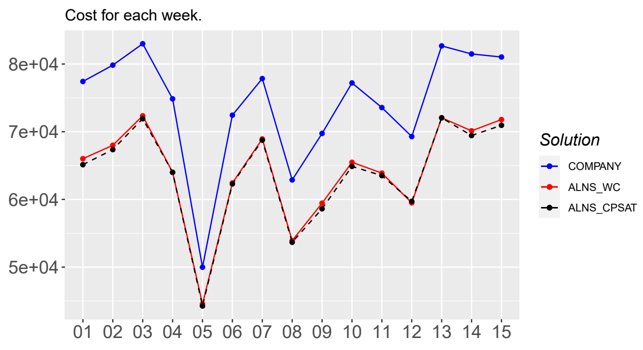
To conclude, it can be said that ALNS_WC and ALNS_CPSAT present a similar behavior in terms of the objective function. However, ALNS_WC is clearly much more faster than ALNS_CPSAT. Note that these results highlight the good performance of the algorithm described in Section 4.1 when computing the schedules inside the insertion operators, which is more efficient than solving the scheduling problems with a CP solver.

[2]Note that the table shows the relative difference of the worst solution found by ALNS_CPSAT minus the worst solution of ALNS_WC. Therefore, positive values mean that the ALNS_CPSAT solution is worse than the ALNS_WC one.

[3]Note the different scales in both figures.



(a) Welfare for each week.



(b) Cost for each week.

Figure 7.9: Computational results (objective functions) for real case study.

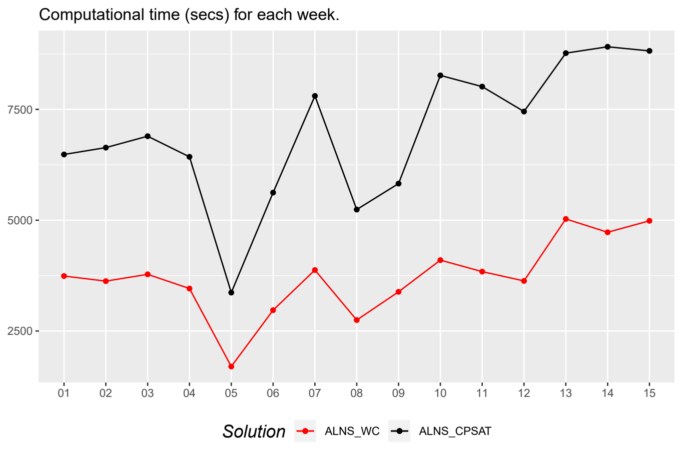


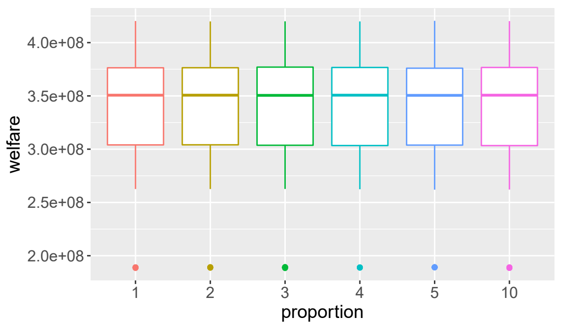
Figure 7.10: Computational time for each week.

In view of these results, it can be safely assumed that it is not worthwhile to use Constraint Programming to obtain the schedule of the routes (inside the ALNS method), since the computational times are significantly worse. Therefore, ALNS_CPSAT will not be taken into account in the remainder of this chapter.

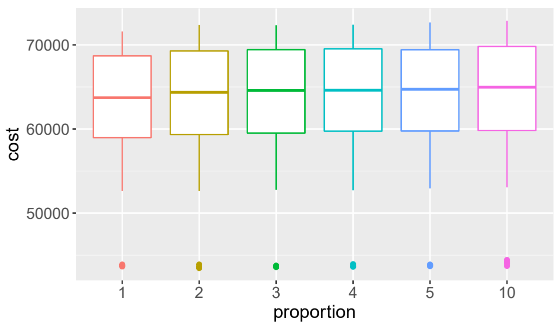
7.2.4 Real data results

To solve the real data using the ALNS_WC method the following automatic configurations of p were considered: *auto*_10%, *auto*_5%, *auto*_4%, *auto*_3%, *auto*_2% and *auto*_1%, with a time limit of 90 minutes. The reason for choosing these values is that, during the analysis of the Solomon instances, it was noticeable that the higher the number of services, the lower the proportion should be. Moreover, the automatic configurations of the parameter have presented better results.

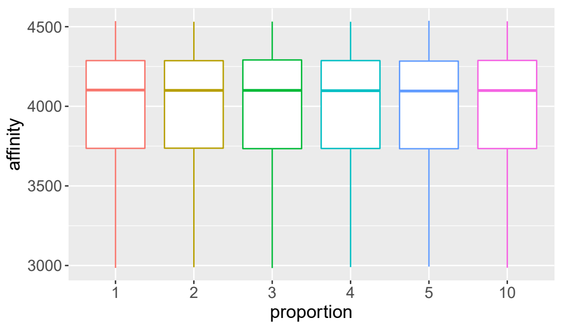
Figure 7.11 presents the boxplots of the objective function values for the different configurations of p. In terms of overall welfare (see Figure 7.11a), it is not easy to appreciate the difference between the configurations of p. However, if it is separated into the affinity and the soft time window penalization (Figures 7.11c and 7.11e), it can be observed that $p = auto\_1\%$ reaches the best results. The same configuration of p is slightly better in the case of the cost (Figure 7.11b), since it presents an improvement of the overtime (Figure 7.11d). Therefore, $p = auto\_1\%$ is the configuration used in the computational analysis presented below.



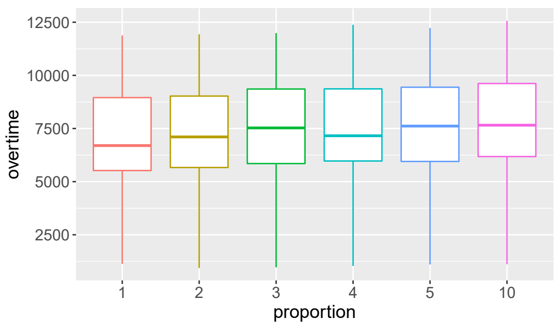
(a) Welfare.



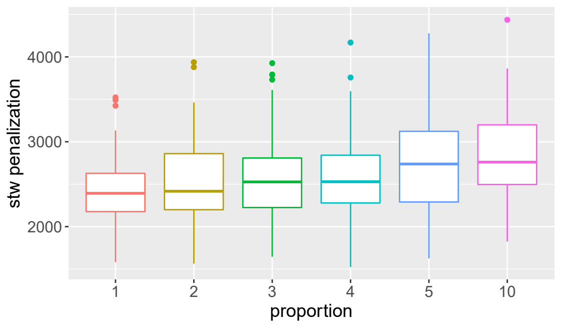
(b) Cost.



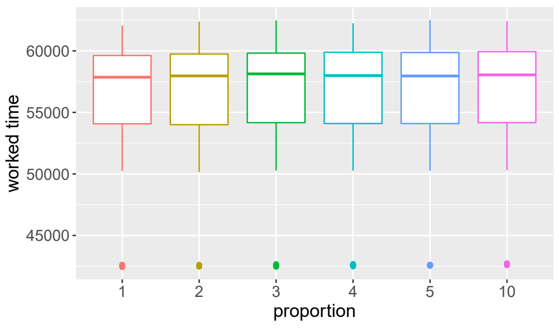
(c) Affinity.



(d) Overtime.



(e) Soft time window penalization.

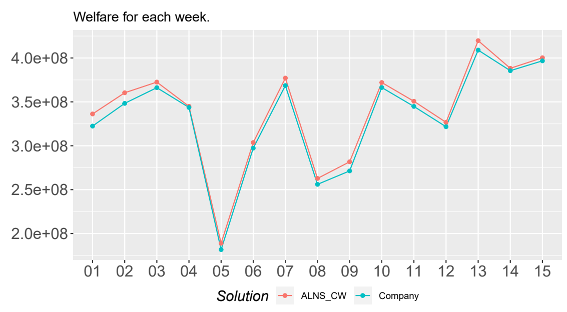


(f) Worked time.

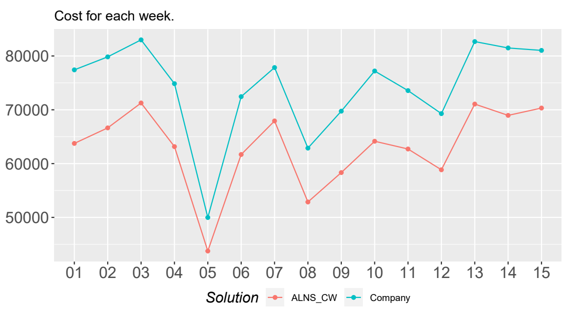
Figure 7.11: Objective function values in terms of p (ALNS_WC).

A comparison between the results obtained by the ALNS_WC and the schedules used by the company during the considered weeks is shown in Figure 7.12.

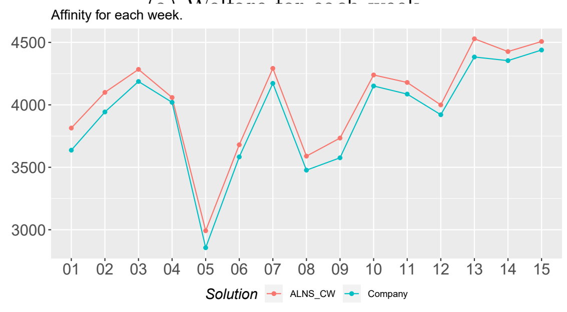
It can be seen in Figures 7.12a and 7.12b that the ALNS_WC always finds better results than those employed by the company. In addition, if breaking down the welfare into the affinity (see Figure 7.12c) and the penalization (see Figure 7.12e), it can also be observed that the ALNS_WC improves the results in both cases. Similar conclusions are obtained when disaggregating the cost in overtime (see Figure 7.12d) and worked time (see Figure 7.12f).



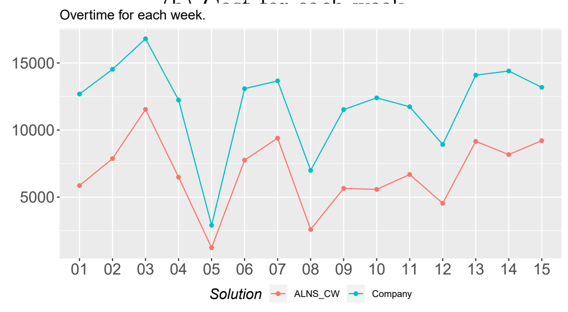
(a) Welfare for each week.



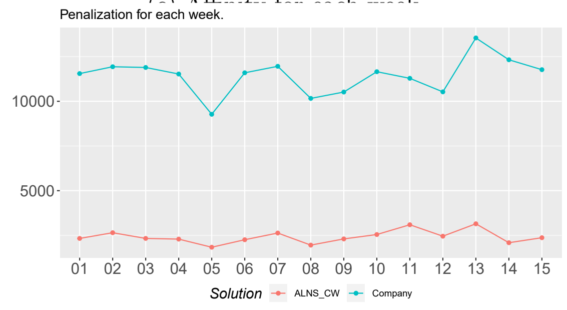
(b) Cost for each week.



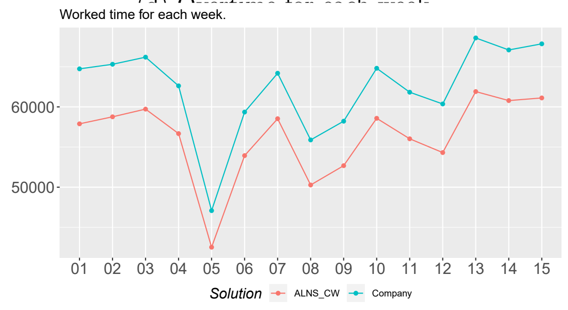
(c) Affinity for each week.



(d) Overtime for each week.



(e) Penalization for each week.



(f) Worked time for each week.

Figure 7.12: Objective functions values for real instances (ALNS_WC).

To explore the results in more detail, the solution will be deeply analyzed with respect to the following characteristics: affinity, penalization, overtime, travel time, unpaid break and idle time (paid break).

Thus, Table 7.10 shows the results with respect to the affinity and the penalization on a weekly basis, both globally and on average per user. In all the weeks, the algorithm clearly outperforms the results of the company.

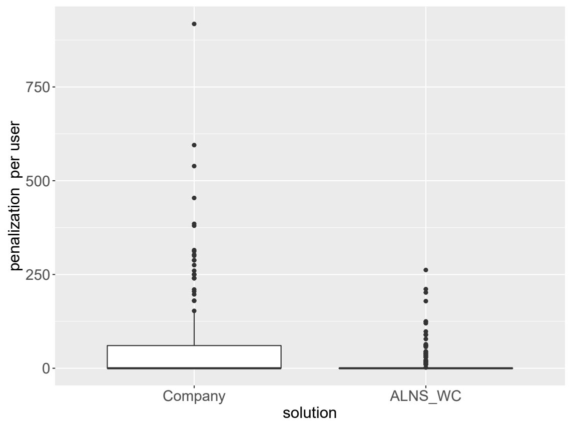
Week	Solution	Global			Per user	
		Welfare ($\times e8$)	Affinity ($\times e3$)	Penalization ($\times e3$)	Affinity	Penalization
1	Algorithm	3.36	3.81	2.33	4.33	2.73
	Company	3.22	3.64	11.55	4.14	13.73
2	Algorithm	3.60	4.10	2.65	4.66	2.76
	Company	3.48	3.94	11.93	4.50	14.94
3	Algorithm	3.72	4.28	2.33	4.76	2.75
	Company	3.66	4.19	11.89	4.66	14.66
4	Algorithm	3.44	4.06	2.29	4.66	3.18
	Company	3.44	4.02	11.53	4.62	15.19
5	Algorithm	1.88	2.99	1.84	4.73	2.95
	Company	1.82	2.86	9.27	4.60	16.10
6	Algorithm	3.03	3.68	2.26	4.37	2.61
	Company	2.97	3.58	11.60	4.25	15.39
7	Algorithm	3.77	4.29	2.63	4.75	3.15
	Company	3.68	4.17	11.96	4.62	14.60
8	Algorithm	2.62	3.59	1.96	4.73	2.50
	Company	2.56	3.48	10.16	4.64	15.03
9	Algorithm	2.81	3.73	2.30	4.57	2.92
	Company	2.71	3.58	10.52	4.44	15.21
10	Algorithm	3.72	4.24	2.55	4.65	2.93
	Company	3.66	4.15	11.66	4.59	14.49
11	Algorithm	3.50	4.18	3.09	4.76	3.11
	Company	3.45	4.09	11.29	4.64	14.50
12	Algorithm	3.26	3.99	2.45	4.73	3.25
	Company	3.22	3.92	10.53	4.65	14.66
13	Algorithm	4.19	4.53	3.15	4.78	3.14
	Company	4.09	4.38	13.54	4.67	15.83
14	Algorithm	3.88	4.43	2.09	4.80	2.58
	Company	3.85	4.35	12.32	4.75	14.20
15	Algorithm	4.00	4.50	2.37	4.81	3.10
	Company	3.97	4.44	11.77	4.71	13.70

Table 7.10: Computational results for real instances according to ALNS_WC (welfare).

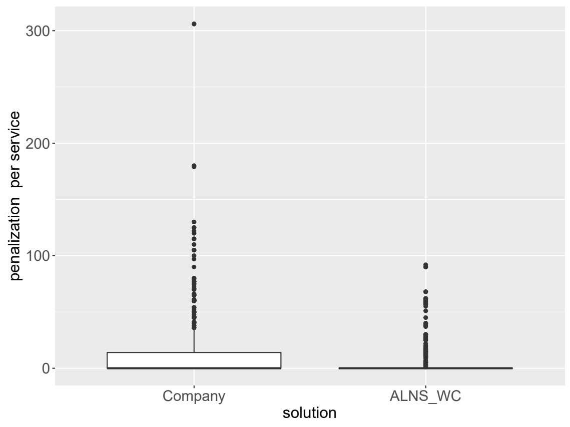
On the other hand, the quality of the solutions can be measured in terms of affinity obtaining the percentage of services where the best possible level of affinity is reached, which means that the best caregivers (according to the level of affinity) have been assigned to these services. Thus, Table 7.11 shows a comparison between the results of the company and the ones of the algorithm per week. In all the cases, the algorithm solution is clearly better than the solution proposed by the company. The percentages of services that have the highest affinity for the company and the algorithm are 81.04% and 88.87%, respectively, when considering all the services across all the weeks.

Week	Company	Algorithm	Week	Company	Algorithm
1	76.30%	86.24%	9	78.46%	90.09%
2	75.90%	85.00%	10	79.58%	87.70%
3	82.79%	90.27%	11	84.82%	90.91%
4	82.96%	86.21%	12	83.09%	88.69%
5	77.40%	89.41%	13	80.52%	91.05%
6	76.64%	86.00%	14	86.59%	91.35%
7	83.89%	91.27%	15	85.83%	91.48%
8	78.42%	86.44%			

Table 7.11: Weekly percentage of services with maximum affinity level (ALNS_WC).



(a) For the users.



(b) For the services.

Figure 7.13: Soft time window penalization according to ALNS_WC (all weeks).

The soft time window penalization for the users and the services in all the weeks is represented in the boxplots of Figures 7.13a and 7.13b. The graphical results confirm a remarkable improvement with respect to the solutions of the company.

		Global			Per caregiver			
Week	Solution	Cost	Overtime	Worked time	Overtime	Travel time	Break	Idle time
1	Algorithm	1062.49	97.66	964.82	2.55	1.23	6.48	0.10
	Company	1290.31	211.38	1078.93	5.42	1.29	7.48	3.05
2	Algorithm	1110.62	131.21	979.40	3.48	1.34	6.67	0.13
	Company	1330.55	242.13	242.13	6.37	1.37	7.28	2.97
3	Algorithm	1187.72	192.33	995.38	5.58	1.51	8.17	0.32
	Company	1383.23	280.06	1103.16	8.24	1.54	7.70	3.41
4	Algorithm	1052.64	108.07	944.56	2.92	1.32	6.45	0.18
	Company	1247.58	203.93	1043.65	5.51	1.33	7.22	2.86
5	Algorithm	729.22	20.44	708.77	0.53	0.89	4.99	0.08
	Company	833.06	48.33	784.73	1.27	0.94	6.51	2.07
6	Algorithm	1028.19	129.28	898.90	3.59	1.30	4.79	0.12
	Company	1207.38	218.10	989.28	5.89	1.31	6.76	2.62
7	Algorithm	1132.08	156.60	975.47	4.48	1.53	6.52	0.21
	Company	1297.50	227.71	1069.78	6.51	1.53	8.22	2.90
8	Algorithm	881.07	43.12	837.94	1.05	1.08	5.30	0.16
	Company	1047.96	116.53	931.43	2.99	1.11	6.15	2.53
9	Algorithm	972.19	94.20	877.99	2.55	1.28	6.65	0.20
	Company	1162.40	192.08	970.31	5.49	1.33	8.53	2.75
10	Algorithm	1069.21	92.89	976.32	2.46	1.42	5.84	0.19
	Company	1286.75	206.65	1080.10	5.59	1.41	6.58	2.99
11	Algorithm	1045.20	111.42	933.78	3.20	1.43	5.98	0.08
	Company	1226.06	195.60	1030.46	5.29	1.38	7.12	2.71
12	Algorithm	980.80	75.60	905.19	1.94	1.31	5.68	0.13
	Company	1154.76	148.68	1006.08	3.81	1.34	6.21	2.72
13	Algorithm	1184.37	152.59	1031.78	4.06	1.53	6.14	0.16
	Company	1377.91	234.85	1143.06	6.18	1.47	7.13	3.18
14	Algorithm	1149.30	136.22	1013.08	3.39	1.42	6.34	0.22
	Company	1358.13	240.00	1118.13	6.32	1.44	6.87	3.01
15	Algorithm	1172.04	153.46	1018.58	4.18	1.52	6.58	0.23
	Company	1350.56	219.73	1130.83	5.94	1.50	7.47	3.29

Table 7.12: Computational results in hours for real instances according to ALNS_WC (cost).

Table 7.12 shows the results per week with respect to the global overtime and worked time, as well as the average per caregiver with respect to the overtime, the travel time, the duration of the unpaid break and the idle time (i.e., the total duration of the paid breaks).

Since the overtime, the travel time, the duration of the unpaid break and the idle time are associated with the cost and are in the same time units, all their global results have been aggregated in Figure 7.14. So, Figure 7.14 shows the mean weekly times per caregiver according to the results in all the weeks.

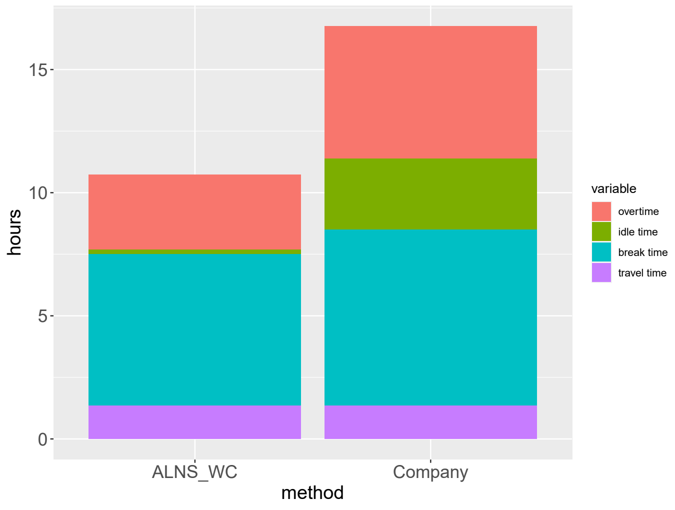
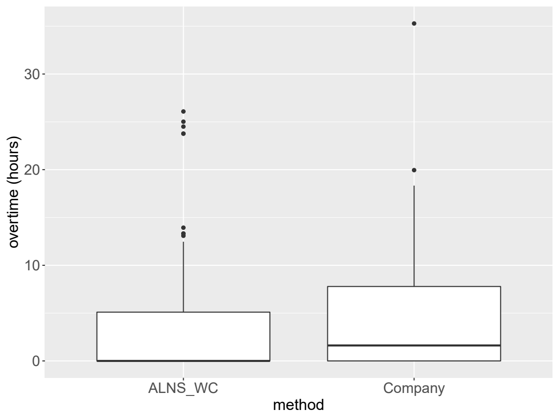


Figure 7.14: Mean weekly times per caregiver according to ALNS_WC (all weeks).

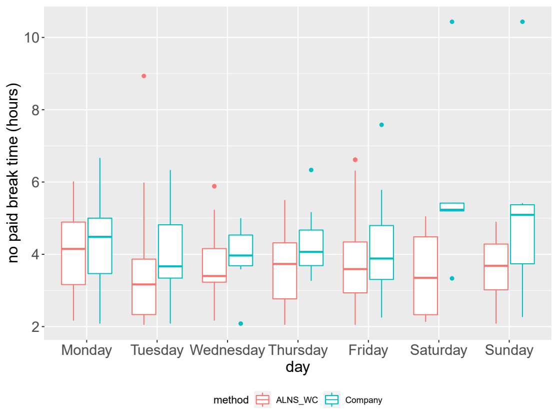
Both Table 7.12 and Figure 7.14 evidence the good behavior of the algorithm with respect to the results of the company. In the figure it can be appreciated that the mean of the overtime, the idle time and the break time are drastically reduced if the algorithm is employed. However, the difference is negligible in the case of travel times. The table also shows similar results over the weeks.

A more detailed study of the overtime, the unpaid break, the idle time and the travel time can be found in Figure 7.15. As sake of illustration, the four characteristics are shown in week[4] 6. Hence, Figure 7.15a presents the comparison of the evolution of the overtime with respect to the agreed weekly working time for the different caregivers in both solutions, showing again the good performance of the algorithm. Figures 7.15b to 7.15d make a comparison of both solutions on a daily basis. It is worth mentioning that there is a drastically reduction of the idle times every day (see Figure 7.15c). This reduction can be one of the key factors in reducing the overtimes. In terms of travel time, both solutions perform quite reasonably, with a median time of approximately 5 minutes every day and travel times that will never exceed 25 minutes.

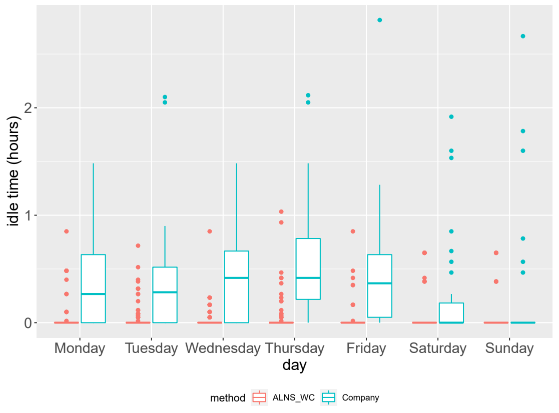
[4]A similar behavior is observed in the other weeks.



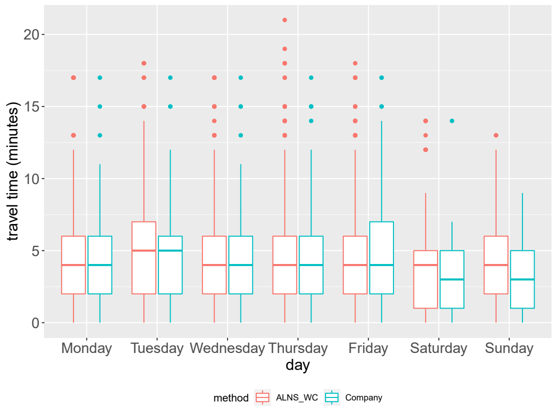
(a) Overtime for the caregivers.



(b) Unpaid break for the caregivers per day.



(c) Idle time for the caregivers per day.



(d) Travel time between services per day.

Figure 7.15: Overtime, unpaid break, idle time and travel time for week 6 (ALNS_WC).

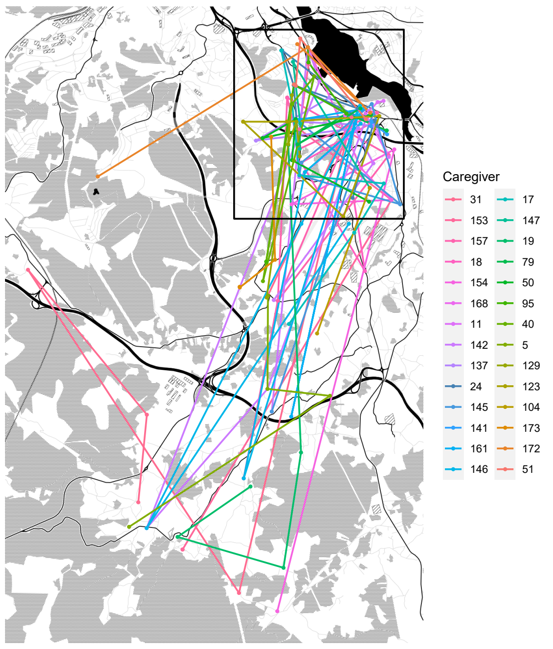
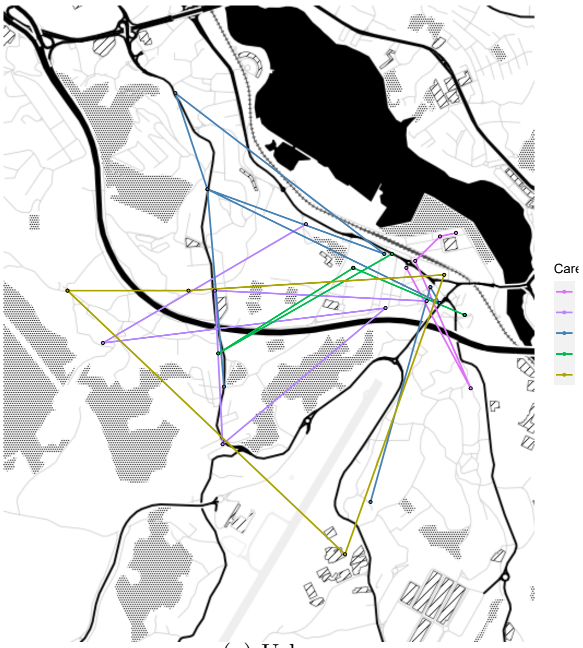


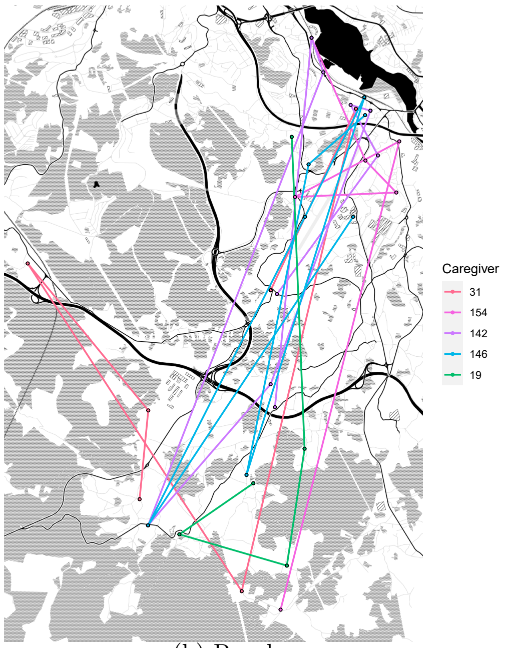
Figure 7.16: Routes for a random day (ALNS_WC).

Finally, Figure 7.16 illustrates the routes that caregivers follow on a random day, according to the solution given by the ALNS_WC. Analyzing these routes it can be seen that they cover two different areas: a rural and an urban one. The urban area (surrounded by the frame) is distinguished for having a lot of clustered services while the rural area has fewer services, but spread over a more extensive area.

To illustrate the different areas, 5 random caregivers who work on the urban area (see Figure 7.17a) and 5 caregivers who cover the rural area (see Figure 7.17b) were selected. It can be seen that the caregivers working in the rural area also have to carry out some urban services. This is necessary to balance the working time of the caregivers, because there are more services required in the urban area.



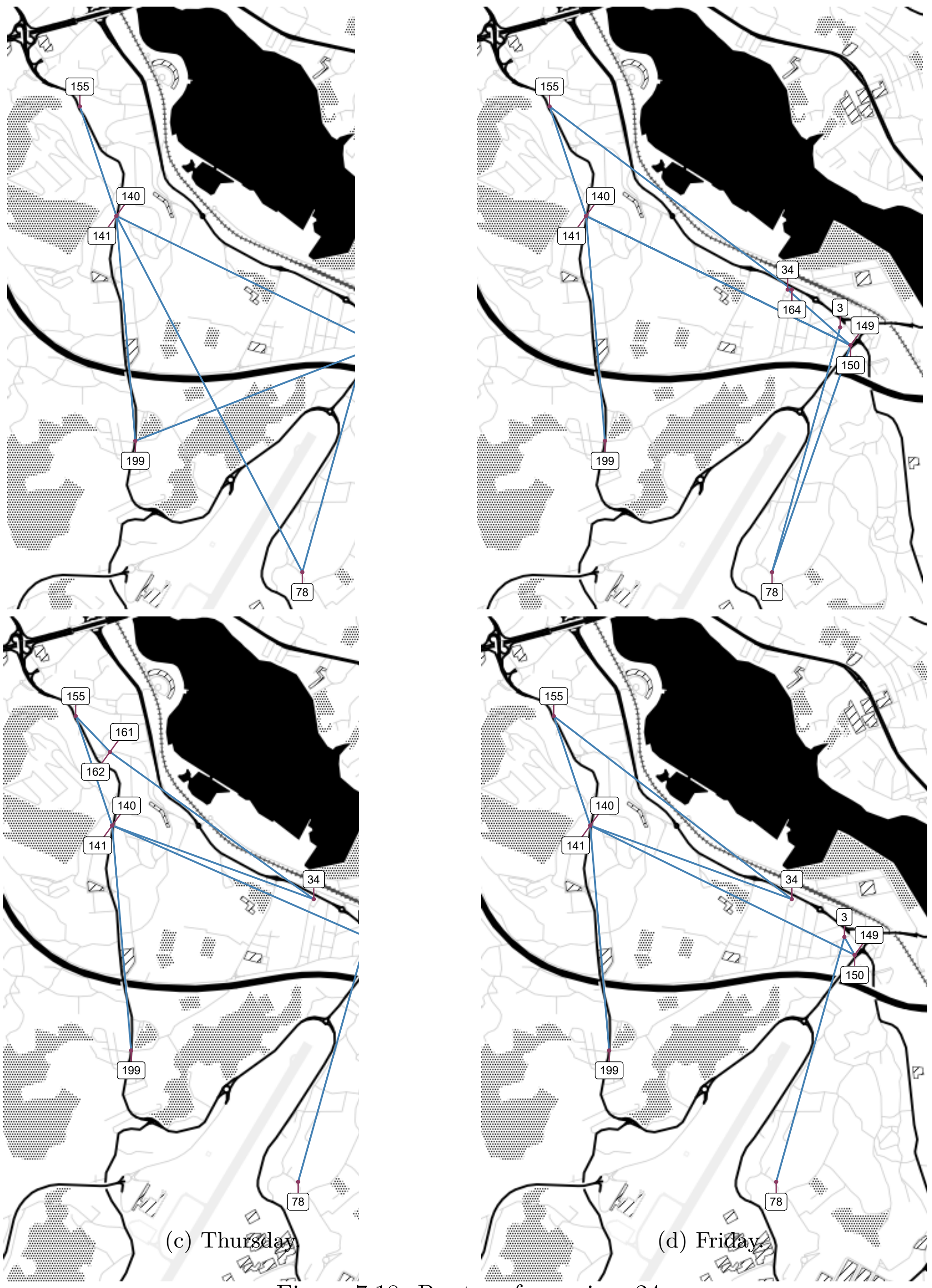
(a) Urban area.



(b) Rural area.

Figure 7.17: Routes separated into areas (ALNS_WC).

With the aim of analyzing more in detail the solution, Figure 7.18 presents the routes of caregiver 24 in four consecutive days (Tuesday is a day off). Notice that, even though the routes are different, some services (3, 78, 140, 141, 155 and 199) appear on the three routes. This is a reasonable behavior since users prefer not to change the caregiver that attends them and recurring services tend to be assigned to the same caregiver.



(c) Thursday

(d) Friday.

Figure 7.18: Routes of caregiver 24.

Figure 7.19 shows the details of the schedule for the four days. Note that the caregiver only has one break per day, with a duration of more than two hours, which means that it is not considered as working time. It can be seen that, when the services are close to each other (for example, see services 149 and 150 on Figure 7.18a) the travel time between them is neglected. The daily availability of the caregiver starts at 7:25 and ends at 21:51, which is respected by all her services. According to the schedule, the caregiver works 7.55 hours on Monday, 9.12 hours on Wednesday, 8.5 hours on Thursday and 8.53 on Friday, resulting in a total of 33.7 hours. These working times do not exceed the daily and weekly maximum working time imposed by the contract of the caregiver, which is 13.42 hours and 38.5 hours, respectively.

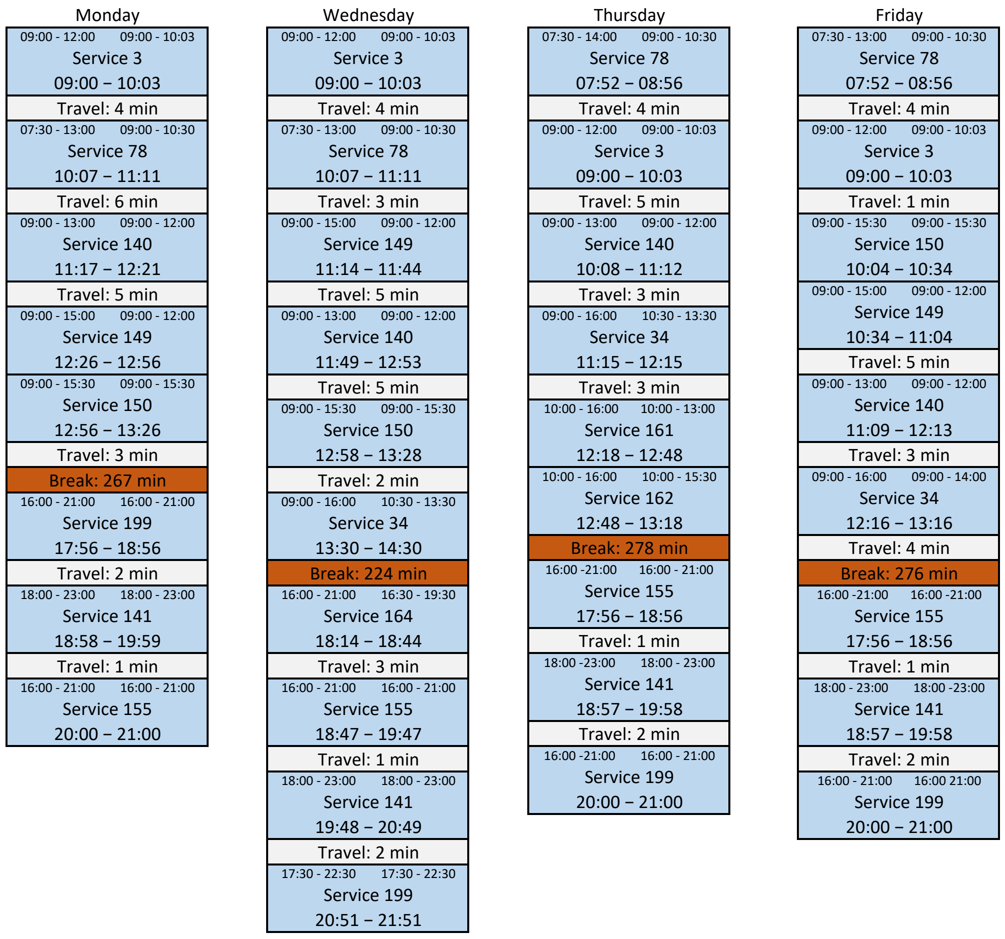
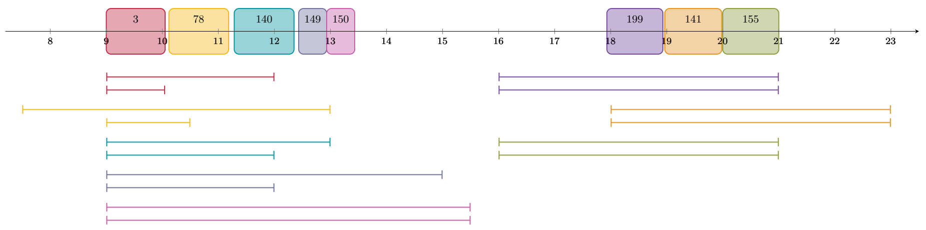
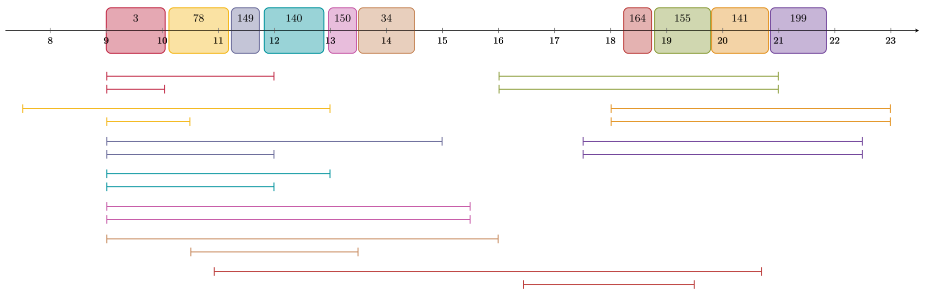


Figure 7.19: Schedules of caregiver 24.

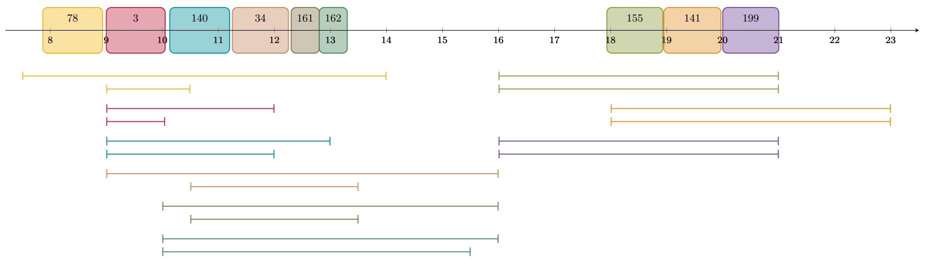
Finally, Figure 7.20 presents the schedules of the caregiver including the hard and soft time windows for each service. According to the schedules, services are as close as possible to their soft time windows, and they are always within the corresponding hard time windows.



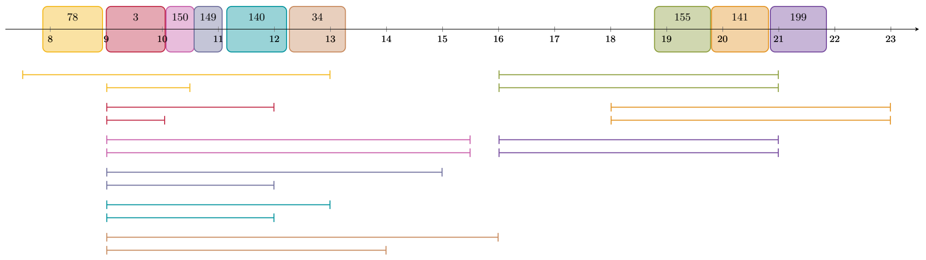
(a) Monday.



(b) Wednesday.



(c) Thursday.



(d) Friday.

Figure 7.20: Schedule for each day of caregiver 24.

7.3 Hierarchical approach: cost-welfare

This section deals with the computational study to evaluate the good performance of the ALNS_CW method. Note that this method combines the ALNS metaheuristic for the routes with the heuristic for the schedules described in Section 5.1.

7.3.1 Gurobi results

The MILP has been solved using Gurobi with a time limit of 12 hours. The results are presented in Table 7.13. Note that, although Gurobi finds a feasible solution for 72% of the instances, it only ensures the global optimality in 30.5% of them. It is worth noting that it is not able to solve any of the instances with 100 services.

instance	feasible	opt	gap	secs	instance	feasible	opt	gap	secs
10_01	✓	✓	-	0.56	10_06	✓	✓	-	13.24
10_02	✓	✓	-	2532.30	10_07	✓	✓	-	17654.99
10_03	✓	✓	-	2882.26	10_08	✓	✓	-	11095.85
10_04	✓	?	143.98	limit	10_09	✓	?	486.50	limit
10_05	✓	?	1294.81	limit	10_10	✓	✓	-	3955.34
15_01	✓	✓	-	1811.69	15_06	✓	?	904.22	limit
15_02	✓	✓	-	15162.27	15_07	✓	?	1251.06	limit
15_03	✓	✓	278.51	limit	15_08	✓	?	618.82	limit
15_04	✓	?	30.99	limit	15_09	✓	?	984.49	limit
15_05	✓	?	176.83	limit	15_10	✓	?	968.86	limit
25_01	✓	?	198.27	limit	25_06	✓	?	1247.53	limit
25_02	✓	?	314.39	limit	25_07	✓	?	388.51	limit
25_03	✓	?	433.10	limit	25_08	-	-	-	-
25_04	-	-	-	-	25_09	✓	?	1734.74	limit
25_05	-	-	-	-	25_10	✓	?	2056.57	limit
50_01	✓	?	364.33	limit	50_06	✓	?	905.70	limit
50_02	✓	?	172.99	limit	50_07	✓	?	1480.54	limit
50_03	✓	?	636.62	limit	50_08	✓	?	749.31	limit
50_04	✓	?	1116.37	limit	50_09	✓	?	10.67	limit
50_05	✓	?	1295.95	limit	50_10	-	-	-	-
100_01	-	-	-	-	100_06	-	-	-	-
100_02	-	-	-	-	100_07	-	-	-	-
100_03	-	-	-	-	100_08	-	-	-	-
100_04	-	-	-	-	100_09	-	-	-	-
100_05	-	-	-	-	100_10	-	-	-	-

Table 7.13: Gurobi computational results prioritizing cost over welfare.

7.3.2 ALNS_CW results

Regarding the configuration of the ALNS_CW, the parameters considered are the same as the ones used when prioritizing the welfare over the cost in Section 7.2. The number of iterations is 50, 100, 150, 200, 250, 500, 750 and 1000. The proportion of solution to be destroyed is 25%, 50%, 75%, 100%, *auto*_25%, *auto*_50%, *auto*_75% and *auto*_100%. Each combination of parameters has been run 5 times with a time limit of 1 hour.

7.3.2.1 Gurobi vs ALNS_CW

Tables 7.14 and 7.15 present, for the instances with 10 and 15 services, the mean RPD with respect to the first and second objective, respectively. So, Table 7.14 shows the RPD value for the first objective and, in the instances where this value is 0, the RPD value for the second objective is shown in Table 7.15. The values indicate the mean of the deviations of the solutions found by the ALNS_CW from the ones found by Gurobi. For the ALNS_CW, the different values of p have been combined with 1000 iterations.

Analyzing the results, they show that the ALNS_CW is able to find solutions of the same quality or even better[5] than the ones provided by Gurobi. In particular, as it happened when

[5]Negative values of the RPD mean that the solution of the ALNS_CW improves the one found using Gurobi.

prioritizing the welfare over the cost, the best solutions are found using larger values of p (75%, *auto*_75%, 100% and *auto*_100%). This suggests that, for small instances, it is better to explore the solution space by allowing major changes in the solution.

	25%	50%	75%	100%	*auto*_25%	*auto*_50%	*auto*_75%	*auto*_100%
10_01	2.27	0	0	0	1.52	0	0	0
10_02	0	0	0	0	0	0	0	0
10_03	0	0.170	0	0	0.68	0	0	0
10_04	0.08	0	0	0	0.12	0.04	0	0
10_05	0	0	0	0	0	0	0	0
10_06	3.52	0	0	0	2.05	0.88	0	0
10_07	0.36	0.73	0	0	0.36	0.73	0.36	0
10_08	2.81	0.58	0	0	2.81	0.58	0	0
10_09	0	0	0	0	0	0	0	0
10_10	2.145	0	0	0	1.03	0	0	0
15_01	0	0.44	0.22	0	0.22	0.66	0.22	0
15_02	0.35	0.13	0.27	0.41	0.70	0.54	0.27	0.41
15_03	0	0	0	0	0.42	0	0	0
15_04	0	0	0	0	0	0	0	0
15_05	0	0	0	0	0	0	0	0
15_06	1.35	0.45	0.33	0	1.80	0.78	0.39	0.39
15_07	-2.00	-2.18	-2.73	-2.73	-1.64	-2.73	-2.73	-2.73
15_08	-1.47	-4.58	-4.58	-4.58	-2.08	-3.41	-4.58	-4.58
15_09	-0.15	-0.85	-1.55	-1.39	0.07	-0.62	-1.16	-1.55
15_10	-0.29	-1.19	-1.19	-1.19	0.05	-1.19	-1.19	-1.19

Table 7.14: Mean RPD values comparing ALNS_CW with Gurobi (first objective).

	25%	50%	75%	100%	*auto*_25%	*auto*_50%	*auto*_75%	*auto*_100%
10_01	-	0	0	0	-	0	0	0
10_02	0	0	0	0	0	0	0	0
10_03	0	-	0	0	-	0	0	0
10_04	-	-8.04	-8.04	-8.04	-	-	-8.04	-8.04
10_05	-2.48	-2.48	-2.48	-2.48	-2.48	-2.48	-2.48	-2.48
10_06	-	0	0	0	-	-	0	0
10_07	-	-	0	0	-	-	-	0
10_08	-	-	0	0	-	-	0	0
10_09	-19.32	-19.39	-19.39	-19.39	-19.33	-19.39	-19.39	-19.39
10_10	-	0	0	0	-	0	0	0
15_01	0	-	-	0	-	-	-	0
15_02	-	-	-	-	-	-	-	-
15_03	-5.11	-5.11	-5.11	-5.11	-	-5.11	-5.11	-5.11
15_04	-1.51	-1.51	-1.51	-1.51	-1.51	-1.51	-1.51	-1.51
15_05	-5.34	-5.34	-5.34	-5.34	-5.34	-5.34	-5.34	-5.34
15_06	-	-	-	-16.21	-	-	-	-
15_07	-	-	-	-	-	-	-	-
15_08	-	-	-	-	-	-	-	-
15_09	-	-	-	-	-	-	-	-
15_10	-	-	-	-	-	-	-	-

Table 7.15: Mean RPD values comparing ALNS_CW with Gurobi (second objective).

Since the solutions found by Gurobi are not necessary the best ones, Table 7.16 presents the mean RPD value (first objective) for all the instances, comparing the solutions obtained by the algorithm with the best solution found.

	25%	50%	75%	100%	*auto*_25%	*auto*_50%	*auto*_75%	*auto*_100%
0_01	2.27	0	0	0	1.52	0	0	0
10_02	0	0	0	0	0	0	0	0
10_03	0	0.17	0	0	0.68	0	0	0
10_04	0.08	0	0	0	0.12	0.04	0	0
10_05	0	0	0	0	0	0	0	0
10_06	3.52	0	0	0	2.05	0.88	0	0
10_07	0.36	0.73	0	0	0.36	0.73	0.36	0
10_08	2.81	0.58	0	0	2.81	0.58	0	0
10_09	0	0	0	0	0	0	0	0
10_10	2.145	0	0	0	1.03	0	0	0
15_01	0	0.44	0.22	0	0.22	0.66	0.22	0
15_02	0.35	0.13	0.27	0.41	0.70	0.54	0.27	0.41
15_03	0	0	0	0	0.42	0	0	0
15_04	0	0	0	0	0	0	0	0
15_05	0	0	0	0	0	0	0	0
15_06	1.35	0.45	0.33	0	1.80	0.78	0.39	0.39
15_07	0.75	0.56	0	0	1.12	0	0	0
15_08	3.25	0	0	0	2.61	1.22	0	0
15_09	1.41	0.71	0	0.15	1.65	0.94	0.39	0
15_10	0.91	0	0	0	1.27	0	0	0
25_01	0	0	0	0	0.01	0	0	0
25_02	0.03	0.02	0.02	0.01	0.03	0.02	0.03	0.03
25_03	0.05	0.02	0	0.03	0.08	0.01	0.01	0.03
25_04	6.88	6.27	6.27	7.49	8.17	7.21	5.67	5.67
25_05	1.48	0.37	1.15	0.91	1.41	0.66	0.78	0.70
25_06	1.61	0.52	0.36	0.44	1.45	0.36	0.44	0.08
25_07	0	0	0	0	0.24	0	0	0
25_08	0	0	0	0	0	0	0	0
25_09	0.24	0	0	0	0.49	0	0	0
25_10	0	0	0	0	0	0	0	0
50_01	0.01	0.03	0.06	0.06	0.02	0	0.02	0
50_02	0.01	0.01	0.01	0.02	0.01	0.01	0.01	0.01
50_03	0.04	0.02	0.03	0.04	0.03	0.03	0.04	0.05
50_04	2.44	1.66	3.13	3.73	1.69	1.23	0.64	1.47
50_05	0.11	0.53	0.78	1.64	0.55	0.51	0.53	0.64
50_06	1.89	1.27	2.15	2.48	2.46	1.62	1.39	0.69
50_07	0.37	0.92	2.47	1.48	0.25	1.70	0.71	0.89
50_08	2.70	0.91	1.88	3.67	3.37	2.58	0.62	1.38
50_09	0.42	0.05	0.42	0.58	1.37	0.08	0.06	0.28
50_10	0.22	0.71	0.74	0.41	0.68	0.22	0.25	0.25
100_01	0.06	0.09	0.16	1.29	0.05	0.07	0.09	0.09
100_02	0.12	0.72	1.45	1.55	0.05	0.07	0.19	0.35
100_03	0.05	0.15	0.23	0.27	0.04	0.07	0.09	0.14
100_04	2.93	8.04	15.38	9.88	1.34	3.23	6.01	5.98
100_05	2.50	3.37	6.17	6.59	1.36	2.97	3.89	3.41
100_06	0.88	4.46	6.39	-	0.33	1.78	2.03	4.72
100_07	1.47	-	-	-	1.04	2.05	3.53	-
100_08	1.55	3.28	2.92	-	1.00	1.17	2.54	1.45
100_09	0.81	2.26	-	-	0.81	1.25	0.94	1.49
100_10	1.52	1.47	3.64	4.69	0.96	0.95	2.10	2.49

Table 7.16: RPD values comparing ALNS_CW with the best solution found.

According to the results in Table 7.16, big values of p, like 75%, 100% and *auto*_100%, would be the best option for instances with 10 and 15 services. However, when the number of services increases, it can be seen that smaller values of p result in better solutions. Furthermore, for instances with 100 services, the configurations $p \in \{50\%, 75\%, 100\%, auto\_100\%\}$ are not able to solve some of the instances in the given time.

7.3.2.2 Parameter analysis of ALNS_CW

Figures 7.21, 7.22 and 7.23 present the evolution of the success rate (the proportion of times where the best solution is found) and the computational time (in seconds) along the iterations. With regard to instances with 10 services (see Figure 7.21) the best configurations, according to the success rate, are $p \in \{75\%, 100\%, auto\_75\%, auto\_100\%\}$, being $auto\_75\%$ and $auto\_100\%$ the fastest ones. Figure 7.22 shows that, for instances with 15 services, $p = auto\_100\%$ is the best configuration in terms of success rate and computational time. For instances with 25 services, Figure 7.23 shows that both $p = 75\%$ and $p = auto\_100\%$ behave similarly in terms of success rate, but $p = auto\_100\%$ is faster. Apart from that, any $p \in \{auto\_50\%, auto\_75\%\}$ also shows a good success rate and the experiments are faster than the ones with $p \in \{75\%, auto\_100\%\}$.

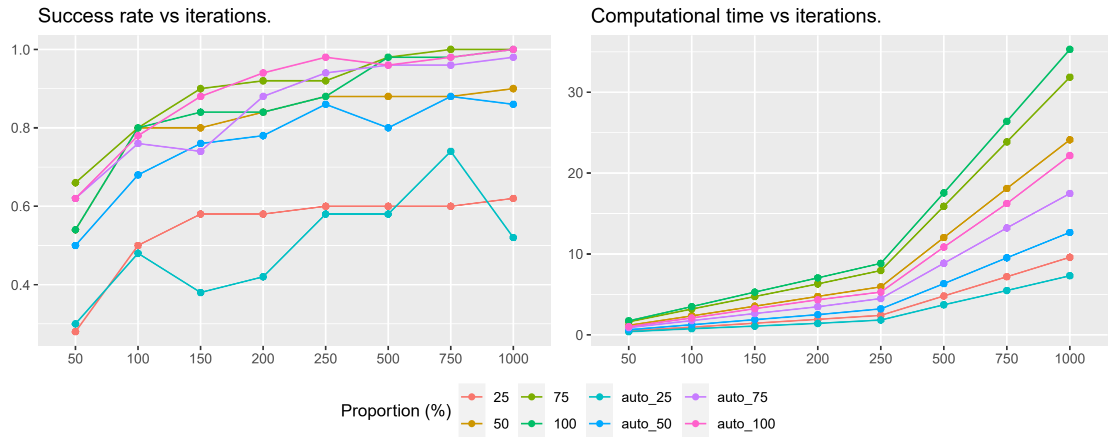


Figure 7.21: Computational results for instances with 10 services (ALNS_CW).

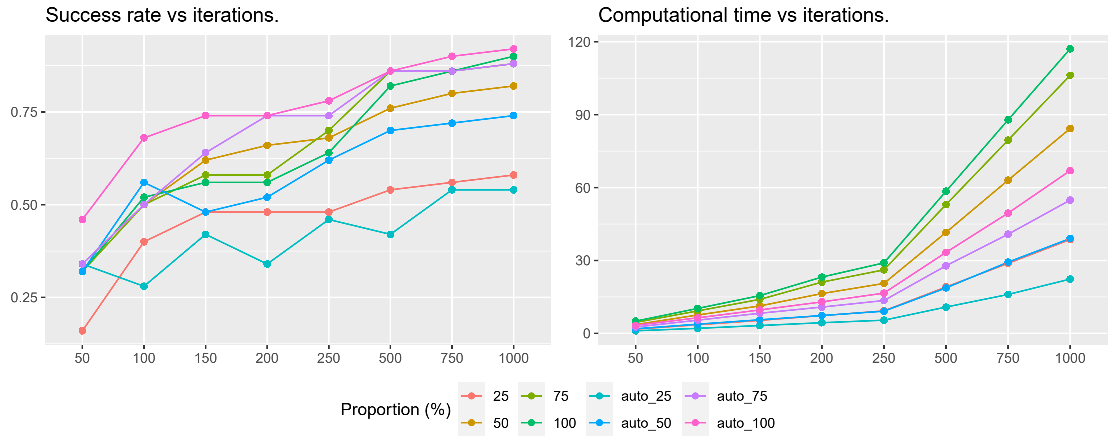


Figure 7.22: Computational results for instances with 15 services (ALNS_CW).

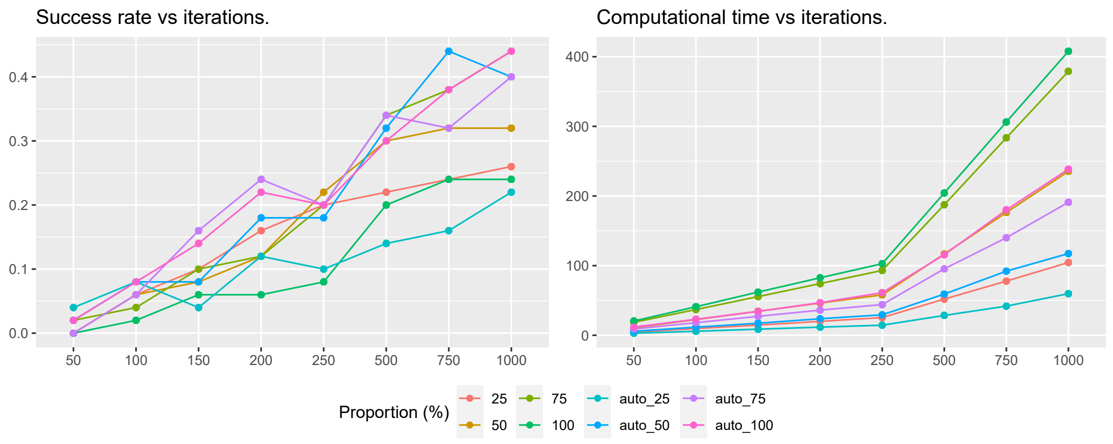


Figure 7.23: Computational results for instances with 25 services (ALNS_CW).

Figures 7.24 and 7.25 evaluate the mean RPD and computational time along the iterations for instances with 50 and 100 services.

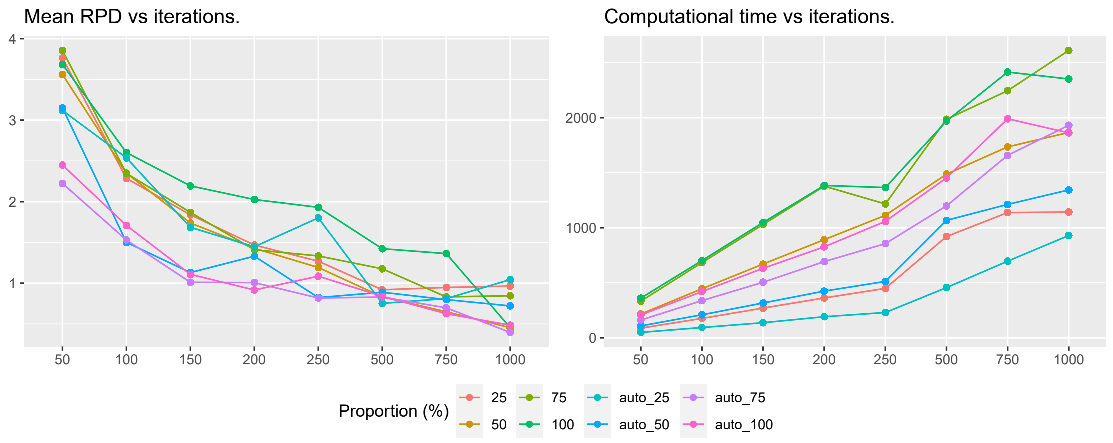


Figure 7.24: Computational results for instances with 50 services (ALNS_CW).

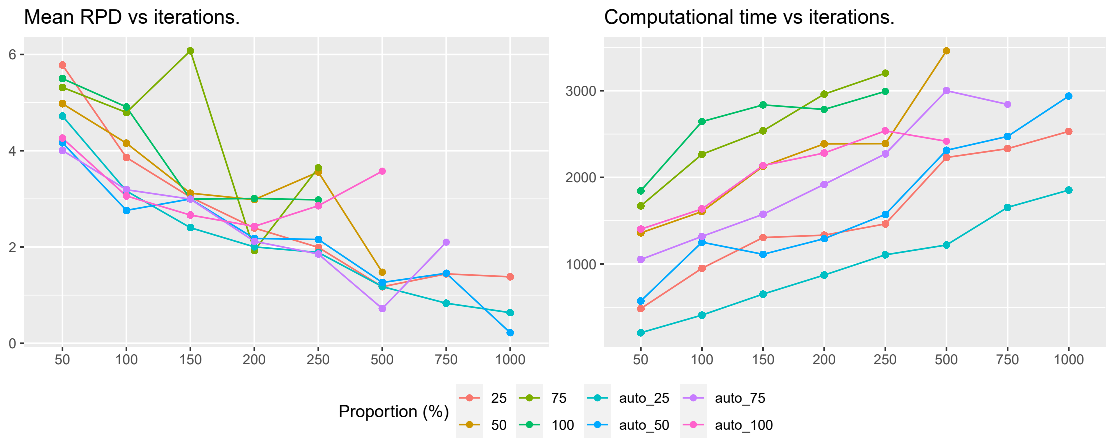


Figure 7.25: Computational results for instances with 100 services (ALNS_CW).

For instances with 50 services (see Figure 7.24), multiple configurations (any $p \in \{50\%, auto\_75\%, auto\_100\%\}$) present good behaviour in terms of RPD and computational times. Notice that, even though $p = 100\%$ has similar RPD values, it is slower than the other configurations. For instances with 100 services (see Figure 7.25) only three configurations, any

$p \in \{25\%, auto\_25\%, auto\_50\%\}$, allow to solve the instances within the time limit for 1000 iterations. From these configurations, the best one according to the mean RPD is $p = auto\_50\%$, although $p = auto\_25\%$ is faster and also reaches similar RPD values.

7.3.3 Constraint programming results

In view of the results presented in Section 7.2.3, the behavior of the ALNS combined with Constraint Programming or with the heuristic algorithm is similar. The main difference between the two methodologies is that the computational times when using the CPSAT solver are significantly worse. Therefore, this section focuses on the comparison between the algorithm presented in Chapter 5 and the Constraint Programming method.

7.3.3.1 Heuristic scheduling algorithm vs CPSAT

The Constraint Programming method and the heuristic algorithm presented in Chapter 5 (Algorithm 5.1) are used to find the best schedule once a route (that is, a set of ordered services assigned to a caregiver) is given, prioritizing the cost over the welfare. Although Algorithm 5.1 does not guarantee that an optimal solution will be obtained in all cases, although the computational experiments showed that high quality solutions can be found in a very short time. Thus, to evaluate the behavior of the heuristic algorithm, the two methods are used to obtain the schedules of the same set of routes.

To carry out the comparison a set of 3000000 routes was randomly generated, out of which 1109129 were feasible (in terms of hard time windows of services and caregivers). For each feasible route, its schedule was obtained using Constraint Programming (CPSAT) and the heuristic algorithm. In 99.997% of the routes the algorithm finds the optimal schedule whereas CPSAT always achieves optimality. The solution given by the algorithm was not the optimal in only 33 cases.

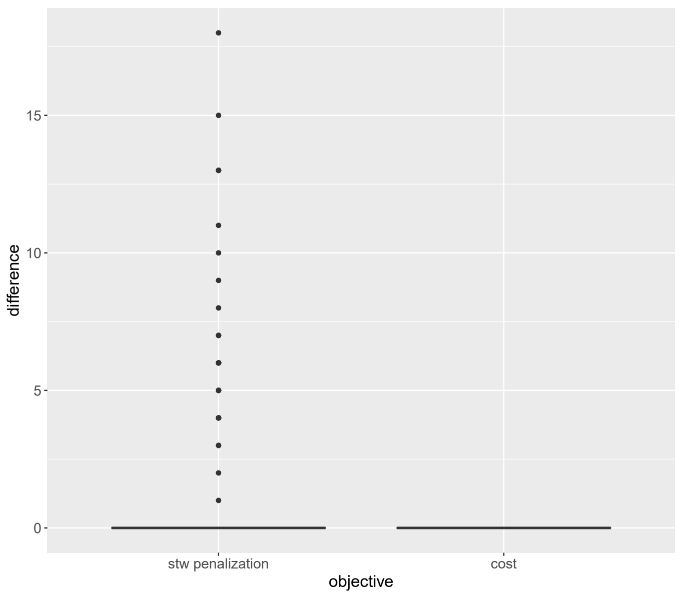
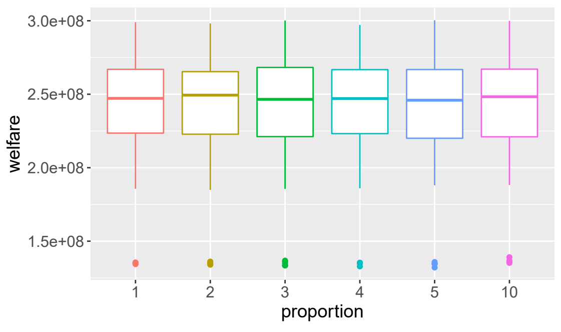


Figure 7.26: Objective function value differences (CPSAT vs Algorithm 5.1).

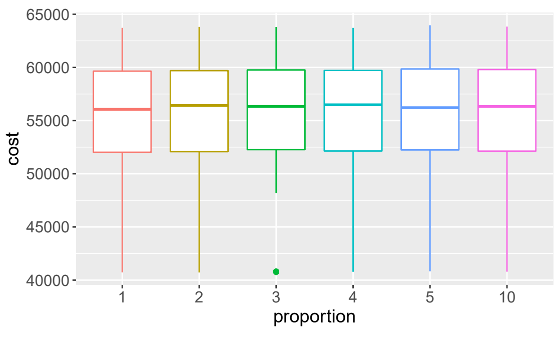
Figure 7.26 shows a boxplot of the differences between objective values (soft time window penalization and cost) of the schedules found by the algorithm and CPSAT. It can be seen that the most common scenario is the one where the objective values are equal, with only few outliers with differences between the values obtained by the algorithm and the ones given by CPSAT. In particular, the cost values are always equal and the only differences are found in the soft time window penalization. This makes sense as the cost of the scheduled is being prioritized over the welfare.

In terms of computational time, the mean time the heuristic algorithm needs to obtain the schedule of a route is 0.0077 seconds. Meanwhile, the mean CPSAT time per route is 0.0337 seconds. Therefore, it can be concluded that the heuristic scheduling algorithm, despite not being an exact method, finds very good results in short computational times.

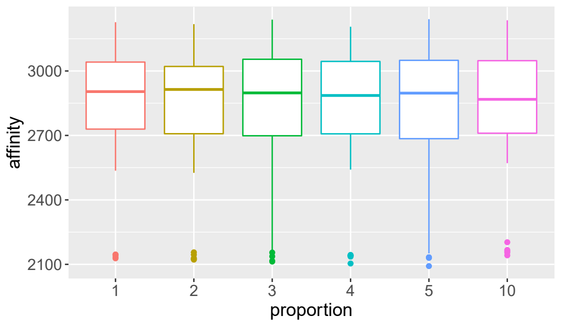
7.3.4 Real data results



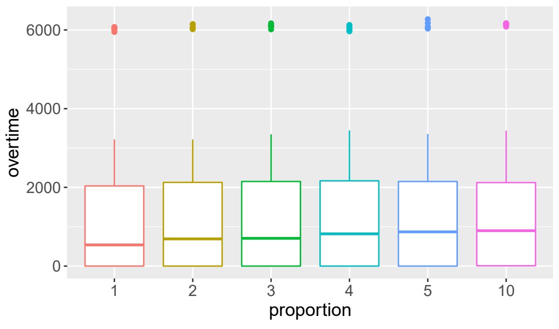
(a) Welfare.



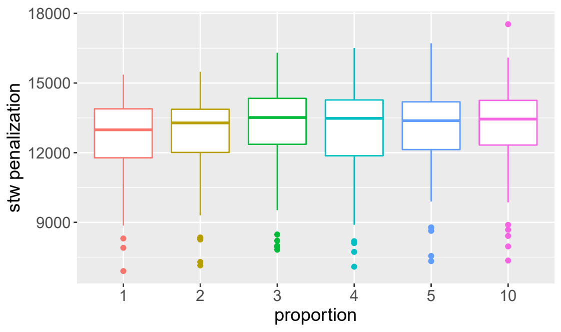
(b) Cost.



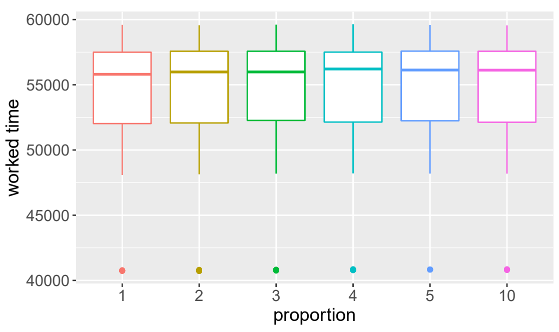
(c) Affinity.



(d) Overtime.



(e) Soft time window penalization.



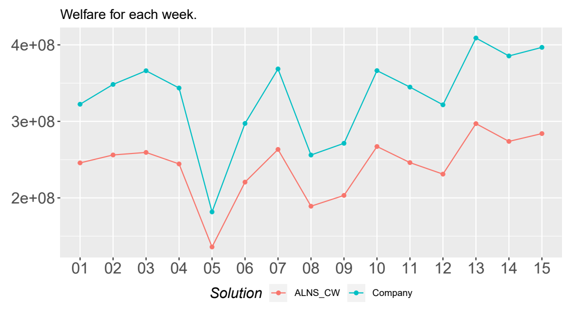
(f) Worked time.

Figure 7.27: Objective function values in terms of p (ALNS_CW).

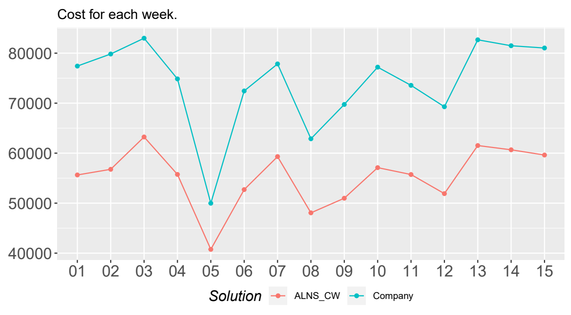
The conclusions reached during the analysis of the Solomon instances were that the higher the number of services, the lower the proportion should be, and that the automatic configurations of the parameter have presented better results. Therefore, the configurations of p considered to solve the real data were: *auto*_10%, *auto*_5%, *auto*_4%, *auto*_3%, *auto*_2% and *auto*_1%, with a time limit of 90 minutes.

Figure 7.27 presents the boxplots of the objective function values for the different configurations of p. In terms of overall cost (see Figure 7.27b), $p = auto\_1\%$ presents slightly better results. Separating the cost into overtime and worked time (Figures 7.27d and 7.27f), it can be seen that *auto*_1% is better than the other configurations for the overtime. In terms of the welfare (see Figure 7.27a), it is not easy to appreciate the difference between the configurations of p. However, splitting it into affinity and soft time window penalization (Figures 7.27c and 7.27e), it can be observed that $p = auto\_1\%$ presents better results in terms of the penalization. Therefore, $p = auto\_1\%$ is the configuration used in the computational analysis presented below.

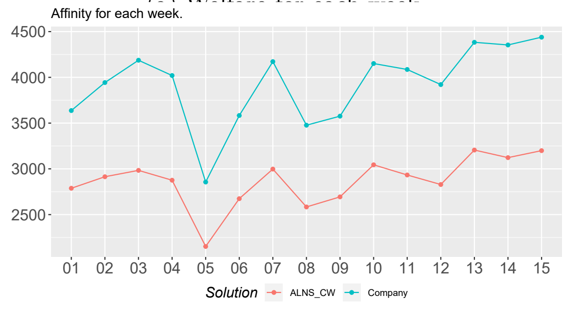
A comparison between the results obtained by the ALNS_CW and the schedules used by the company during the considered weeks is shown in Figure 7.28. Figures 7.28a, 7.28c and 7.28e are devoted to study the welfare, whereas Figures 7.28b, 7.28d and 7.28f are related to the cost.



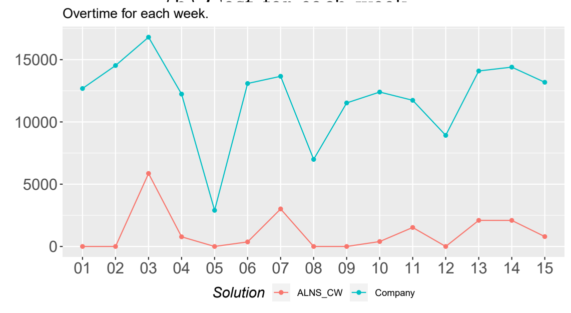
(a) Welfare for each week.



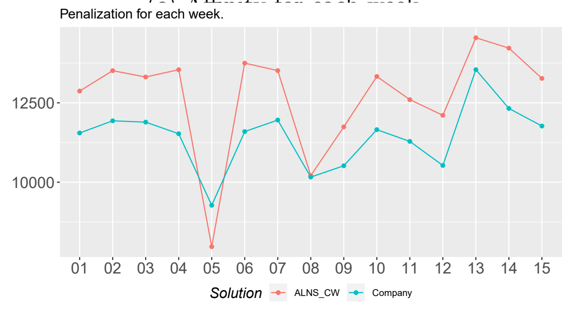
(b) Cost for each week.



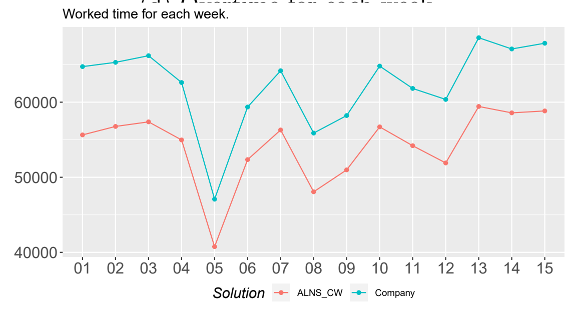
(c) Affinity for each week.



(d) Overtime for each week.



(e) Penalization for each week.



(f) Worked time for each week.

Figure 7.28: Computational results for real instances (ALNS_CW).

Let us start analyzing the cost, which is the objective function prioritized in this case. It can be seen in Figure 7.28b that the ALNS_CW finds better results for the cost than the schedules employed by the company. In addition, if the cost is analyzed in terms of the overtime (see Figure 7.28d) and working time (see Figure 7.28f), it can be seen that the ALNS_CW improves the results in both cases.

However, the ALNS_CW is not competitive with the company in the case of the second objective (see Figure 7.28a). This was to be expected, since the company prioritizes the welfare over the cost. Thus, Figure 7.28 it is only useful to show that it is possible to improve considerably the cost by sacrificing the welfare. It evidences that it may be interesting to try to strike a balance between both objectives in order to achieve the satisfaction of all parties involved in the problem, as it will be discussed in Section 7.5.

To explore the results in more detail, the solution will be deeply analyzed with respect to the following characteristics: overtime, travel time, unpaid break, idle time (paid break), affinity and penalization.

First, the results per week with respect to the global overtime and worked time, as well as the average per caregiver with respect to the overtime, the travel time, the duration of the unpaid break and the idle time are presented in Table 7.17. The solutions found by the algorithm are better than the ones used by the company in all the weeks. Specifically, the most improved elements are the overtime, which is greatly reduced, and the idle time, which is juts 0 for all weeks.

		Global			Per caregiver			
Week	Solution	Cost	Overtime	Worked time	Overtime	Travel time	Break	Idle time
1	Algorithm	927.37	0.00	927.37	0.00	0.47	6.86	0.00
	Company	1290.31	211.38	1078.93	5.42	1.29	7.48	3.05
2	Algorithm	946.16	0.00	946.16	0.00	0.62	7.28	0.00
	Company	1330.55	242.13	242.13	6.37	1.37	7.28	2.97
3	Algorithm	1053.99	97.69	956.30	2.87	0.64	7.29	0.00
	Company	1383.23	280.06	1103.16	8.24	1.54	7.70	3.41
4	Algorithm	929.11	12.92	916.19	0.36	0.74	5.64	0.00
	Company	1247.58	203.93	1043.65	5.51	1.33	7.22	2.86
5	Algorithm	679.26	0.00	679.26	0.00	0.23	6.60	0.00
	Company	833.06	48.33	784.73	1.27	0.94	6.51	2.07
6	Algorithm	878.57	6.08	872.49	0.21	0.81	5.56	0.00
	Company	1207.38	218.10	989.28	5.89	1.31	6.76	2.62
7	Algorithm	988.54	50.22	938.32	1.45	0.68	6.30	0.00
	Company	1297.50	227.71	1069.78	6.51	1.53	8.22	2.90
8	Algorithm	801.07	0.00	801.07	0.00	0.28	6.63	0.00
	Company	1047.96	116.53	931.43	2.99	1.11	6.15	2.53
9	Algorithm	849.78	0.00	849.78	0.00	0.64	6.97	0.00
	Company	1162.40	192.08	970.31	5.49	1.33	8.53	2.75
10	Algorithm	951.77	6.57	945.19	0.15	0.70	6.30	0.00
	Company	1286.75	206.65	1080.10	5.59	1.41	6.58	2.99
11	Algorithm	928.79	25.42	903.37	0.67	0.65	6.99	0.00
	Company	1226.06	195.60	1030.46	5.29	1.38	7.12	2.71
12	Algorithm	865.12	0.00	865.12	0.00	0.48	6.04	0.00
	Company	1154.76	148.68	1006.08	3.81	1.34	6.21	2.72
13	Algorithm	1025.43	34.93	990.50	0.90	0.62	7.10	0.00
	Company	1377.91	234.85	1143.06	6.18	1.47	7.13	3.18
14	Algorithm	1011.21	34.90	976.31	0.94	0.78	6.20	0.00
	Company	1358.13	240.00	1118.13	6.32	1.44	6.87	3.01
15	Algorithm	993.78	13.22	980.55	0.34	0.71	7.51	0.00
	Company	1350.56	219.73	1130.83	5.94	1.50	7.47	3.29

Table 7.17: Computational results in hours for real instances according to ALNS_CW (cost).

Figure 7.29 shows the aggregated mean weekly overtime, travel time, unpaid break and idle time per caregiver. It can be seen that the idle time disappears, the travel time decreases and the overtime is drastically reduced.

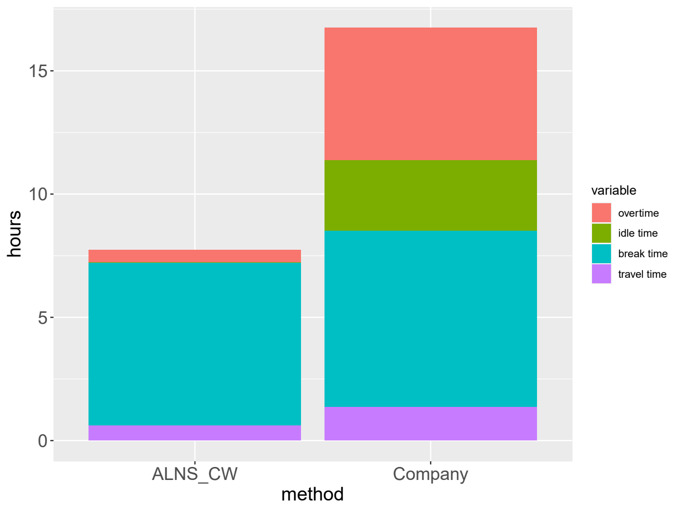
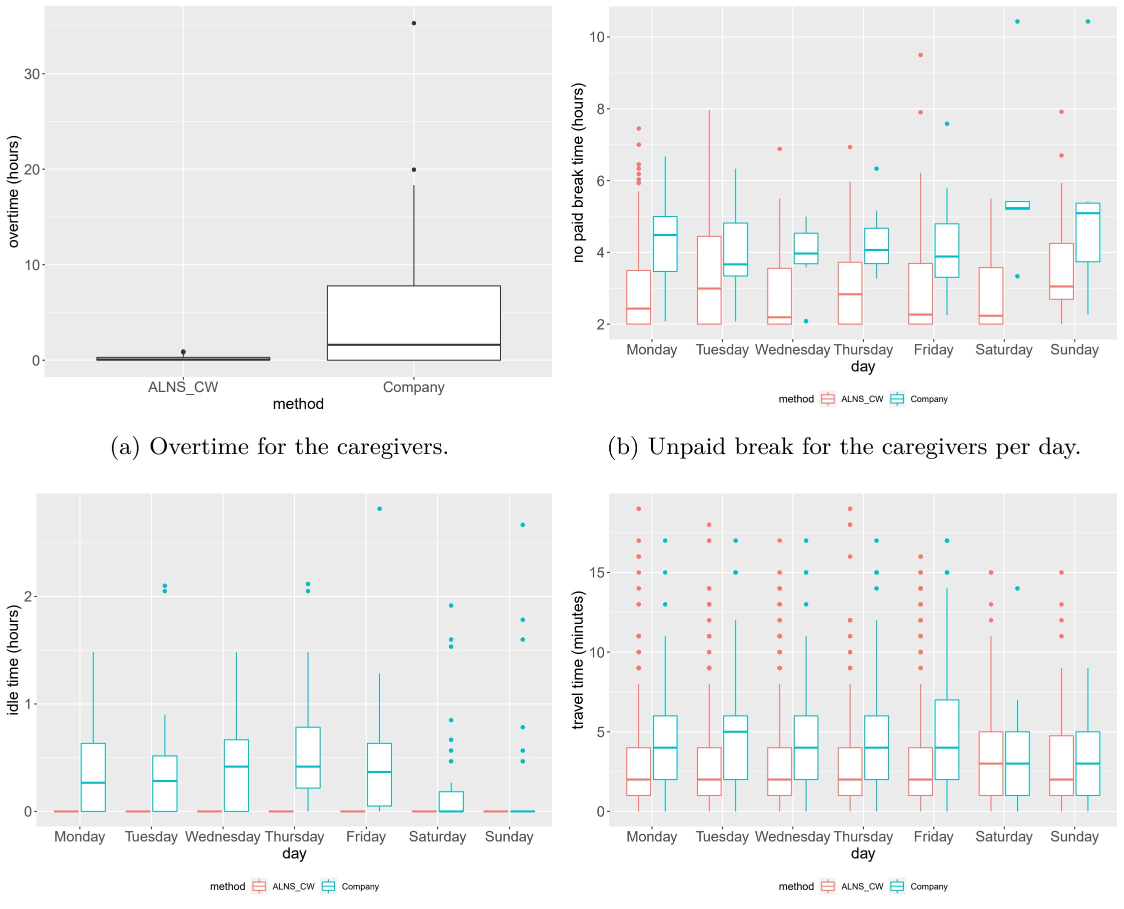


Figure 7.29: Mean weekly times per caregiver according to ALNS_CW (all weeks).



(a) Overtime for the caregivers.

(b) Unpaid break for the caregivers per day.

(c) Idle time for the caregivers per day.

(d) Travel time between services per day.

Figure 7.30: Overtime, unpaid break, idle time and travel time for week 6 (ALNS_CW).

A more detailed study of the overtime, the unpaid break, the idle time and the travel time for week 6 can be found in Figure 7.30. The comparison of the overtime for the caregivers is presented in Figure 7.30a, showing again the good performance of the algorithm. Then, Figures 7.30b to 7.30d make a comparison of the solutions on a daily basis. In all three cases the solutions found by the algorithm improve the ones of the company[6].

The results with respect to the affinity and the penalization on a weekly basis, both globally and on average per user, are presented in Table 7.18. It can be seen that the results of the company are better than the ones provided by ALNS_CW.

Week	Solution	Global			Per user	
		Welfare ($\times e8$)	Affinity ($\times e3$)	Penalization ($\times e3$)	Affinity	Penalization
1	Algorithm	2.45	2.78	12.87	3.22	15.93
	Company	3.22	3.64	11.55	4.14	13.73
2	Algorithm	2.56	2.91	13.51	3.30	18.09
	Company	3.48	3.94	11.93	4.50	14.94
3	Algorithm	2.59	2.98	13.31	3.33	16.96
	Company	3.66	4.19	11.89	4.66	14.66
4	Algorithm	2.44	2.87	13.54	3.27	17.59
	Company	3.44	4.02	11.53	4.62	15.19
5	Algorithm	1.36	2.15	7.97	3.36	13.63
	Company	1.82	2.86	9.27	4.60	16.10
6	Algorithm	2.20	2.67	13.74	3.23	18.25
	Company	2.97	3.58	11.60	4.25	15.39
7	Algorithm	2.63	2.99	13.51	3.41	18.48
	Company	3.68	4.17	11.96	4.62	14.60
8	Algorithm	1.89	2.58	10.20	3.35	15.05
	Company	2.56	3.48	10.16	4.64	15.03
9	Algorithm	2.03	2.69	11.74	3.36	17.95
	Company	2.71	3.58	10.52	4.44	15.21
10	Algorithm	2.67	3.04	13.32	3.38	14.96
	Company	3.66	4.15	11.66	4.59	14.49
11	Algorithm	2.46	2.93	12.60	3.32	16.01
	Company	3.45	4.09	11.29	4.64	14.50
12	Algorithm	2.31	2.82	12.10	3.32	17.02
	Company	3.22	3.92	10.53	4.65	14.66
13	Algorithm	2.97	3.20	14.54	3.36	16.45
	Company	4.09	4.38	13.54	4.67	15.83
14	Algorithm	2.74	3.12	14.22	3.40	16.46
	Company	3.85	4.35	12.32	4.75	14.20
15	Algorithm	2.84	3.19	13.26	3.46	13.75
	Company	3.97	4.44	11.77	4.71	13.70

Table 7.18: Computational results for real instances according to ALNS_CW (welfare).

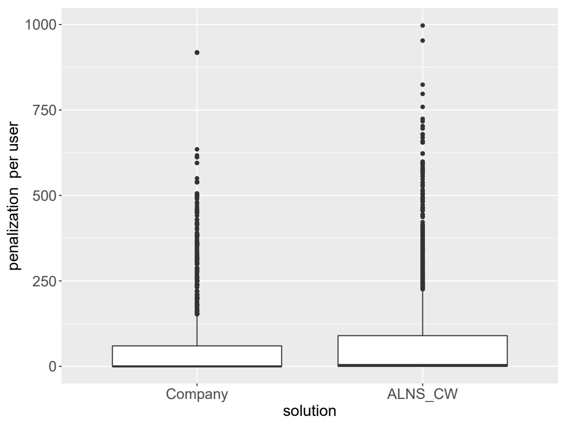
Week	Company	Algorithm	Week	Company	Algorithm
1	76.30%	27.39%	9	78.46%	22.52%
2	75.90%	17.50%	10	79.58%	20.41%
3	82.79%	18.99%	11	84.82%	18.62%
4	82.96%	19.35%	12	83.09%	18.80%
5	77.40%	22.59%	13	80.52%	21.78%
6	76.64%	16.42%	14	86.59%	22.81%
7	83.89%	22.37%	15	85.83%	21.51%
8	78.42%	20.39%			

Table 7.19: Weekly percentage of services with maximum affinity level (ALNS_CW).

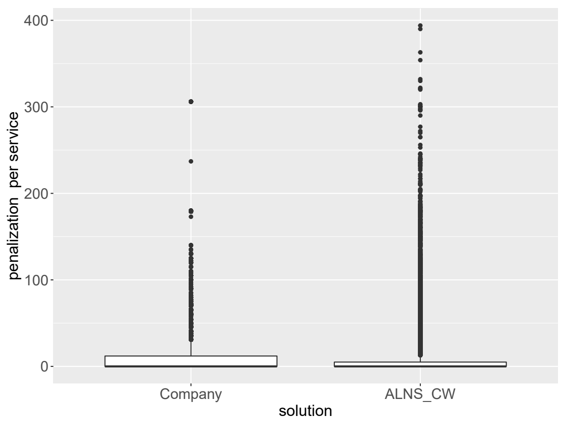
[6]Week 6 has been chosen to explain in more detail the behavior of the solution of the ALNS_CW and the one of the company in a typical week. The behavior of the solutions in the other weeks would be similar.

On the other hand, Table 7.19 shows the percentage of services where the best possible level of affinity is reached. The percentages of services that have the highest affinity for the company and the algorithm are 81.04% and 20.76%, respectively, when considering all the services across all the weeks. Therefore, the solution proposed by the company presents a much better behavior, which was to be expected, since the company tries to maintain the allocation of caregivers to users as much as possible.

Finally, the soft time window penalization for the users and the services in all the weeks is represented in the boxplots of Figures 7.31a and 7.31b. It can be seen that the company is better in terms of the penalization for the users. But, both solutions are closer regarding the penalization of the services, although the ALNS_CW solution presents more outliers. These results may be because, even though there is generally little penalization for the services, users tend to have some service with a penalty.



(a) For the users.



(b) For the services.

Figure 7.31: Soft time window penalization according to ALNS_CW (all weeks).

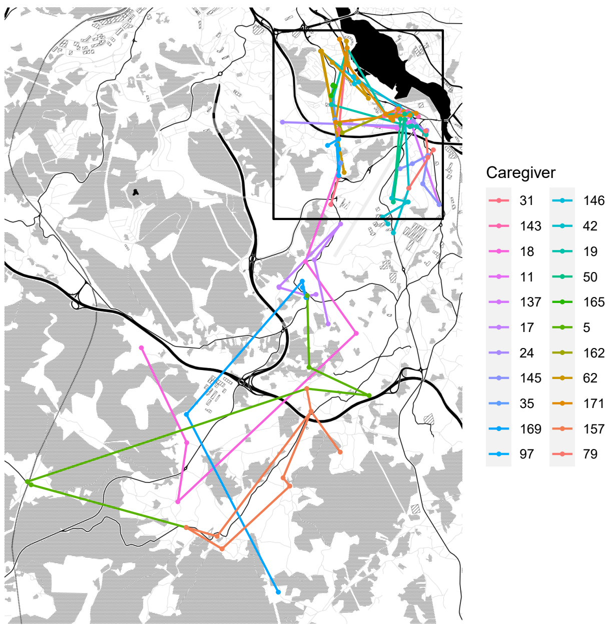
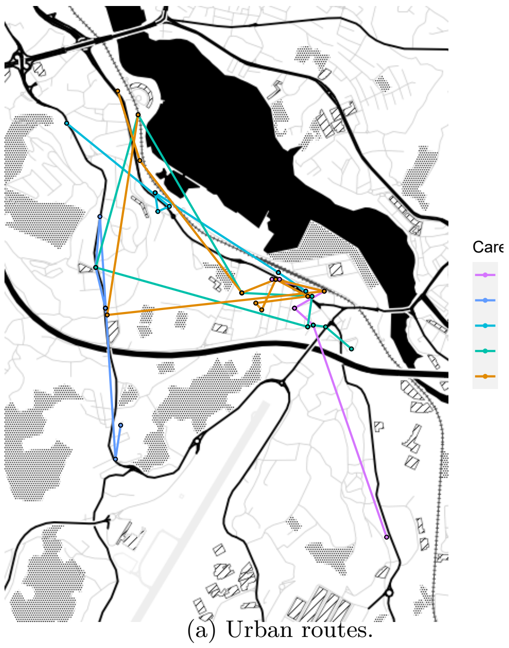
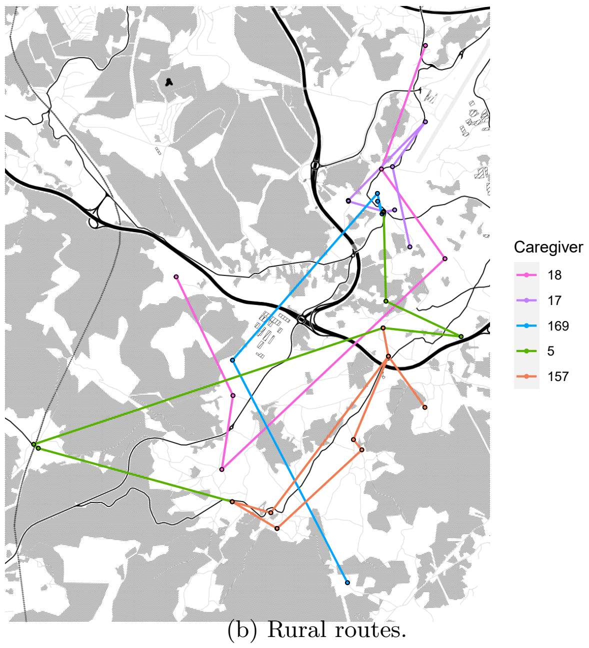


Figure 7.32: Routes for a random day (ALNS_CW).

Continuing with the analysis of the results, Figure 7.32 illustrates the routes that caregivers follow, according to the solutions given by the ALNS_CW, on a random day. It can be seen that the routes tend to be concentrated either in the urban[7] (see Figure 7.33a) or rural (see Figure 7.33b) areas, since there is only one route that connects rural and urban services. It means that the traveling times of caregivers are smaller than the ones of the routes presented in Figure 7.16, where caregivers usually travel between areas.



(a) Urban routes.

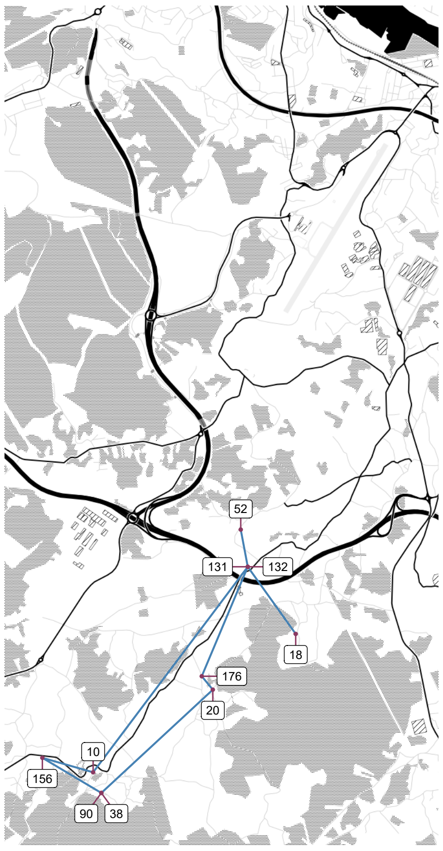


(b) Rural routes.

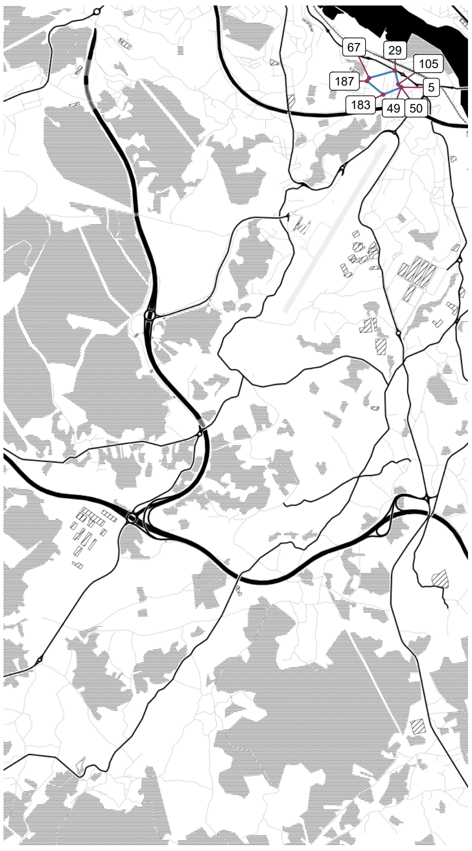
Figure 7.33: Routes separated into areas (ALNS_CW).

To analyze in more detail the solution, Figure 7.34 shows the routes of Caregiver 157 in four days (no services are scheduled on Thursday). On Monday and Wednesday, the caregiver works in the rural area, meanwhile, on Tuesday and Friday, she is assigned to urban services. This is different to what happened when the welfare was prioritized (see Figure 7.18), where the caregiver would stay in the same area and repeat several services throughout the week. The reason for such differences is that now the cost is being prioritized over welfare, meaning that routes do not have to be consistent in terms of the services performed by the caregivers (affinity levels). Instead, routes should minimize breaks and travel times.

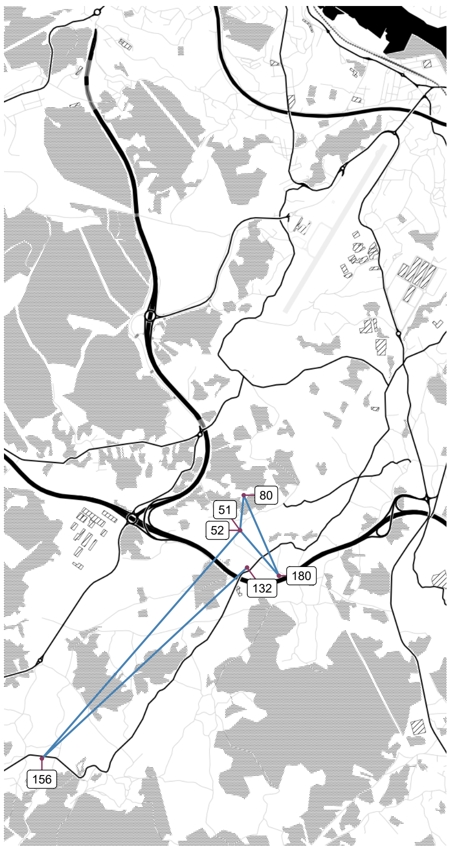
[7]The urban area is the one surrounded by the frame in Figure 7.32.



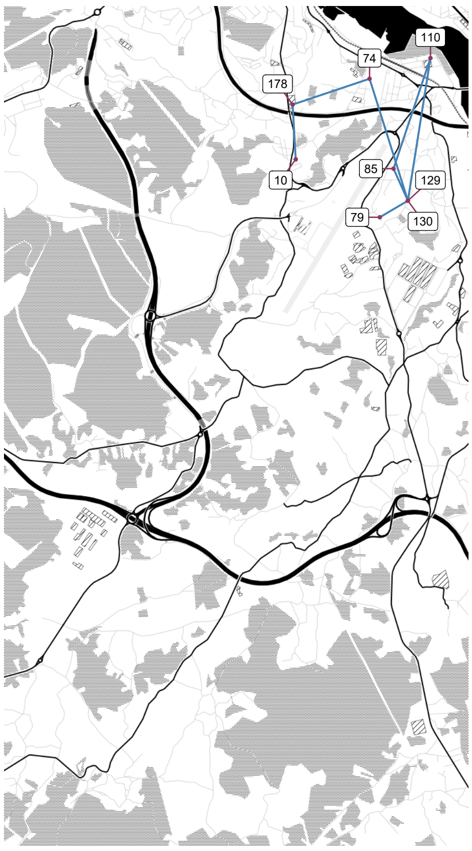
(a) Monday.



(b) Tuesday.



(c) Wednesday.



(d) Friday.

Figure 7.34: Routes of Caregiver 157.

Figure 7.35 shows the details of the schedule for the four days. Note that on Tuesday, Wednesday and Friday the caregiver only has one break per day, with a duration of two hours, which means that it will not considered as working time. According to the schedules, the soft time windows are not usually upheld (for example, on Monday, service 18 could be carried out at 9:00 and service 131 at 10:18, but this would result in a break with service 176). It can be seen that, when the services are close to each other (for example, see services 90 and 38 on Figure 7.34a), the travel time between them is neglected. The daily availability of the caregiver starts at 8:00 and ends at 22:30, which is respected by all her services. The caregiver works 10.53 hours on Monday, 10.13 hours on Tuesday, 7.38 hours on Wednesday and 8.01 on Friday, resulting in a total of 36.05 hours. These working times do not exceed the daily and weekly maximum working time imposed by the contract of the caregiver, which is of 12 and 40 hours, respectively.

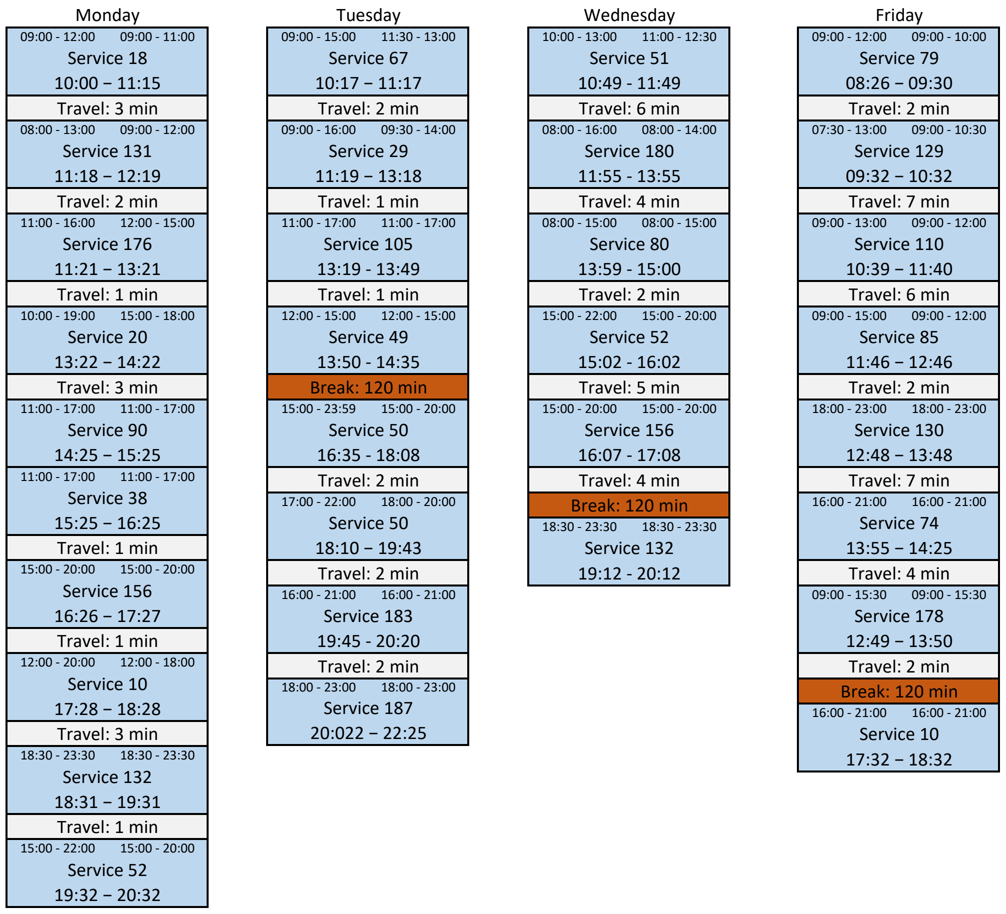


Figure 7.35: Schedules of caregiver 157.

7.4 Comparison of the two hierarchical solutions in the real case study

The real instances have been solved prioritizing the welfare over the cost, with algorithm ALNS_WC, and prioritizing the cost over the welfare, with algorithm ALNS_CW. Therefore, the results obtained will now be compared in order to analyze the trade-off between welfare and cost.

First, Table 7.20 presents the percentage of reduction of the proportion of services with maximum affinity, that is, how much the proportion is reduced when the instances are solved prioritizing the cost instead of prioritizing the welfare. It can be seen that this proportion is greatly reduced in all weeks, with a mean of 76.62%, which means that the number of services attended by their best caregiver decreases drastically.

Week	Algorithm
1	68.24%
2	79.41%
3	78.96%
4	77.55%
5	74.73%
6	80.91%
7	75.49%
8	76.41%
9	75.00%
10	76.73%
11	79.52%
12	78.80%
13	76.08%
14	75.03%
15	76.49%
Mean	76.62%

Table 7.20: Reduction of the percentage of services with maximum affinity.

Second, Table 7.21 presents the percentage of reduction of the objectives related with the welfare, i.e., how much the welfare, affinity and penalization[8] (global and per user) are reduced if the instances are solved with ALNS_CW instead of using ALNS_WC. The global welfare is reduced by 28.73% and the global affinity is decreased by 28.88%. In terms of the global penalization, it can be seen that it increases 429.05%, which means that the soft time window penalization obtained when the cost is prioritized is much greater than the one obtained prioritizing the welfare. Similarly, the affinity per user is reduced by 28.53% and the penalization per user increases by 468.53%.

	Global			Per user	
Week	Welfare	Affinity	Penalization	Affinity	Penalization
1	27.08%	27.03%	−452.36%	25.64%	−483.52%
2	28.89%	29.02%	−409.81%	29.18%	−555.43%
3	30.38%	30.37%	−471.24%	30.04%	−516.73%
4	29.07%	29.31%	−491.27%	29.83%	−453.14%
5	27.66%	28.09%	−333.15%	28.96%	−362.03%
6	27.39%	27.45%	−507.96%	26.09%	−599.23%
7	30.24%	30.30%	−413.69%	28.21%	−486.67%
8	27.86%	28.13%	−420.41%	29.18%	−502.00%
9	27.76%	27.88%	−410.43%	26.48%	−514.73%
10	28.23%	28.30%	−422.35%	27.31%	−410.58%
11	29.71%	29.90%	−307.77%	30.25%	−414.79%
12	29.14%	29.32%	−393.88%	29.81%	−423.69%
13	29.12%	29.36%	−361.59%	29.71%	−423.89%
14	29.38%	29.57%	−580.38%	29.17%	−537.98%
15	29.00%	29.11%	−459.49%	28.07%	−343.55%
Mean	28.73%	28.88%	−429.05%	28.53%	−468.53%

Table 7.21: Reduction of the welfare related objectives.

[8]Notice that, since the goal is to reduce the penalization, the percentages of the table related to this element are negative. Which means that the penalization increases when solving the instances with ALNS_CW instead of using ALNS_WC.

Finally, Table 7.22 contains the percentage of reduction of the objectives related to the cost, i.e., how much the objectives decrease when prioritizing the cost instead of the welfare. There is a reduction of the global cost of 12.05% and, when dividing the cost into overtime and worked time, the mean reduction is 87.56% and 3.66%, respectively. In terms of the values per caregiver, it can be seen that the overtime is reduced by 87.53%, the travel time by 55.66% and the idle time by 100%. Meanwhile, the unpaid break increases by 8.34%. The reason for this increment may be that this break is not considered as worked time.

	Global			Per caregiver			
Week	Cost	Overtime	Worked time	Overtime	Travel time	Break	Idle time
1	12.72%	100.00%	3.88%	100.00%	61.79%	−5.86%	100.00%
2	14.81%	100.00%	3.39%	100.00%	53.73%	−9.15%	100.00%
3	11.26%	49.21%	3.93%	48.57%	57.62%	10.77%	100.00%
4	11.74%	88.04%	3.00%	87.67%	43.94%	12.56%	100.00%
5	6.85%	100.00%	4.16%	100.00%	74.16%	−32.26%	100.00%
6	14.55%	95.30%	2.94%	94.15%	37.69%	−16.08%	100.00%
7	12.68%	67.93%	3.81%	67.63%	55.56%	3.37%	100.00%
8	9.08%	100.00%	4.40%	100.00%	74.07%	−25.09%	100.00%
9	12.59%	100.00%	3.21%	100.00%	50.00%	−4.81%	100.00%
10	10.98%	92.93%	3.19%	93.90%	50.70%	−7.88%	100.00%
11	11.14%	77.19%	3.26%	79.06%	54.55%	−16.89%	100.00%
12	11.79%	100.00%	4.43%	100.00%	63.36%	−6.34%	100.00%
13	13.42%	77.11%	4.00%	77.83%	59.48%	−15.64%	100.00%
14	12.02%	74.38%	3.63%	72.27%	45.07%	2.21%	100.00%
15	15.21%	91.39%	3.73%	91.87%	53.29%	−14.13%	100.00%
Mean	12.05%	87.56%	3.66%	87.53%	55.66%	−8.34%	100.00%

Table 7.22: Reduction of the cost related objectives.

Therefore, to reduce the cost by 12.05%, the welfare needs to be decreased by 28.73%. In particular, there is a 87.53% reduction of the overtime per caregiver, a 55.66% decrease in travel time, an increase of the unpaid break of 8.34% and a 100% reduction of idle time for the caregivers. In order to achieve this, the affinity per user is reduced by 28.53%, the penalization increases by 468.53% and the percentage of services with maximum affinity decreases by 76.62%.

7.4.1 Trade-off between soft time window penalization and cost

The previous results indicate that it is necessary to sacrifice welfare to improve the cost, but the continuity of care is usually one of the priority targets in HCSP. Therefore, the impact that the soft time window penalization has on the cost will be studied.

To this aim, for each service, a system of weights has been applied to relax the soft time window according to the difference between the hard and the soft time windows. Thus, a weight of 100% means that the original soft time windows are considered, a weight of 95% means that the soft time windows are expanded according to the 5% of the difference between the hard and the soft time window, and so on. In this way, the soft time windows are expanded, in increments of the 5% of the difference between the hard and the soft time window, until the hard time windows are reached. This relaxation will make possible to seek solutions for cost improvement.

The results are presented in Figures 7.36 and 7.37, which show the evolution of the affinity and the soft time window penalization with weights from 100% to 60%. For each weight, the ALNS_WC algorithm was executed. The last weight considered is 60% because the value 55% gives worse solutions (in terms of the soft time window penalization) than those of the company (although not in all weeks). As it can be seen in Figure 7.36, the affinity hardly changes, given the stability of the algorithm and the fact that the weights do not affect the affinity. However, as it was to be expected, the penalty increases as the weight decreases (see Figure 7.37).

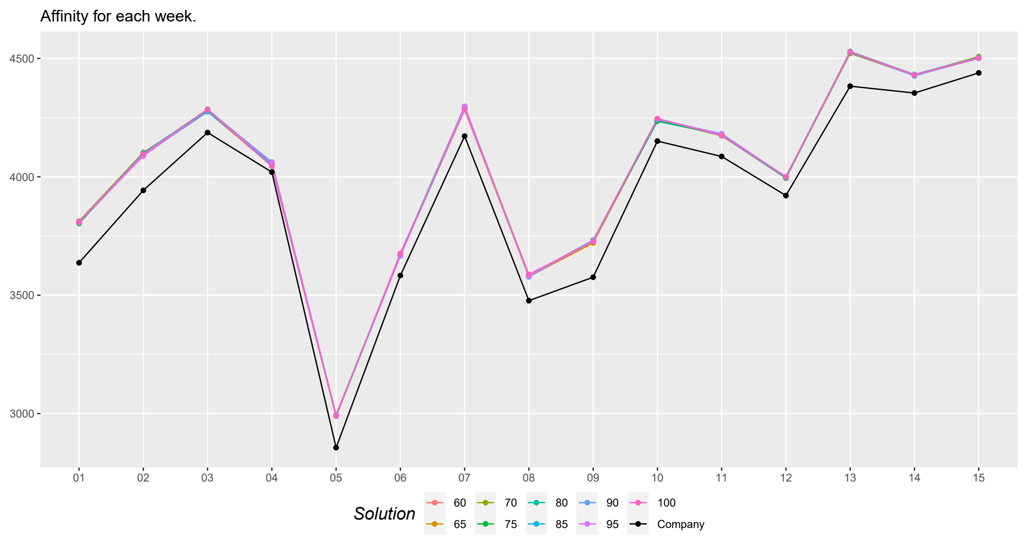


Figure 7.36: Affinity for different weights.

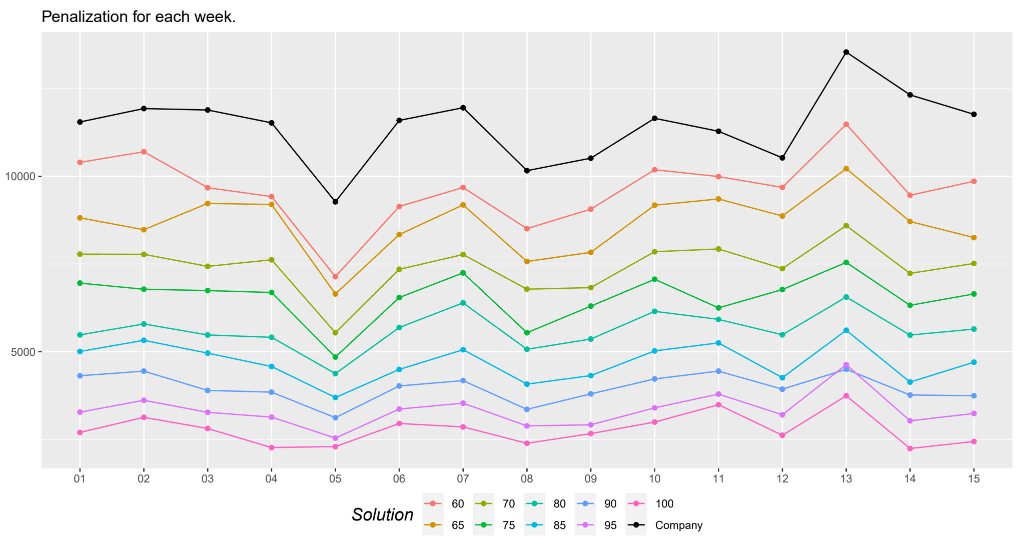


Figure 7.37: Soft time window penalization for different weights.

Figures 7.38 and 7.39 are used to study better the effect of the relaxation of the soft windows in the cost. Figure 7.38 presents the cost objective value obtained for each week using the different weight values and Figure 7.39 shows overtime, idle time, break time and travel time. Analyzing the figures it can be appreciated that, although the total cost is not drastically reduced, it is possible to reduce the overtime, idle time and break time while providing better solutions than the company as far as the weight decreases.

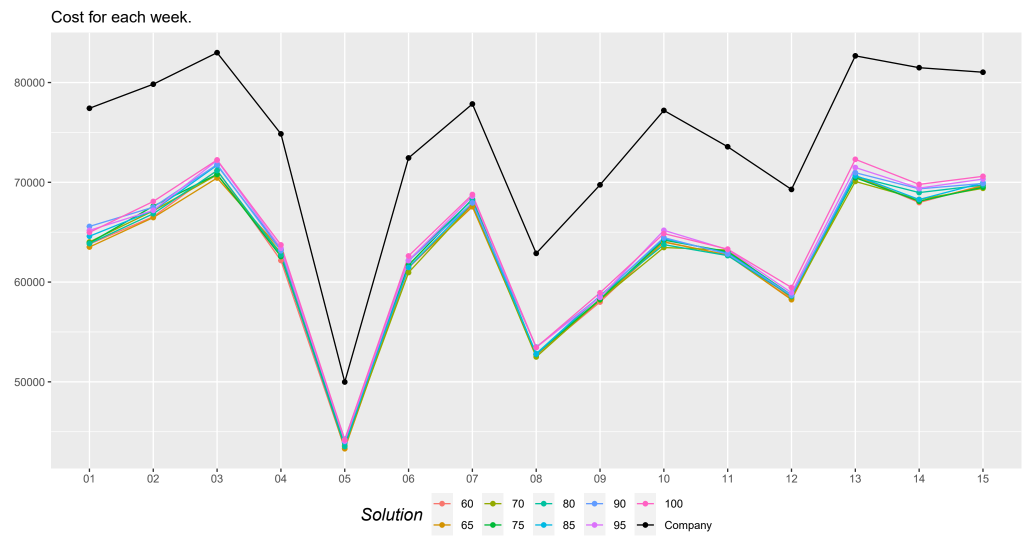


Figure 7.38: Cost for different weights.

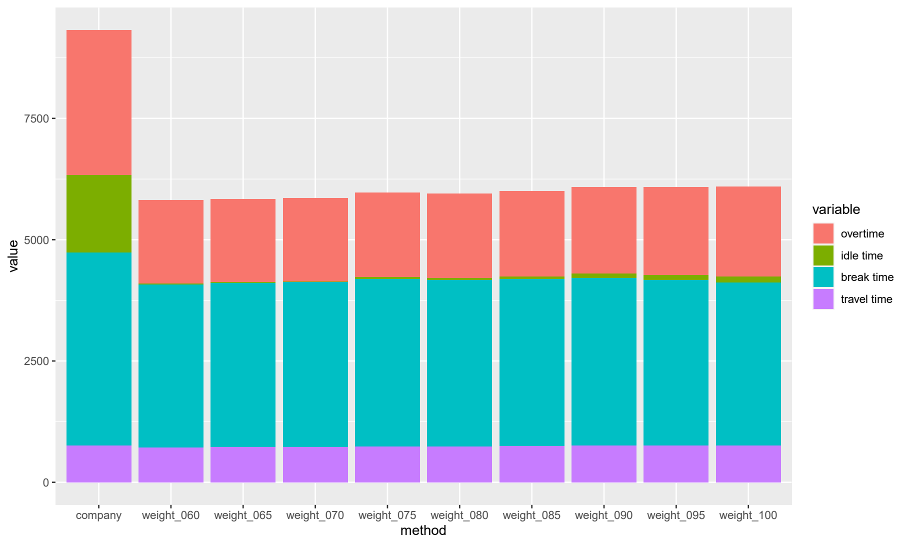


Figure 7.39: Decomposition of the cost for different weights.

7.5 Biobjective algorithm

This section details the computational results related to the biobjective version of the HCSP. Again, to study the performance of the methods, two types of different instances were considered: the Solomon instances and the real instances of the company.

The first results presented in this section are the ones obtained when solving the problem using the AUGMECON2 method. Then, a set of performance indicators is described, which will be used to compare the solutions obtained using the metaheuristic algorithm (BIALNS) with the ones found by the AUGMECON2 method. After that, a study of the BIALNS parameters is shown. Finally, several examples are used to illustrate the types pf Pareto frontiers that can appear.

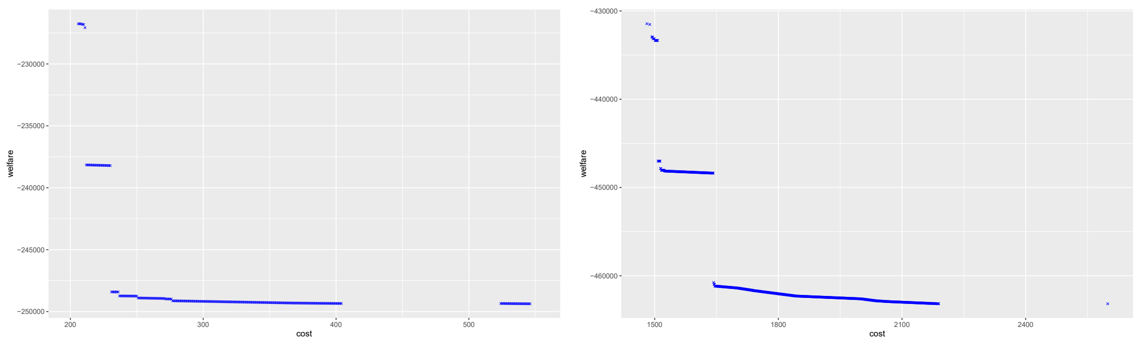
7.5.1 AUGMECON2 method

The AUGMECON2 method was used to solve the Solomon instances with 10 and 15 services. For this purpose, a time limit of 12 hours has been established to solve each lexicographical MILP with the optimization solver Gurobi, in order to obtain the range of the objective functions necessary to define the grid points. Then, the MILP associated to each grid point has been solved with Gurobi with a time limit of 1 hour. Table 7.23 presents the computational times, in hours, needed to solve each instance[9]. The fastest one is solved in 2.58 hours, while the slowest one needs 236.69 hours. Therefore, the AUGMECON2 method will not be suitable to solve large size instances. However, the results obtained with this method might be useful to evaluate the performance of the BIALNS algorithm.

Instance	01	02	03	04	05	06	07	08	09	10
10	4.92	2.58	7.23	20.39	5.22	5.83	7.08	5.13	14.99	3.58
15	77.68	36.67	13.05	58.97	86.51	236.69	143.44	124.00	164.72	90.29

Table 7.23: AUGMECON2 computational times (in hours).

Figure 7.40 presents an illustration of the Pareto frontier obtained for an instance of 10 (Figure 7.40a) and 15 (Figure 7.40b) services. It can be seen that the figures are defined in sections, which is due to the effect of the affinity on the welfare. In these instances, each segment of the frontier usually corresponds to a different value of the affinity and the breaks in the frontier are due to an abrupt change in the value of the affinity.



(a) Instance 10_06.

(b) Instance 15_01.

Figure 7.40: Pareto frontier.

7.5.2 Performance indicators

Since there are two different objectives involved in the problem, it is not immediate to evaluate the quality of a solution or to decide which solutions are the best ones. Therefore, to compare the solutions of the biobjective problem, a set of performance indicators are used. The formal definition of the indicators is presented in Table 7.24 and they measure the convergence and the diversity of the solutions. To introduce them, it is necessary to consider an approximation of the Pareto frontier, A, and a reference set, RF. The set A is the solution of the problem that has to be evaluated (for example, an approximation of the Pareto frontier obtained using BIALNS). Meanwhile, the reference set RF is composed exclusively of non dominated points (since this set is usually not known a priori, a common approach is to define RF by selecting the non dominated points of the solutions that are being evaluated by the indicator).

[9]Notice that, because of the time limit considered, there is no guarantee that the Pareto frontier given by the AUGMECON2 method is the optimal one.

The indicators studied are the following ones.

Coverage. It represents the percentage of elements of A dominated by RF. The smaller the value of this measure, the better the quality of the approximation of the Pareto front.

Generational Distance. It measures how far are the elements of A from those of RF. This indicator is obtained using d_i, which is the euclidean distance between the elements $i \in A$ and the nearest one of RF. The smaller the value of this measure, the closer A is to RF.

Inverted Generational Distance. This variant of GD measures how far are the elements of the reference set RF to the set A. Now, it is done taken the euclidean distance between the element $i \in RF$ and the nearest one of A, represented by $\tilde{d_i}$. For this indicator, smaller values are preferred because it means that RF is close to the approximation A.

Epsilon. It computes the minimum distance needed to translate every element of A so it dominates the solution RF. It is said that $x \succ_\epsilon y$ if, for each objective $k \in \{1, ..., p\}$, $f_k(x) < \epsilon + f_k(y)$. If the value of this indicator is small, all the elements of solution A are close to the solution RF, because it is necessary to translate them a small distance in order to achieve the dominance of RF. Therefore the smaller the value, the better the quality of the approximation of the Pareto front.

All of these indicators, except the coverage, are implemented in the jMetal framework, described in Durillo & Nebro (2011).

Indicator	**Formula**
Coverage (CV)	$CV(RF, A) = \dfrac{\lvert\{x \in A : \exists y \in RF / y \succ x\}\rvert}{\lvert A\rvert}$
Generational Distance (GD)	$GD(RF, A) = \dfrac{\sqrt{\sum_{i=1}^{\lvert A\rvert} d_i^2}}{\lvert A\rvert}$
Inverted Generational Distance (IGD)	$IGD(RF, A) = \dfrac{\sqrt{\sum_{i=1}^{\lvert RF\rvert} \tilde{d}_i^2}}{\lvert RF\rvert}$
Epsilon (EPS)	$EPS(RF, A) = \inf_{\epsilon \in \mathbb{R}} \{\forall y \in RF\ \exists x \in A : x \succ_\epsilon y\}$

Table 7.24: Performance indicators for the biobjective problem.

7.5.3 BIALNS method

As far as the configuration of the biobjective algorithm is concerned, it is necessary to set multiple parameters. Below, the values of the parameters fixed at each stage of the BIALNS are indicated.

Step 1: Initialize the sets. In the first step it is necessary to establish the parameters for each lexicographic ALNS method. The values used are the ones giving good results, in reasonable computational times, during the study presented in Sections 7.2 and 7.3:

10 services. Number of iterations (n): 1000. Proportion of solution to destroy (p): *auto*_100%.

15 services. Number of iterations (n): 1000. Proportion of solution to destroy (p): *auto*_100%.

25 services. Number of iterations (n): 1000. Proportion of solution to destroy (p): *auto*_75%.

50 services. Number of iterations (n): 1000. Proportion of solution to destroy (p): *auto*_25%.

100 services. Number of iterations (n): 1000. Proportion of solution to destroy (p): *auto*_25%.

Step 2: Generate different solutions. This step deals with the parameters related to the stopping criteria, as well as the parameters for each lexicographic ALNS method.

Stopping criteria. Number of iterations (nroutes): 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000 and 10000.

ALNS. Number of iterations (nalns): 5, 10, 25 and 50. Proportion of solution to destroy (pr): 5%, 10% and 25%.

Step 3: Get non dominated solutions. In the third step, it is necessary to fix the stopping criteria parameter.

Stopping criteria. Number of iterations (nsols): 1×10^5, 2×10^5 and 3×10^5.

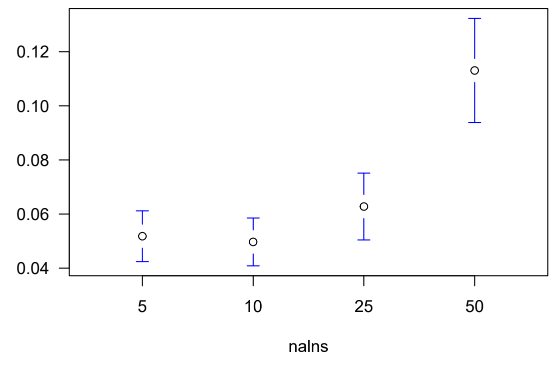
Next, a computational analysis is presented to study the behavior of the BIALNS with respect to its different parameters, as well as their most suitable values for steps 2 and 3. For 10 and 15 services, the solutions found by the algorithm were compared to the ones obtained with the AUGMECON2 method using the performance indicators. Since for 25 and 50 services there are no solutions obtained with the AUGMECON2 method, the solutions given by the BIALNS will be compared with a reference set (which was obtained, for each instance, by combining all the solutions found and keeping only the non dominated points).

7.5.3.1 ALNS parameters for step 2: generation of different solutions

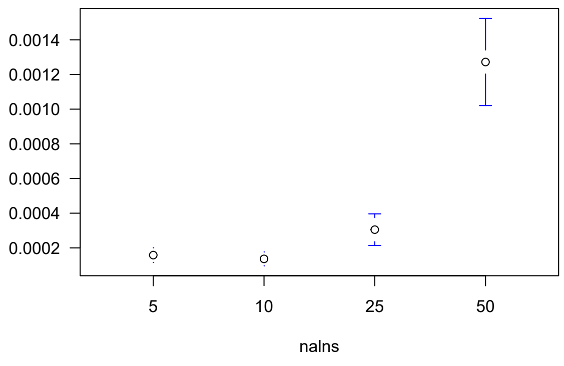
To analyze the parameters of the ALNS (number of iterations and proportion of solution to destroy), the instances of 10 services were solved five times for each combination of parameters[10].

The 95% confidence intervals of each indicator, obtained when solving the instances using each number of iterations (nalns), are shown in Figure 7.41. It can be seen that, for all indicators, big values of the nalns result in worse solutions. Specifically, similar values of CV, GD and IGD are found setting the parameter to be 5 or 10.

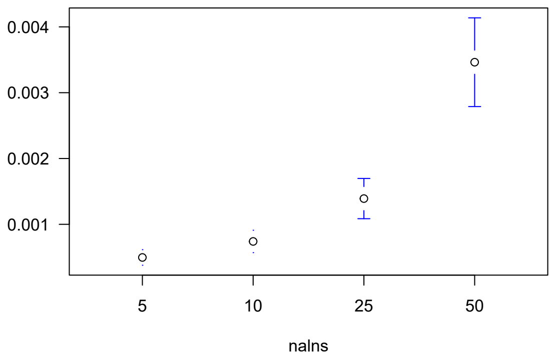
[10]The value of nroutes considered in this analysis is 1000, this parameter will be studied in more detail later on.



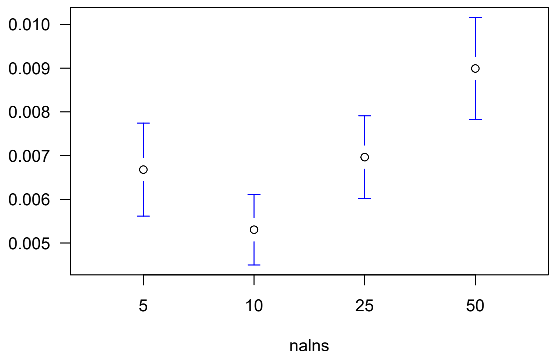
(a) Coverage.



(b) Generational Distance.

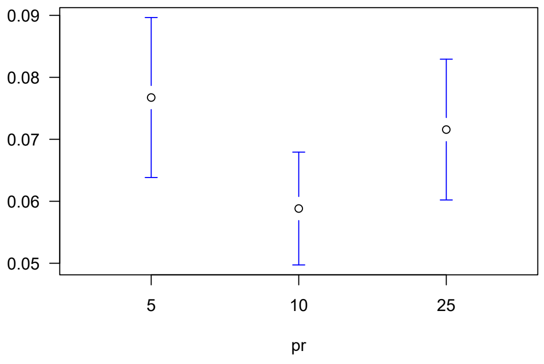


(c) Inverted Generational Distance.

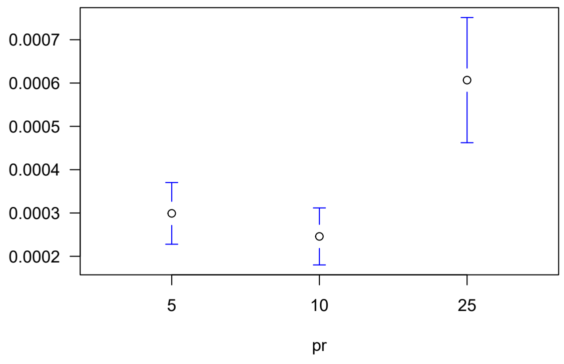


(d) Epsilon.

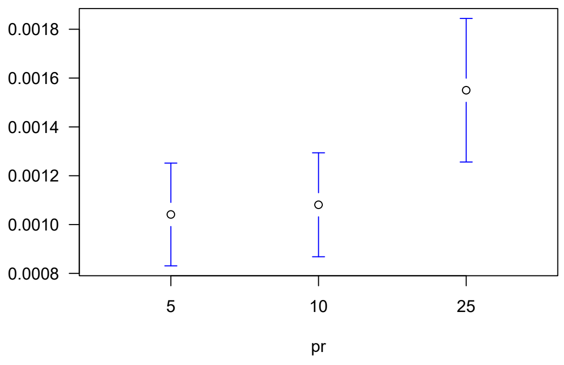
Figure 7.41: Confidence intervals of the indicators in terms of nalns.



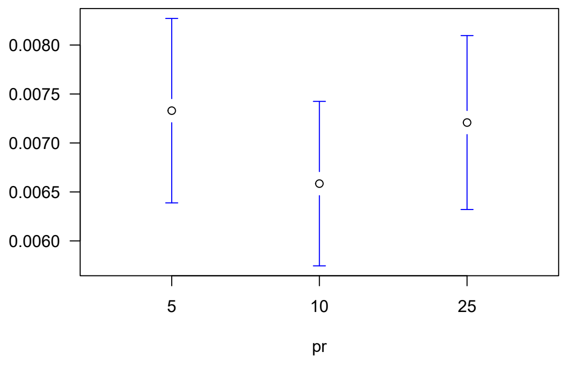
(a) Coverage.



(b) Generational Distance.



(c) Inverted Generational Distance.



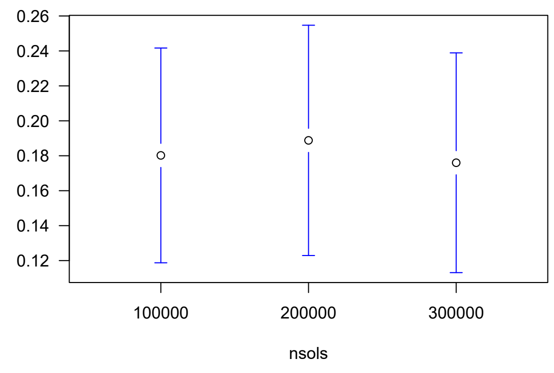
(d) Epsilon.

Figure 7.42: Confidence intervals of the indicators in terms of pr.

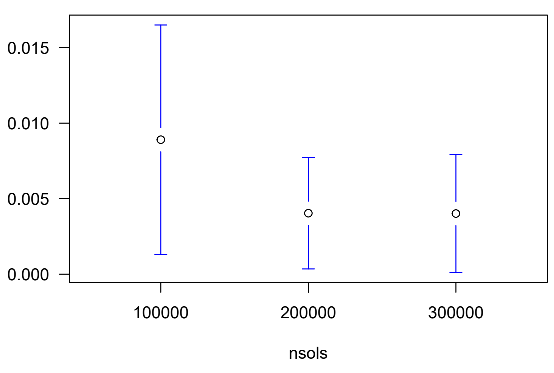
Figure 7.42 presents the 95% confidence intervals of the indicators for the different proportion of solution to destroy (pr). As it has already happened with nalns, setting big values for the proportion results in worse solutions. But it is not clear which configuration is the best, because the behavior of proportions 5% and 10% is indicator dependant: 5% is better for IGD (see Figure 7.42c), 10% is better in terms of GD (see Figure 7.42d) and CV (see Figure 7.42a), and both are similar for EPS (see Figure 7.42d).

7.5.3.2 Number of iterations in step 3: get non dominated solutions

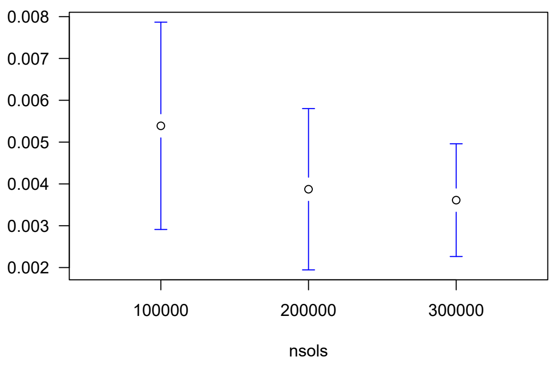
The values of the indicators, according to the number of iterations when obtaining non dominated solutions (nsols), are shown in Figure 7.43. The three configurations seem to be similar in terms of CV (see Figure 7.43a) and EPS (see Figure 7.43d), but 2×10^5 and 3×10^5 iterations result in better values of GD (see Figure 7.43b) and IGD (see Figure 7.43c).



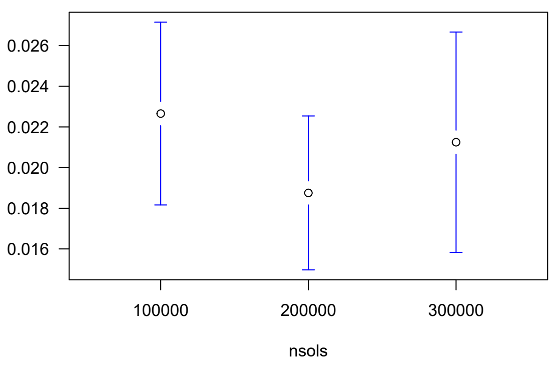
(a) Coverage.



(b) Generational Distance.



(c) Inverted Generational Distance.



(d) Epsilon.

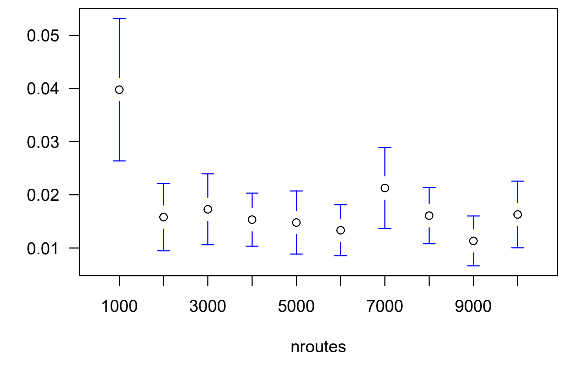
Figure 7.43: Confidence intervals of the indicators in terms of nsols.

7.5.3.3 Number of iterations in step 2: generation of different solutions

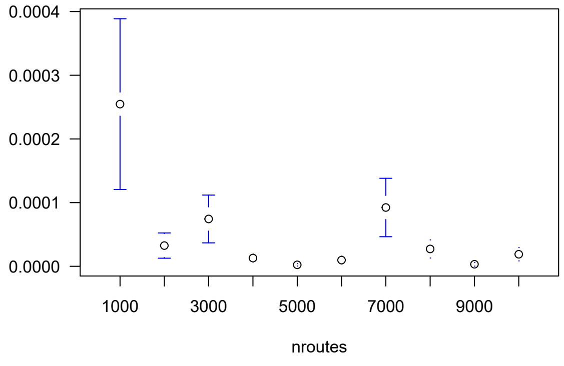
In a preliminary study it was found that this parameter is one of the most relevant when solving the problem, as it highly affects the quality of the solutions. Therefore, the BIALNS was used to solve the instances with 10, 15, 25 and 50 services, with the other parameters fixed. The algorithm was run five times for all possible values of the parameter, while setting nalns=5, pr=5% and nsols=3×10^5.

Results of instances with 10 services

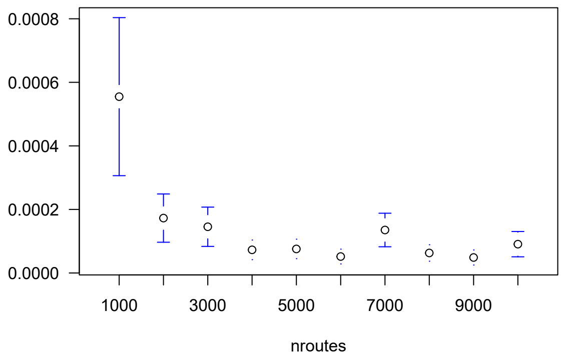
Analyzing the 95% confidence intervals presented in Figure 7.44, it can be concluded that choosing a number of iterations bigger than 6000 would not guarantee better results for instances with 10 services. Furthermore, according to some indicators, like CV and EPS, the quality of the solutions found with 7000 and 8000 decreases.



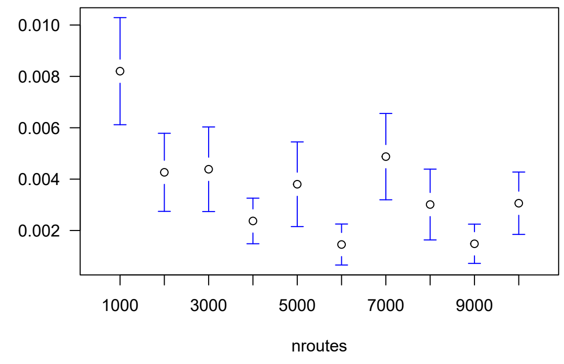
(a) Coverage.



(b) Generational Distance.



(c) Inverted Generational Distance.



(d) Epsilon.

Figure 7.44: Confidence intervals of the indicators in terms of nroutes (10 services).

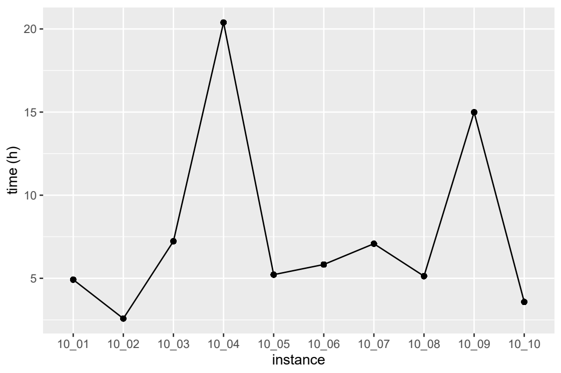
The results obtained are presented in Table 7.25, which shows the indicator values, as well as the number of non dominated (NDP) and total (TP) points found by the AUGMECON2 method and the BIALNS algorithm[11].

In this case, no method performs better than the others. Instead, the behavior of the methods is instance dependant: AUGMECON2 has better results for instances 10_01, 10_02, 10_06, 10_07 and 10_08, meanwhile the BIALNS is better for instances 10_03, 10_04, 10_05 and 10_09. Finally, for instance 10_10 both methods behave similarly.

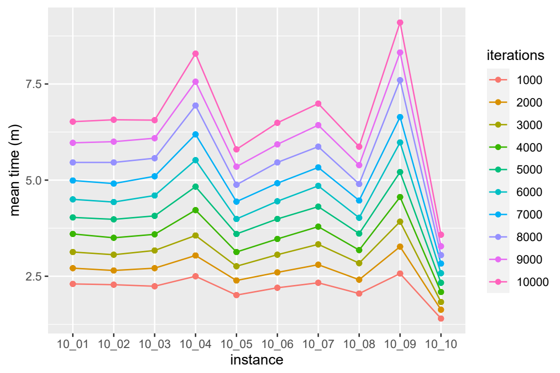
[11]Notice that, for each instance, the results obtained for the 5 runs of the BIALNS are shown. The best values, for each indicator and run, are highlighted in bold to easily compare the methods.

	CV		EPS		GD		IGD		NDP		TP	
Instance	AUGM	ALG	AUGM	ALG	AUGM	ALG	AUGM	ALG	AUGM	ALG	AUGM	ALG
	0	2.98e-2	**0**	4.10e-3	**0**	5.73e-5	**0**	1.03e-4	**237**	228	**237**	235
	0	4.27e-3	**0**	4.35e-5	**0**	6.19e-8	**0**	9.12e-5	**237**	233	**237**	234
10_01	**0**	2.11e-2	**0**	1.09e-3	**0**	4.78e-5	**0**	4.02e-5	**237**	232	**237**	**237**
	0	2.55e-2	**0**	9.68e-3	**0**	1.43e-4	**0**	2.47e-4	**237**	229	**237**	235
	0	2.55e-2	**0**	4.20e-3	0	4.04e-5	**0**	1.28e-4	**237**	229	**237**	235
	0	3.13e-1	**2.64e-5**	7.22e-3	**0**	6.25e-5	**1.34e-5**	5.21e-4	**268**	182	**268**	265
	0	3.70e-3	**2.64e-5**	**2.64e-5**	**0**	9.78e-8	2.67e-5	**9.78e-8**	**268**	269	268	**270**
10_02	**0**	1.50e-2	**2.64e-5**	1.37e-3	**0**	5.56e-6	**2.67e-5**	5.90e-5	**268**	262	**268**	266
	0	2.22e-2	**2.64e-5**	1.37e-3	**0**	6.16e-6	2.67e-5	**6.16e-6**	**268**	264	268	**270**
	0	1.49e-2	**2.64e-5**	1.37e-3	**0**	5.40e-6	**2.67e-5**	1.88e-5	**268**	265	268	**269**
	0	**0**	2.37e-5	**0**	**0**	**0**	3.38e-4	**0**	98	**102**	98	**102**
	0	**0**	7.12e-5	**0**	**0**	**0**	4.18e-4	**0**	98	**103**	98	**103**
10_03	**0**	**0**	2.37e-5	**0**	**0**	**0**	3.38e-4	**0**	98	**102**	98	**102**
	0	**0**	7.12e-5	**0**	**0**	**0**	4.18e-4	**0**	98	**103**	98	**103**
	0	**0**	7.12e-5	**0**	**0**	**0**	4.18e-4	**0**	98	**103**	98	**103**
	8.33e-2	**0**	6.52e-2	**0**	1.62e-2	**0**	3.05e-2	**0**	11	**13**	12	**13**
	8.33e-2	**0**	6.52e-2	**0**	1.62e-2	**0**	3.05e-2	**0**	11	**13**	12	**13**
10_04	8.33e-2	**0**	6.52e-2	**0**	1.62e-2	**0**	3.05e-2	**0**	11	**13**	12	**13**
	8.33e-2	**0**	6.52e-2	**0**	1.62e-2	**0**	3.05e-2	**0**	11	**13**	12	**13**
	8.33e-2	**0**	6.52e-2	**0**	1.62e-2	**0**	3.05e-2	**0**	11	**13**	12	**13**
	0	**0**	4.73e-5	**0**	**0**	**0**	1.57e-4	**0**	50	**51**	50	**51**
	0	**0**	4.73e-5	**0**	**0**	**0**	1.57e-4	**0**	50	**51**	50	**51**
10_05	**0**	**0**	4.73e-5	**0**	**0**	**0**	1.57e-4	**0**	50	**51**	50	**51**
	0	**0**	4.73e-5	**0**	**0**	**0**	1.57e-4	**0**	50	**51**	50	**51**
	0	**0**	4.73e-5	**0**	**0**	**0**	1.57e-4	**0**	50	**51**	50	**51**
	0	9.01e-2	**0**	2.06e-2	**0**	7.12e-4	**0**	6.12e-4	**222**	202	**222**	**222**
	0	9.01e-3	**0**	8.85e-5	**0**	5.98e-7	**0**	5.98e-7	**222**	220	**222**	**222**
10_06	**0**	7.80e-2	**0**	4.71e-2	**0**	1.84e-3	**0**	1.97e-3	**222**	201	**222**	218
	0	1.36e-2	**0**	8.85e-5	**0**	8.01e-7	**0**	1.40e-5	**222**	218	**222**	221
	0	2.75e-2	**0**	2.94e-3	**0**	1.55e-5	**0**	1.09e-4	**222**	212	**222**	218
	0	2.78e-2	**1.47e-4**	5.16e-4	**0**	1.02e-5	1.37e-4	**7.88e-5**	**107**	105	107	**108**
	0	8.26e-2	**1.47e-4**	7.46e-3	**0**	1.60e-4	**1.37e-4**	1.80e-4	**107**	100	107	**109**
10_07	**0**	9.43e-3	**1.47e-4**	1.48e-2	**0**	7.04e-5	**1.37e-4**	4.36e-4	**107**	105	107	**106**
	0	9.26e-2	**1.47e-4**	1.49e-2	**0**	5.89e-4	**1.37e-4**	6.71e-4	**107**	98	107	**108**
	0	2.78e-2	**1.47e-4**	4.20e-3	**0**	4.43e-5	1.37e-4	**1.12e-4**	**107**	105	107	**108**
	0	4.47e-2	**0**	2.97e-4	**0**	7.48e-6	**0**	4.64e-5	**181**	171	**181**	179
	0	4.44e-2	**0**	7.14e-3	**0**	1.37e-4	**0**	1.66e-4	**181**	172	**181**	180
10_08	**0**	6.08e-2	**0**	7.14e-3	**0**	1.41e-4	**0**	1.50e-4	**181**	170	**181**	**181**
	0	4.49e-2	**0**	1.19e-3	**0**	7.52e-6	**0**	6.69e-5	**181**	170	**181**	178
	0	3.33e-2	**0**	7.44e-4	**0**	1.24e-5	**0**	3.18e-5	**181**	174	**181**	180
	1.28e-2	2.56e-2	**1.28e-2**	**1.28e-2**	**2.15e-4**	**3.17e-4**	6.26e-4	**2.31e-4**	**77**	76	**78**	**78**
	1.28e-2	**1.28e-2**	1.28e-2	**1.21e-4**	2.15e-4	**1.55e-6**	6.26e-4	**1.55e-6**	**77**	**77**	**78**	**78**
10_09	**1.28e-2**	1.30e-2	**1.28e-2**	**1.28e-2**	2.15e-4	**1.57e-6**	6.26e-4	**2.87e-4**	**77**	76	**78**	77
	1.28e-2	2.56e-2	1.28e-2	**1.22e-2**	2.15e-4	**1.58e-4**	6.26e-4	**1.58e-4**	**77**	76	**78**	**78**
	1.28e-2	2.56e-2	1.28e-2	**1.21e-4**	2.15e-4	**3.11e-6**	6.26e-4	**3.11e-6**	**77**	76	**78**	**78**
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**6**	**6**	**6**	**6**
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**6**	**6**	**6**	**6**
10_10	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**6**	**6**	**6**	**6**
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**6**	**6**	**6**	**6**
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**6**	**6**	**6**	**6**

Table 7.25: Indicator values for 6000 iterations (10 services).



(a) AUGMECON2 time (hours).



(b) BIALNS time (minutes).

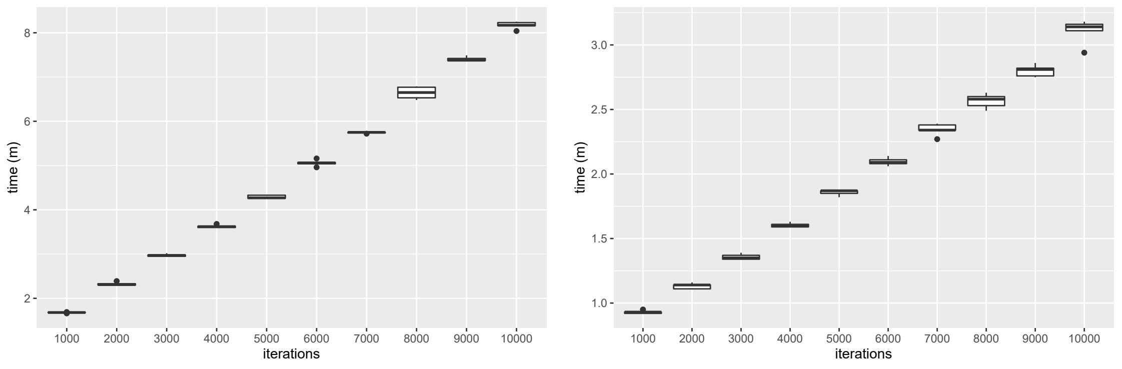
Figure 7.45: Computational times (10 services).

Figure 7.45 and Table 7.26 show the computational times[12] after solving the instances with 10 services. The AUGMECON2 method needs more than 4 hours to solve most of the considered instances, meanwhile the BIALNS solves all of them in less than 10 minutes. Interestingly, the comparison of the two figures shows that the most time-consuming instances for the AUGMECON2 method (10_04 and 10_09) are also the slowest ones for the algorithm.

	AUGMECON2	Algorithm mean time (minutes)									
Instance	(hours)	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
10_01	4.92	2.30	2.71	3.13	3.60	4.03	4.50	4.99	5.46	5.97	6.52
10_02	2.58	2.28	2.65	3.06	3.50	3.98	4.43	4.91	5.46	6.00	6.57
10_03	7.23	2.24	2.71	3.17	3.59	4.07	4.60	5.10	5.57	6.09	6.56
10_04	20.39	2.50	3.04	3.56	4.22	4.83	5.52	6.19	6.94	7.56	8.29
10_05	5.22	2.01	2.39	2.76	3.13	3.60	3.99	4.44	4.88	5.35	5.80
10_06	5.83	2.20	2.60	3.06	3.47	3.99	4.45	4.92	5.46	5.93	6.49
10_07	7.08	2.33	2.80	3.33	3.79	4.31	4.85	5.33	5.87	6.43	6.99
10_08	5.13	2.05	2.41	2.84	3.18	3.61	4.02	4.47	4.90	5.39	5.87
10_09	14.99	2.57	3.27	3.92	4.56	5.21	5.98	6.64	7.60	8.32	9.10
10_10	3.58	1.40	1.63	1.83	2.09	2.33	2.58	2.83	3.05	3.28	3.58

Table 7.26: Computational time for instances with 10 services.

Note that the first step of the BIALNS mainly consists in solving the hierarchical versions of the problem, whose computational study has already being studied in previous sections. For this reason, it is interesting to study the aggregated time that the BIALNS needs to generate different solutions (step 2) and to obtain non dominated points (step 3). Figure 7.46 presents a boxplot of the computational time associated with steps 2 and 3 of the algorithm, for two instances. According to the figures it is clear that the BIALNS is consistent, in the sense that the time needed to solve an instance (for each number of iterations) is similar in every run. Notice that, since the behavior of the remaining instances is very similar to the ones presented in Figure 7.46, they are omitted to avoid redundancy.



(a) Instance 10_09.

(b) Instance 10_10.

Figure 7.46: Computational time of steps 2 and 3 (10 services).

The study finished with the comparison of the number of total and non dominated points found by both methods, represented in Figure 7.47. The values are obtained considering, for each instance, the best and worst solutions obtained by the BIALNS algorithm. Given an instance, the best solution is the one that has the largest number of non dominated points, when comparing it to the AUGMECON2 solution. In a similar way, the worst solution would be the one with the minimum number of non dominated points.

[12]Notice that the BIALNS algorithm was run 5 times. Therefore, the figure presents the mean time employed to solve each instance for each value of the parameter.

In terms of the total number of points, the worst and best solutions are very similar. According to the non dominated points, the best solution is as good as the AUGMECON2 one while in the worst case BIALNS finds considerably fewer points than the AUGMECON2 solution.

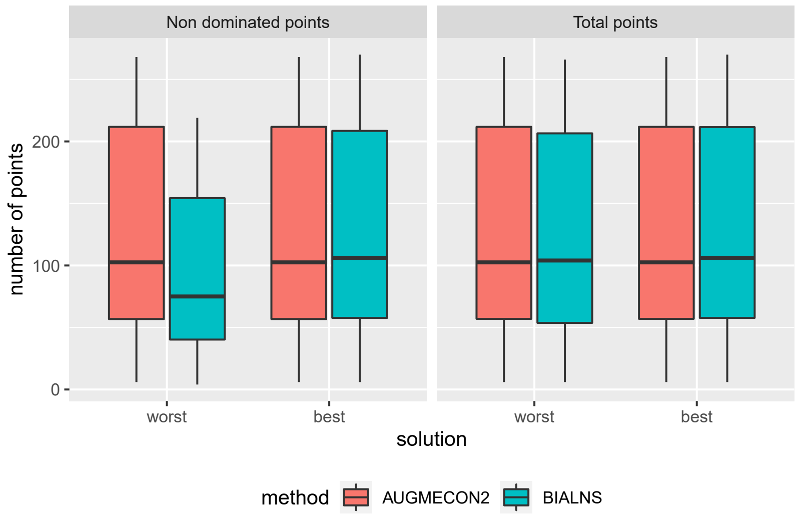


Figure 7.47: Number of non dominated and total points (10 services).

Figure 7.48 shows the number of non dominated points for different values of the iterations of the BIALNS. It can be observed that, when the best solution is chosen, the BIALNS is very competitive with AUGMECON2 in terms of the number of non dominated points. However, this is not the case of the worse solution. Specifically, the number of non dominated points for the worst solution increases with the iterations, reaching its best value from 2000 onwards.

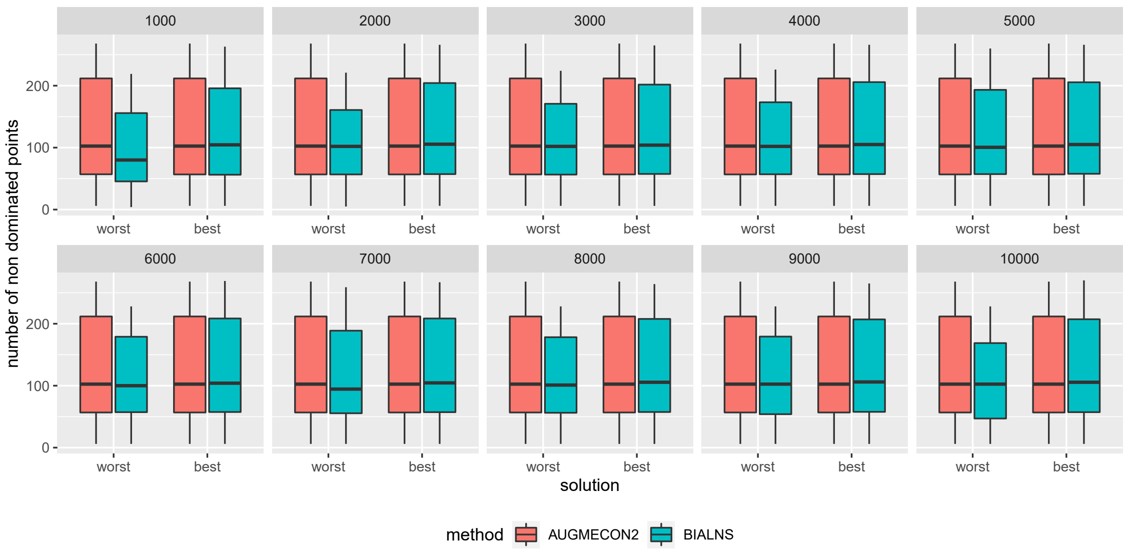
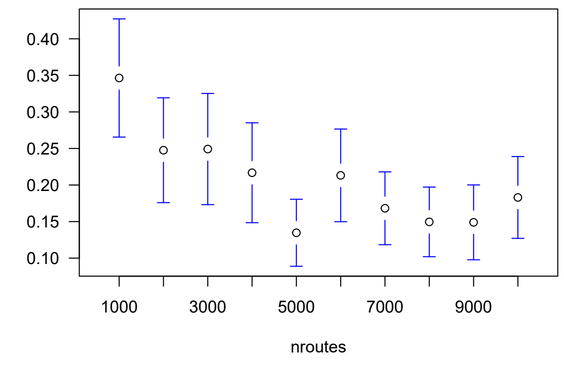


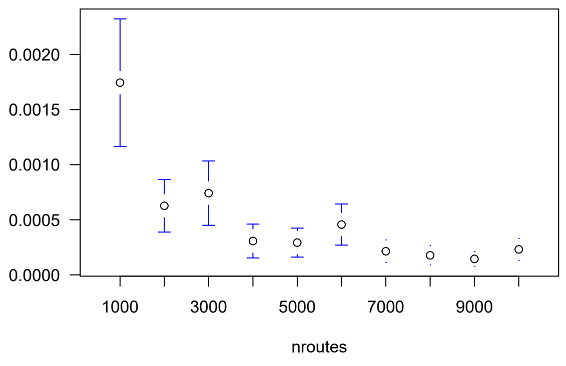
Figure 7.48: Number of non dominated points by iterations (10 services).

Results of instances with 15 services

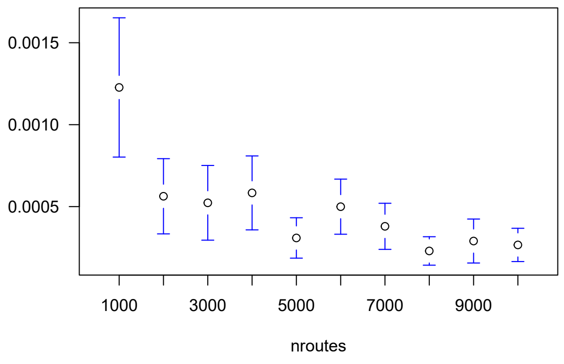
Figure 7.49 presents the indicator values obtained for instances with 15 services. It can be seen that the quality of the solution improves as the number of iterations increases. For these instances, an adequate number of iterations could be 5000, because it presents good values for all indicators and its increment would not necessary improve the results. Additionally, another configuration that shows good results is 8000, which slightly improves the values of some of the indicators.



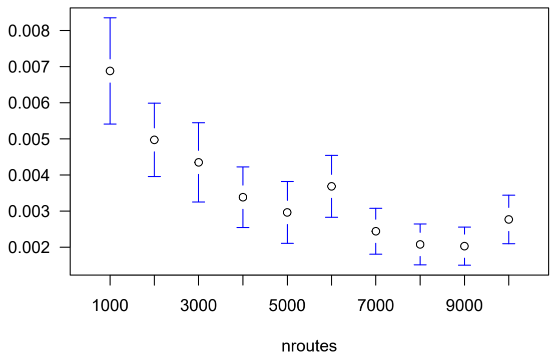
(a) Coverage.



(b) Generational Distance.



(c) Inverted Generational Distance.



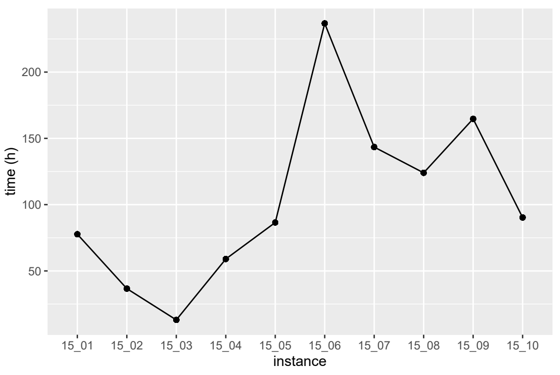
(d) Epsilon.

Figure 7.49: Confidence intervals of the indicators in terms of nroutes (15 services).

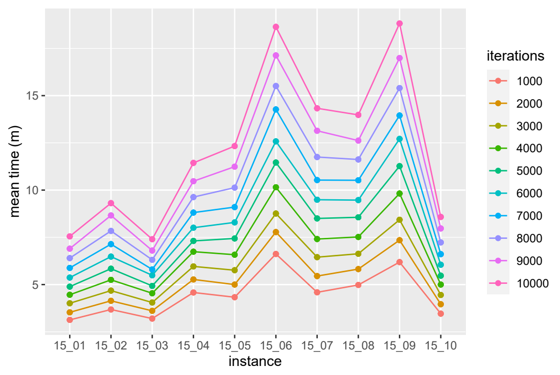
Table 7.27 presents the indicator values for all the instances. In terms of CV, AUGMECON2 tends to find better values but, according to EPS, GD and IGD, the smaller values are usually found by the BIALNS. The CV values indicate that the proportion of dominated points of the algorithm is larger than the one of the AUGMECON2 method, which is consistent with the number of total and non dominated points (TP and NDP). Meanwhile, indicators EPS, GD and IGD analyze the distances between the solutions (found by the two methods) and the reference set (all non dominated points found by the two methods). Therefore, the values obtained indicate that the distances are larger for the AUGMECON2 solutions. Thus, it can be concluded that, although the BIALNS finds more dominated points than the AUGMECON2 method, the solutions given by the algorithm are closer to the reference set, which reinforces the good performance of the BIALNS.

	CV		EPS		GD		IGD		NDP		TP	
Instance	AUGM	ALG	AUGM	ALG	AUGM	ALG	AUGM	ALG	AUGM	ALG	AUGM	ALG
	1.01e-02	4.37e-01	**8.94e-04**	6.26e-03	**2.87e-06**	1.53e-03	**9.27e-06**	1.36e-04	**685**	378	**692**	671
	1.01e-02	1.27e-01	**1.79e-03**	3.58e-03	**5.92e-07**	7.35e-06	**3.12e-05**	1.23e-04	**685**	572	**692**	655
15_01	**1.01e-02**	1.74e-01	**8.94e-04**	**8.94e-04**	**5.92e-07**	9.21e-06	**2.35e-05**	1.01e-04	**685**	527	**692**	638
	1.01e-02	3.50e-02	**1.79e-03**	5.45e-03	**5.92e-07**	4.26e-05	**4.44e-05**	9.80e-05	**685**	661	**692**	685
	1.01e-02	3.06e-02	**1.79e-03**	5.99e-03	**5.92e-07**	1.60e-05	**3.90e-05**	9.33e-05	**685**	665	**692**	686
	1.68e-02	**2.11e-03**	2.77e-02	**2.20e-03**	1.64e-03	**8.18e-08**	1.42e-03	**1.76e-04**	409	**473**	416	**474**
	1.68e-02	**4.15e-03**	2.77e-02	**1.29e-04**	1.64e-03	**3.48e-07**	1.51e-03	**1.21e-04**	409	**480**	416	**482**
15_02	1.68e-02	**6.24e-03**	2.77e-02	**4.78e-04**	1.64e-03	**2.07e-06**	1.49e-03	**1.03e-04**	409	**478**	416	**481**
	1.68e-02	**8.39e-03**	2.77e-02	**2.61e-03**	1.64e-03	**2.10e-05**	1.56e-03	**2.34e-04**	409	**473**	416	**477**
	1.68e-02	**2.09e-03**	2.77e-02	**3.62e-04**	1.64e-03	**8.11e-08**	1.46e-03	**1.58e-04**	409	**477**	416	**478**
	0	**0**	**3.75e-05**	1.50e-04	**0**	**0**	9.11e-05	**6.64e-05**	165	**166**	165	**166**
	0	**0**	**3.75e-05**	3.00e-04	**0**	**0**	**9.11e-05**	3.04e-04	**165**	164	**165**	164
15_03	**0**	**0**	**3.75e-05**	1.50e-04	**0**	**0**	**9.11e-05**	1.58e-04	**165**	164	**165**	164
	0	**0**	3.75e-05	**0**	**0**	**0**	9.11e-05	**0**	165	**168**	165	**168**
	0	**0**	**3.75e-05**	1.50e-04	**0**	**0**	**9.11e-05**	6.64e-05	165	**166**	165	**166**
	5.88e-03	**5.81e-03**	2.22e-03	**1.36e-03**	3.26e-04	**4.63e-06**	4.27e-04	**2.37e-05**	845	**856**	850	**861**
	5.88e-03	8.17e-03	**2.22e-03**	9.99e-03	3.26e-04	**3.66e-05**	2.34e-04	**9.02e-05**	845	**850**	850	**857**
15_04	**4.71e-03**	1.17e-02	2.22e-03	**1.74e-03**	3.23e-04	**1.95e-05**	4.21e-04	**6.58e-05**	**846**	**846**	850	**856**
	5.88e-03	6.04e-03	2.22e-03	**1.42e-04**	3.26e-04	**9.18e-08**	4.34e-04	**7.40e-05**	**845**	823	**850**	828
	4.71e-03	6.00e-02	2.22e-03	**1.82e-03**	3.25e-04	**1.24e-05**	2.24e-04	**7.15e-05**	**846**	783	**850**	833
	4.52e-02	2.55e-01	4.43e-02	**5.11e-03**	2.96e-03	**8.06e-04**	3.10e-03	**6.55e-04**	**190**	184	199	**247**
	3.52e-02	9.72e-02	4.43e-02	**3.94e-03**	2.88e-03	**1.35e-04**	2.82e-03	**1.31e-04**	192	**223**	199	**247**
15_05	**4.52e-02**	1.33e-01	4.43e-02	**1.95e-03**	2.92e-03	**1.09e-04**	3.06e-03	**9.46e-05**	190	**215**	199	**248**
	4.52e-02	1.96e-01	4.01e-02	**3.60e-03**	3.03e-03	**6.65e-04**	2.45e-03	**3.35e-04**	190	**197**	199	**245**
	2.51e-02	1.41e-01	4.43e-02	**2.22e-03**	2.86e-03	**2.43e-04**	2.80e-03	**1.86e-04**	194	**213**	199	**248**
	7.09e-03	1.19e-02	6.05e-02	**4.26e-04**	3.11e-03	**3.29e-05**	8.64e-03	**1.27e-06**	280	**333**	282	**337**
	7.09e-03	**0**	7.03e-02	**0**	6.17e-04	**0**	8.89e-03	**0**	280	**337**	282	**337**
15_06	**7.09e-03**	1.18e-02	7.03e-02	**1.04e-02**	1.87e-03	**7.66e-05**	1.28e-02	**3.09e-05**	280	**336**	282	**340**
	7.09e-03	**0**	7.03e-02	**0**	1.87e-03	**0**	1.37e-02	**0**	280	**341**	282	**341**
	7.09e-03	**0**	7.03e-02	**1.17e-05**	1.87e-03	**0.00e+00**	9.70e-03	**2.34e-05**	280	**332**	282	**332**
	7.41e-02	2.32e-01	1.29e-01	**2.06e-03**	2.13e-03	**3.83e-04**	1.75e-02	**4.02e-04**	150	**208**	162	**271**
	6.79e-02	2.16e-01	1.29e-01	**2.50e-03**	2.24e-03	**4.99e-04**	1.80e-02	**5.18e-04**	151	**214**	162	**273**
15_07	**7.41e-02**	2.19e-01	1.29e-01	**2.62e-03**	2.04e-03	**5.18e-04**	1.69e-02	**4.94e-04**	150	**214**	162	**274**
	6.17e-02	2.80e-01	9.66e-02	**1.02e-03**	1.19e-03	**7.37e-04**	3.59e-02	**6.83e-04**	152	**195**	162	**271**
	4.94e-02	3.53e-01	1.07e-01	**4.34e-03**	2.11e-03	**1.01e-03**	1.27e-02	**9.19e-04**	154	**172**	162	**266**
	4.26e-02	1.52e-01	6.05e-02	**1.89e-03**	1.05e-03	**1.84e-04**	3.49e-02	**7.36e-04**	180	**206**	188	**243**
	2.13e-02	4.90e-01	3.78e-02	**1.32e-02**	**7.21e-05**	8.95e-04	2.04e-02	**2.10e-03**	**184**	126	188	**247**
15_08	**3.19e-02**	1.91e-01	5.86e-02	**1.66e-03**	1.59e-03	**1.40e-04**	2.51e-02	**9.85e-04**	**182**	186	188	**230**
	4.26e-02	1.74e-01	4.91e-02	**1.89e-03**	**1.19e-04**	1.38e-04	2.27e-02	**2.03e-03**	180	**199**	188	**241**
	4.26e-02	2.25e-01	6.05e-02	**1.92e-03**	1.05e-03	**2.36e-04**	2.89e-02	**1.07e-03**	**180**	176	188	**227**
	3.08e-02	4.93e-01	1.31e-01	**1.74e-03**	4.85e-03	**8.72e-04**	5.89e-02	**1.76e-03**	**126**	110	130	**217**
	3.08e-02	5.71e-01	1.31e-01	**3.06e-03**	4.83e-03	**2.26e-03**	6.52e-02	**2.02e-03**	**126**	94	130	**219**
15_09	**7.69e-03**	6.65e-01	1.23e-01	**7.37e-03**	2.83e-05	**1.42e-03**	4.31e-02	**4.99e-03**	**129**	72	130	**215**
	2.31e-02	6.61e-01	1.31e-01	**5.85e-03**	4.82e-03	**3.40e-03**	8.20e-02	**2.91e-03**	**127**	84	130	**248**
	7.69e-03	5.34e-01	1.31e-01	**3.35e-03**	4.80e-03	**1.41e-03**	4.36e-02	**1.85e-03**	**129**	97	130	**208**
	2.23e-02	3.85e-01	6.70e-03	**1.92e-04**	1.45e-03	**3.14e-05**	2.08e-04	**7.38e-05**	**395**	256	404	**416**
	2.72e-02	2.01e-01	1.17e-02	**1.68e-03**	1.53e-03	**2.10e-05**	2.96e-04	**9.06e-05**	**393**	330	404	**413**
15_10	**2.97e-02**	2.40e-01	1.17e-02	**1.54e-04**	1.53e-03	**1.89e-05**	3.04e-04	**5.36e-05**	**392**	319	404	**420**
	2.97e-02	1.60e-01	1.17e-02	**1.68e-03**	1.53e-03	**9.68e-06**	2.87e-04	**4.47e-05**	**392**	352	404	**419**
	1.73e-02	3.07e-01	6.70e-03	**1.68e-03**	1.45e-03	**3.84e-05**	2.10e-04	**7.69e-05**	**397**	289	404	**417**

Table 7.27: Indicator values for 8000 iterations (15 services).



(a) AUGMECON2 time (hours).



(b) BIALNS time (minutes).

Figure 7.50: Computational times (15 services).

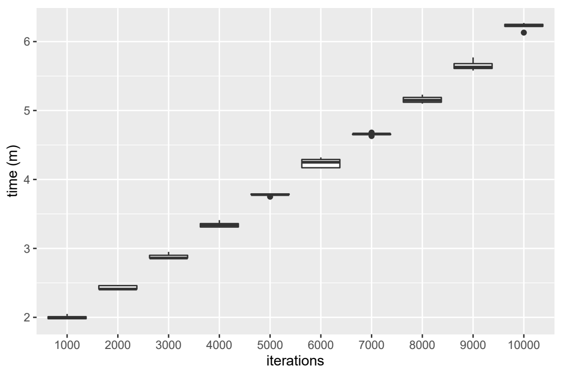
In terms of computational time, Figure 7.50 and Table 7.28 show the behavior of the instances for both resolution methods. Notice that, in both methods, the behavior of the instances in terms of computational time is similar but in different magnitudes: while the computational times in the

AUGMECON2 vary from 13.05 to 236.69 hours, the highest computational time is lower than 20 minutes.

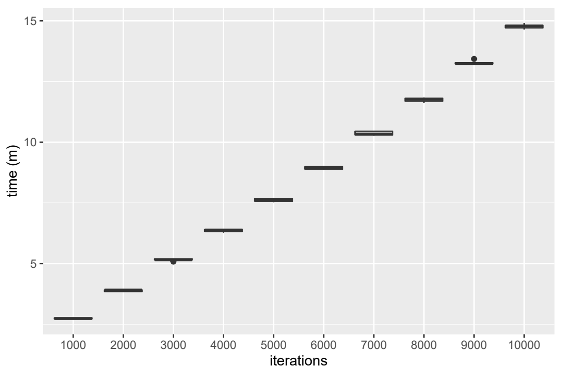
	AUGMECON2	BIALNS mean time (minutes)									
Instance	(hours)	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
10_01	77.68	3.13	3.53	4.01	4.46	4.89	5.37	5.88	6.40	6.90	7.55
10_02	36.67	3.68	4.14	4.68	5.25	5.84	6.48	7.14	7.84	8.66	9.31
10_03	13.05	3.20	3.61	4.05	4.54	4.93	5.49	5.80	6.31	6.80	7.40
10_04	58.97	4.58	5.27	5.96	6.74	7.31	8.01	8.81	9.63	10.47	11.44
10_05	86.51	4.33	5.00	5.76	6.58	7.44	8.29	9.10	10.13	11.24	12.33
10_06	236.69	6.62	7.78	8.76	10.15	11.46	12.58	14.27	15.51	17.13	18.64
10_07	143.44	4.59	5.45	6.45	7.41	8.50	9.49	10.53	11.75	13.14	14.33
10_08	124.00	4.98	5.82	6.63	7.52	8.56	9.47	10.52	11.62	12.62	13.98
10_09	164.72	6.19	7.35	8.43	9.82	11.27	12.71	13.95	15.40	12.62	18.82
10_10	90.29	3.46	3.96	4.45	5.00	5.47	6.05	6.61	7.23	7.97	8.58

Table 7.28: Computational time for instances with 15 services.

Figure 7.51 presents the boxplot of the computational time, for two instances (for the other ones the behavior is similar), necessary to carry out steps 2 and 3 in the BIALNS method. As it happened with 10 services, the BIALNS is very consistent with respect to the computational time in the 5 runs.



(a) Instance 15_03.



(b) Instance 15_06.

Figure 7.51: Computational time for steps 2 and 3 (15 services).

Figure 7.52 presents boxplots, for the best and worst solutions, of the number of non dominated and total points found by both resolutions methods. According to the total points, both solutions of the BIALNS have a similar behavior, finding more points than the AUGMECON2 method. In terms of the number of non dominated points, it can be seen that the worst solution has significantly fewer points than the AUGMECON2 one. Meanwhile, the best solutions improve the ones of the AUGMECON2 method.

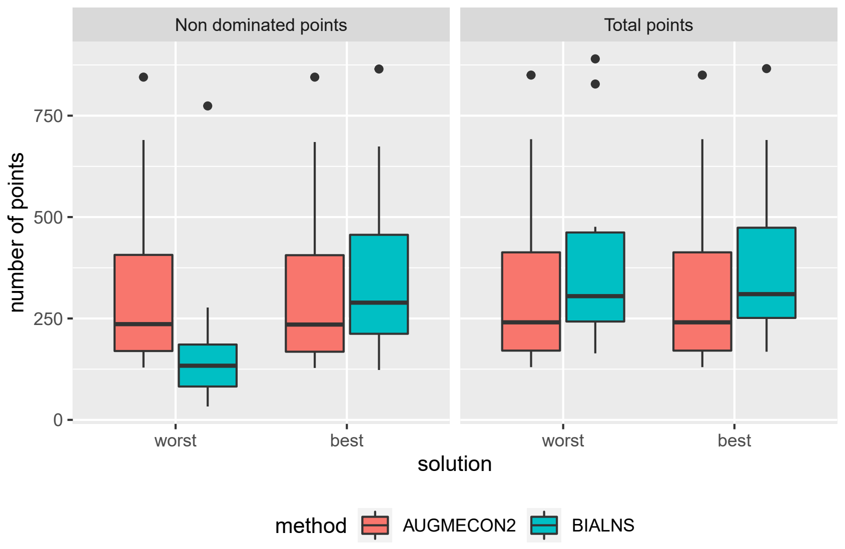


Figure 7.52: Number of non dominated and number of total points (15 services).

The evolution of the non dominated points, according to the number of iterations, is presented in Figure 7.53. The best solution is practically not affected by the parameter, usually finding more non dominated points than the AUGMECON2 method. Meanwhile, when considering a small number of iterations, the number of points is notably smaller for the worst solution. However, from 5000 iterations onwards the worst solutions are similar to the AUGMECON2 one.

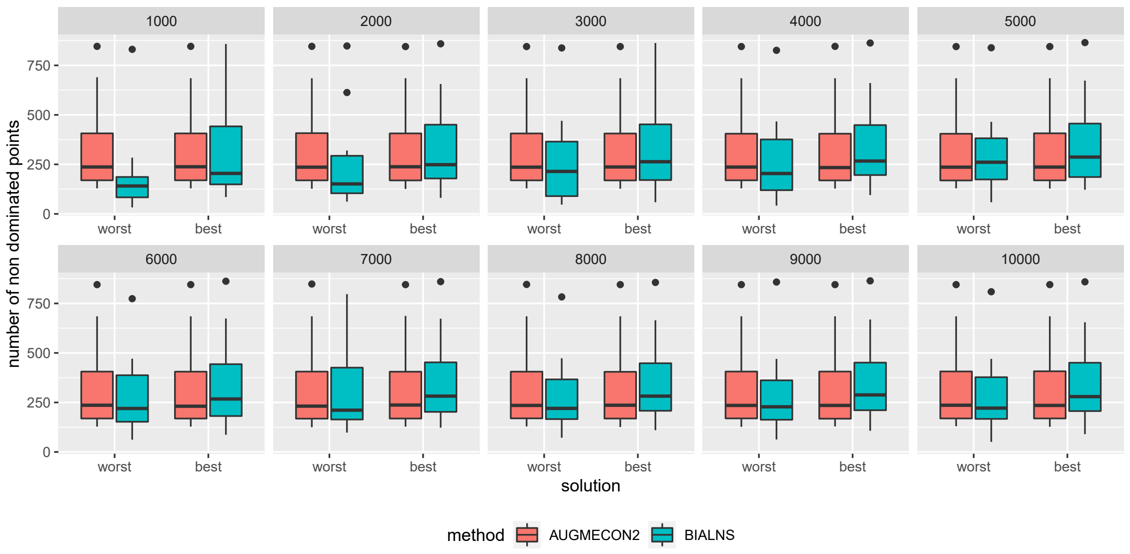
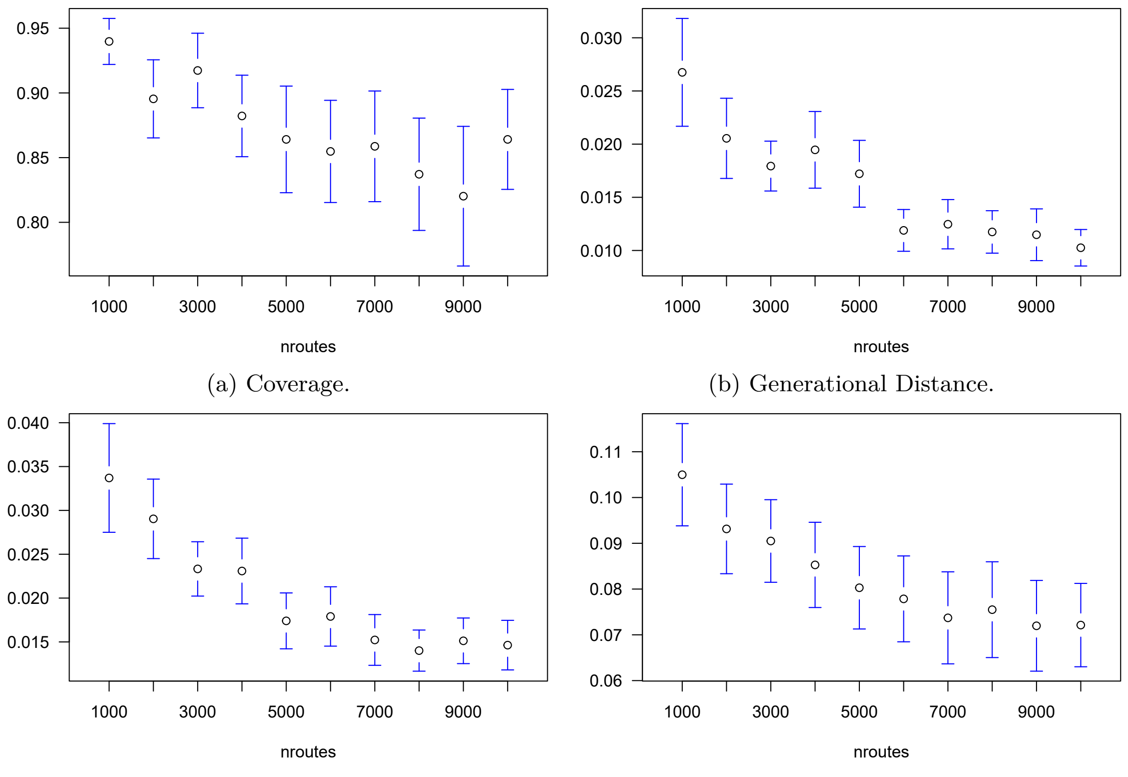


Figure 7.53: Number of non dominated points by iterations (15 services).

Results of instances with 25 services

For the instances with 25 services, increasing the number of iterations results in better solutions. Figure 7.49 shows that the values of the indicators decrease as the number of iterations increases. Thus, setting the parameter to 9000 seems to be the better configuration. Notice that, due to high computational times, the AUGMECON2 method was not used to solve these instances.



(a) Coverage.

(b) Generational Distance.

(c) Inverted Generational Distance.

(d) Epsilon.

Figure 7.54: Confidence intervals of the indicators in terms of nroutes (25 services).

The mean computational times necessary to solve the instances are presented in Figure 7.55. The slowest ones are 25_06, 25_09 and 25_10 which, for 10000 iterations, take between 30 and 45 minutes to be solved.

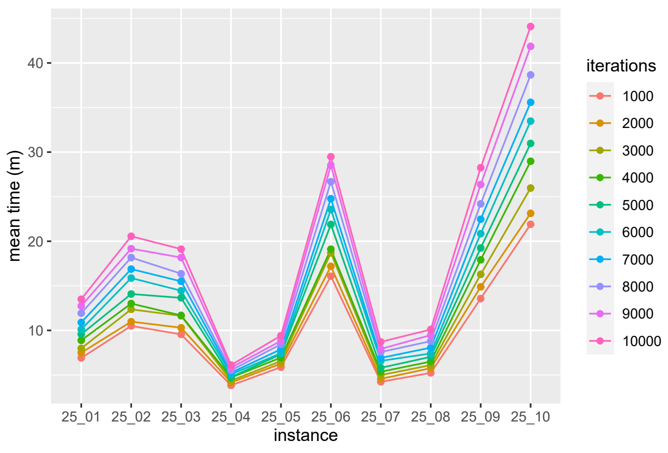
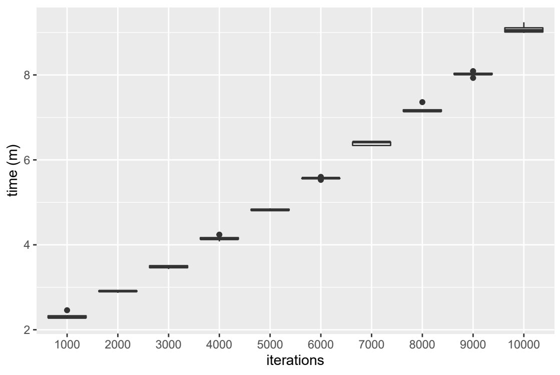
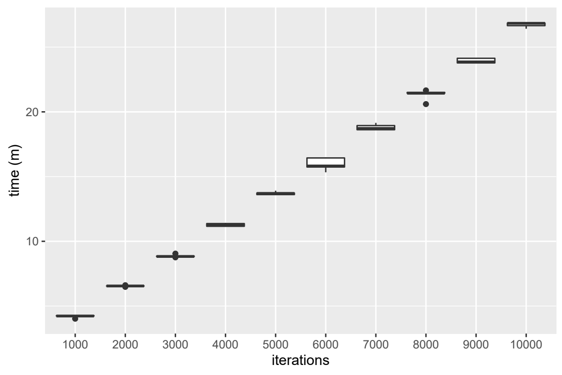


Figure 7.55: Computational times (25 services).

According to Figure 7.56, for the two presented instances, the time taken by the BIALNS to generate different solutions and to obtain non dominated ones is very consistent for all the configurations. The behavior of the remaining instances is similar to those shown in the figure, therefore they are not included.



(a) Computational time instance 25_01.



(b) Computational time instance 25_10.

Figure 7.56: Computational time for steps 2 and 3 (25 services).

7.5.3.4 Pareto frontier

With respect to the appearance of the Pareto frontier, three different scenarios arise when comparing the BIALNS and the AUGMECON2. In order to facilitate the interpretation of the results, three colors are used in the figures: red for the points only found by the BIALNS, green for the points only found by the AUGMECON2 method and blue for the points found by both resolution methods[13].

Figure 7.57 shows the non dominated points for instance 15_03. In this case, the solution found by the BIALNS is almost equal to the one given by the AUGMECON2 method (blue points). The methods only differ in 5 points: 3 found by the BIALNS (red points) and 2 found by the AUGMECON2 method (green points).

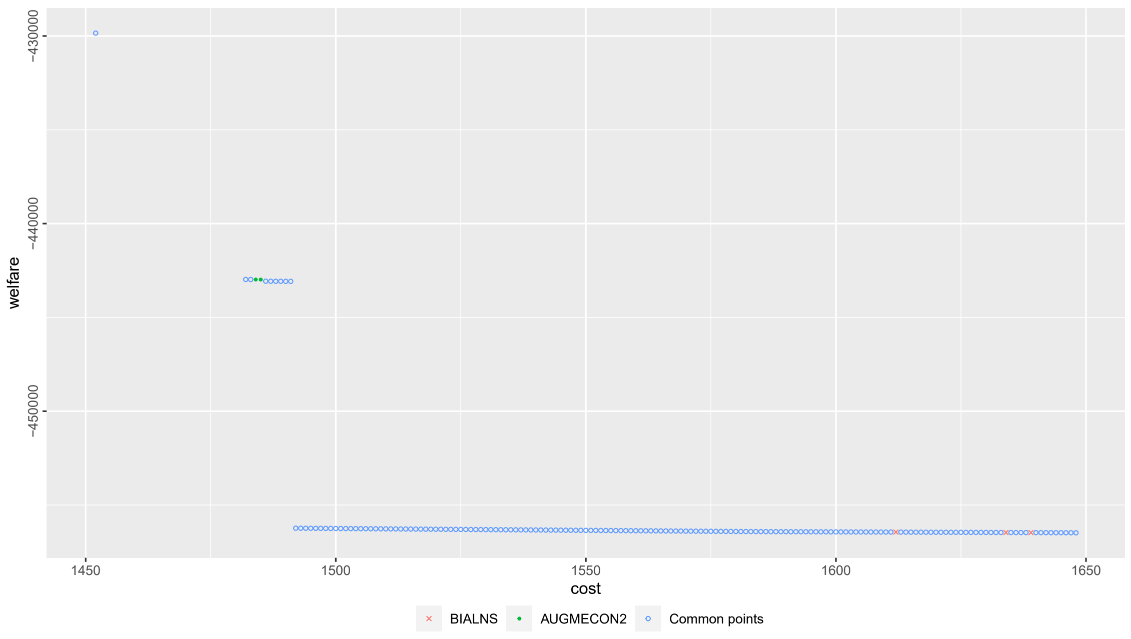


Figure 7.57: Pareto frontier for instance 15_03.

In Figure 7.58 the solutions for instance 15_06 are presented. In this case, the BIALNS finds

[13]Notice that, as it was explained in Chapter 2, in order to maximize the affinity in a minimization problem the weight assigned to it has to be negative. Thus, the values of the welfare in the Pareto frontiers are negative.

the same points than the AUGMECON2 method (blue points) and even more (red points). This is because, due to the time limit of 1 hour for each grid point, the AUGMECON2 is not able to find feasible solutions to cover the left side of the front. To be precise, the only point found by AUGMECON2 in the left side of the frontier (green point) is dominated by the BIALNS.

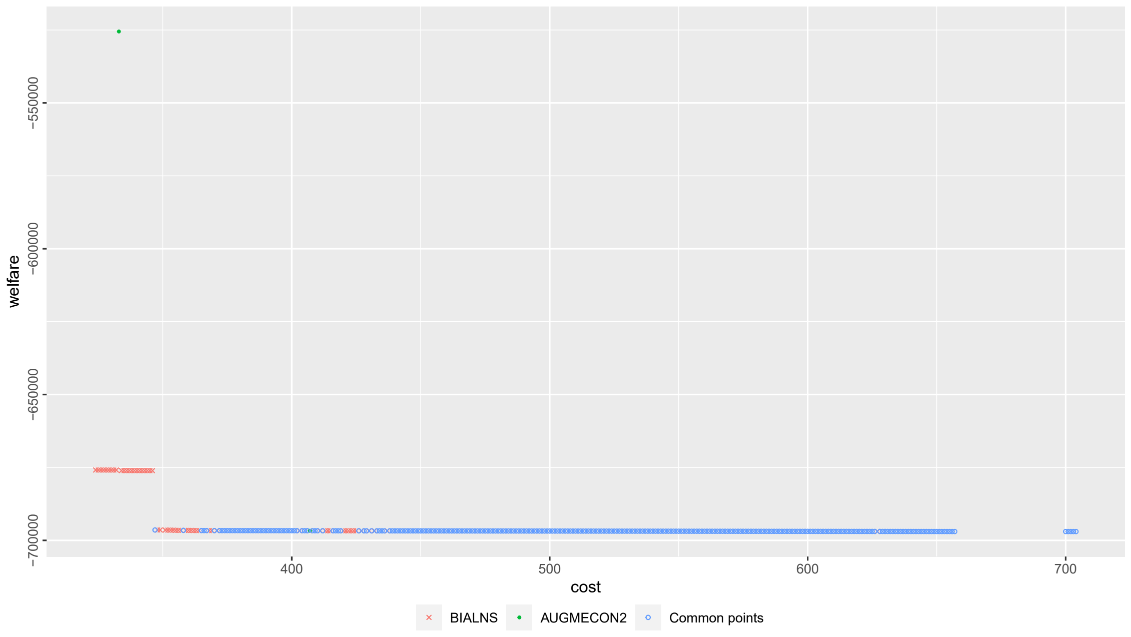


Figure 7.58: Pareto frontier for instance 15_06.

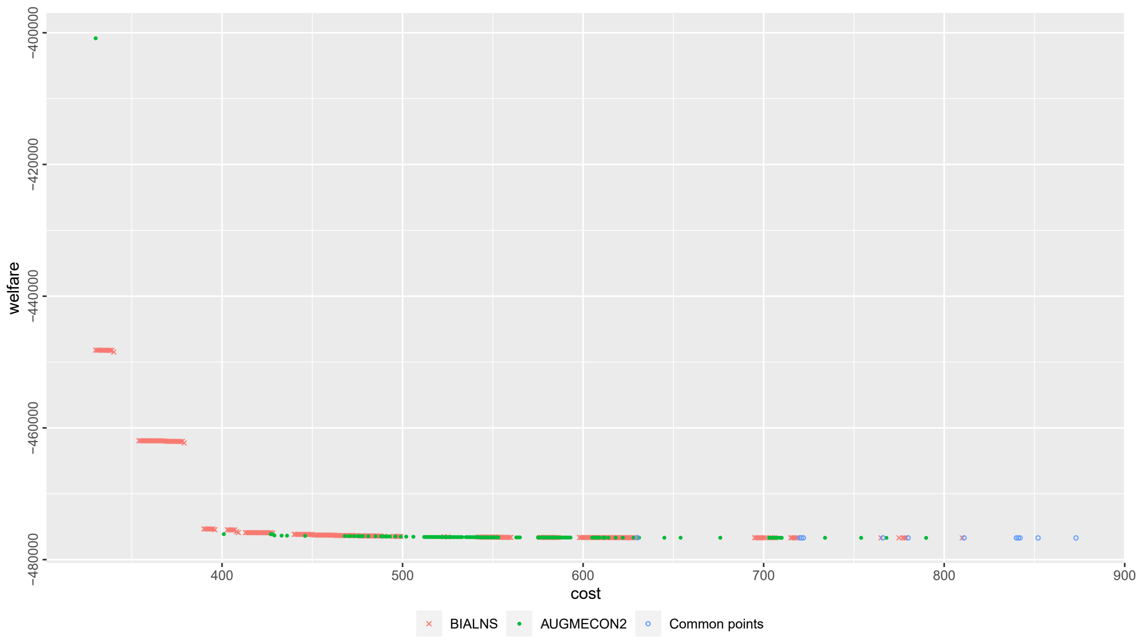


Figure 7.59: Pareto frontier for instance 15_09.

The solutions for instance 15_09 are presented in Figure 7.59. It can be seen, by the reduced number of points in common (blue points), that the solutions given by the two methods are quite different. In this case, some of the points found by the AUGMECON2 method are slightly better than the ones provided by the BIALNS (some green points dominate the red ones), although they are really close. It is important to remark that, although these points are dominated, the values of GD, IGD and EPS (the measures of the distance between solutions) in Figure 7.49 are close to 0. As before, only the BIALNS (red points) is able to cover the left side of the frontier, because of the time limit in the AUGMECON2 method.

7.5.4 Real instances

Finally, to conclude this chapter the BIALNS is used to solve the real instances. The parameters considered are:

Step 1: Initialize the sets. Proportion of solution to destroy is *auto*_1%. Time limit: 90 minutes.

Step 2: Generate different solutions. Number of iterations (nroutes): 10000. Number of iterations (nalns): 1. Proportion of solution to destroy (pr): 1%.

Step 3: Get non dominated solutions. Number of iterations (nsols): $3e^5$.

Figure 7.60 shows the Pareto frontier obtained for week 9. To analyze the solutions found, six points along the frontier have been selected (denoted as A - F). Point A (F) is the solution of the hierarchical problem that prioritizes the welfare over the cost (the cost over the welfare). Points B, C, D and E are intermediate solutions.

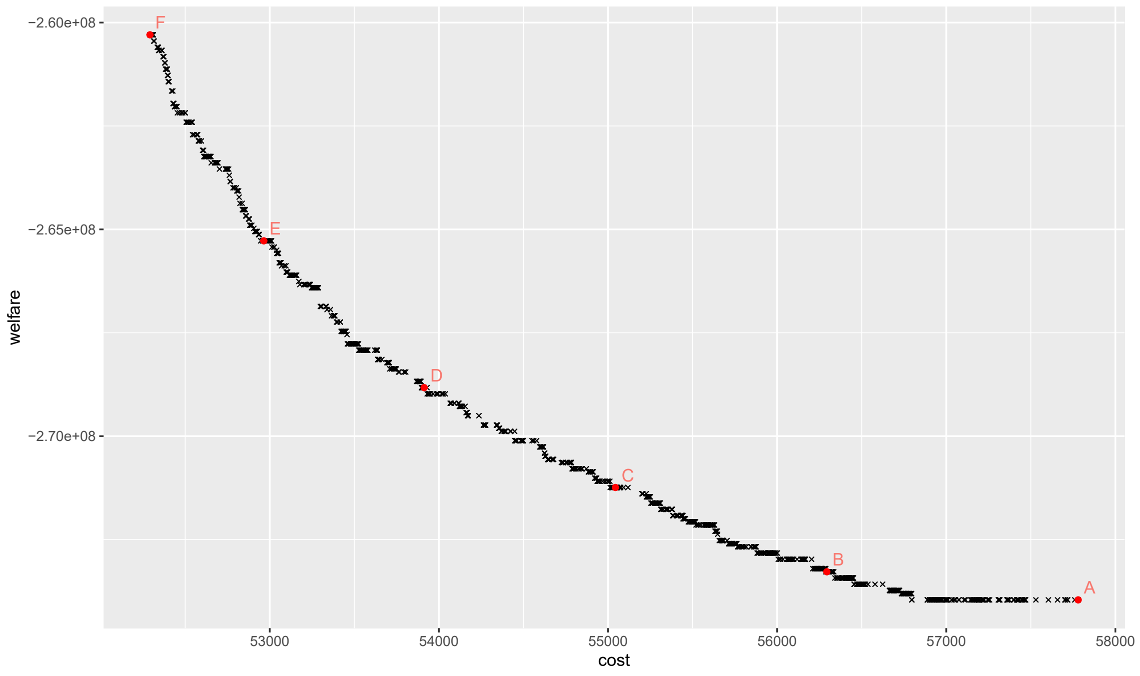


Figure 7.60: Pareto frontier for week 9.

Because all the points presented in the frontier are non dominated, in order to improve the welfare it is necessary to increase the cost. This can be seen in more detail in Table 7.29, which presents the objective values of the selected points, as well as the ones of the solution considered by the company. Regarding the cost, all the solutions considered are better than the one of the company, which can also be assessed analyzing the overtime and working time of the solutions.

In terms of the welfare, the solution of the company is between points B and C, although the penalization value of C is much better than the one of the company.

Solution	Cost	Overtime	Worked time	Welfare ($\times e^8$)	Affinity	Penalization
Company	69744	11525	58219	-2.71	3576	10518
A	57780	4692	53088	-2.74	3631	3342
B	56294	3761	52533	-2.73	3622	4005
C	55044	2733	52311	-2.71	3595	4436
D	53913	1670	52243	-2.68	3563	5218
E	52965	722	52243	-2.65	3516	7900
F	52292	273	52019	-2.60	3450	9961

Table 7.29: Objective values of the solutions.

To complete the comparison of the solutions Figures 7.61 and 7.62 are presented. Figure 7.61 represents the mean overtime, idle time, break time and travel time per caregiver according to the considered solutions. Figure 7.62 displays the mean soft time window penalization per user. According to these figures it is clear that the six solutions taken from the Pareto frontier are better than the one of the company. Specifically, one can observe that solution A (which prioritizes the welfare over the cost) is the one with highest overtime and idle time but, as more priority is given to the cost, the idle time disappears and the overtime decreases. In a similar way, Figure 7.62 shows that as the priority of the welfare decreases, the soft time window penalization increases.

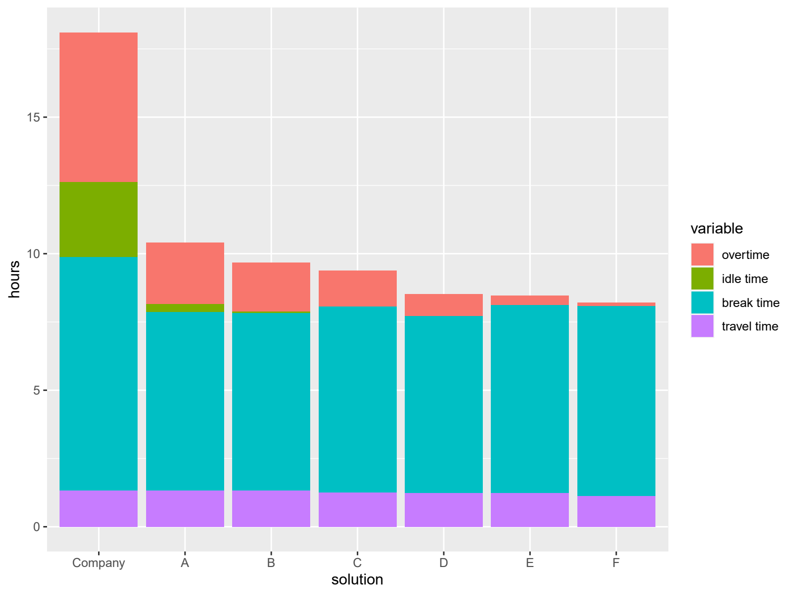


Figure 7.61: Mean weekly time per caregiver (week 9).

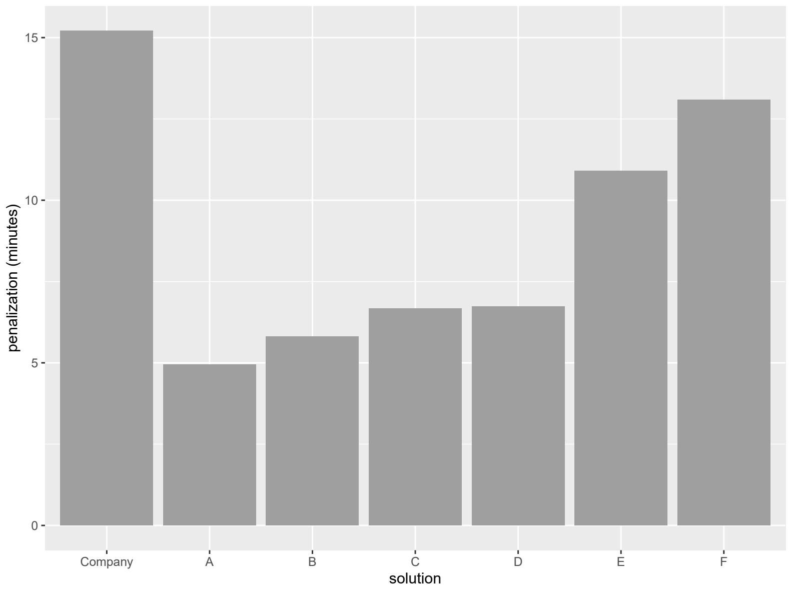


Figure 7.62: Mean soft time window penalization per user (week 9).

Conclusions

In this work, a real problem of a home care company is addressed. The problem shares many of the characteristics of other routing and scheduling problems in home care. But, despite the extensive literature devoted to the understanding of this type of problems, commonly known by the acronym HCSP, there is a special feature that substantially distinguishes it from the others and makes its study of great interest. In accordance with this characteristic, the longest break between two daily consecutive services of each caregiver will not be included as part of her working day, provided that it is greater than a fixed number of hours.

In order to gather all the requirements of the company, a MILP with two clearly differentiated objectives was formulated. The first objective is related to the welfare of the users, since improving the living conditions of the users is a crucial issue for the company. The second objective is to reduce the cost of the salaries of the caregivers as much as possible. Three different approaches to solve the problem were proposed: two hierarchical ones (prioritizing the welfare over the cost, or the cost over the welfare) and a biobjective one (considering the welfare and the cost at the same time).

In a first attempt, the hierarchical problems were tackled with an optimization solver. However, due to the difficulty of the model, it only provided feasible solutions for small instances. With the purpose of solving the problem in a more realistic setting, a metaheuristic based on the ALNS algorithm was developed, incorporating new methods to improve the quality of the solutions. Specifically, these methods arise from the difficulty of, once the services and their order have been assigned to the caregivers, determining the starting time of each service. When prioritizing the welfare, the method combines the margin of service movements within the hard and soft time windows with the grouping of services into blocks, thus allowing to manage the impact of the breaks between services. When prioritizing the cost, the method works with the hard time windows to reduce the impact of breaks between services in the solution, either by making them as small as possible or by making one of them as large as possible. This is not an easy task, since many different scenarios might occur. Thus, it could be seen as the most innovative part of the framework. To evaluate the performance of the algorithms, instances of 10, 15, 25, 50 and 100 services (which were obtained adapting the Solomon instances, for routing and scheduling problems with time windows constraints, to the characteristics of the HCSP under study) were tested. Finally, real data instances of 15 consecutive weeks from 2016 to 2017 were considered to compare the schedules provided by the company with the algorithm solutions. The results were very promising, since multiple configurations of the algorithm can be used to obtain good solutions for the Solomon instances, depending on their size. In terms of the real instances, the schedules obtained using the algorithm improved those provided by the company.

In terms of the biobjective version of the problem, the AUGMECON2 method was used to obtain the exact Pareto front, but it was only able to solve small instances, usually needing large computational times. Therefore, a metaheuristic algorithm, that uses the ALNS technique and the two scheduling methods designed for the hierarchical problems, was developed. This metaheuristic algorithm consists in generating different solutions and then modifying its schedules to obtain non dominated solutions. To validate the algorithm, several computational experiments using the Solomon instances were run, considering different configurations for the algorithm, in order to select the ones that provide the best solutions. The quality of the solutions was analyzed by comparing them to the ones obtained with the AUGMECON2 method, using a set of performance indicators, as well as the computational times needed to solve the instances. Apart from that, the different scenarios that can arise when evaluating the Pareto fronts obtained by both resolution methods were reviewed. Finally, an example illustrating the Pareto front obtained for a real instance was presented, comparing it with the company solution.

Just to conclude, some future research lines are summarized, remarking that there is a wide range of possibilities for further studying this home care scheduling problem:

- Study the same problem but considering the minimization of the number of caregivers necessary to carry out all services. This version of the problem would be useful for the company when they want to start working in a new area, being necessary to hire caregivers to attend the new users.

- Implement other metaheuristic techniques to solve the problem, instead of the ALNS. For example: Greedy Randomized Adaptive Search Procedure (GRASP), Iterated Local Search (ILS) or Tabu Search (TS). These methods could perform well when solving the HCSP problem, in which case they would be useful to study the behavior of the ALNS in more detail.

- Consider other metaheuristic techniques to solve the biobjective problem, specifically the Non-dominated Sorting Genetic Algorithm (NSGA-II), which is a method widely used in the literature when solving multiobjective problems. Thus, there would be another method to compare the BIALNS with, specially for the instances that the AUGMECON2 method was not able to solve.

- Tackle the problem using decomposition techniques, considering three different steps: clustering, assignation and routing. The clustering step would be in charge of dividing the services into groups, according to the distances between them. The assignation step would be used to allocate the caregivers to the clusters of services. Finally, the routing step would obtain the schedules for each route of the caregivers.

Resumen en castellano

Este trabajo surge del proyecto Innterconecta GIRO - Generación, Gestión e Integración de Rutas en OLAP (ITC-20151247), en el que participaron un grupo de empresas de diferentes sectores, pero con retos similares en cuanto a la optimización de sus procesos organizativos. La actividad de estas empresas requería la movilidad de sus empleados, es decir, sus trabajadores debían desplazarse para completar tareas, o prestar servicios, en diferentes ubicaciones. Otra característica común a todas estas empresas era la necesidad de ajustar de forma continua sus horarios para solventar una serie de contingencias tales como: altas o bajas de clientes, ausencia de un trabajador, incompatibilidad de tareas y horarios, etc. De modo que los problemas logísticos de las empresas estaban relacionados con la planificación de tareas, la elaboración de horarios y la obtención de rutas. Por lo tanto, nuestra tarea en el proyecto consistió en explorar la integración de técnicas de investigación operativa y optimización matemática en los procesos de planificación de rutas y horarios de las empresas participantes. El objetivo del proyecto era diseñar herramientas automáticas para ayudar a las empresas con la toma de decisiones. Así, una vez conocida la situación de los trabajadores, las herramientas serían capaces de proporcionar un plan de trabajo para cada uno de ellos y adaptarlo en función de las contingencias que pudieran surgir.

En el proyecto participaron seis empresas:

Gesuga (gesuga.com) Esta empresa presta un servicio de recogida de subproductos cárnicos no destinados al consumo humano. Dispone de una flota de vehículos, ubicados en varias plantas, que siguen la programación diaria establecida por el departamento de rutas de la empresa. Estos horarios deben cubrir los pedidos de recogida del día cumpliendo una serie de restricciones. La empresa pretende mejorar la planificación de estas rutas, minimizando las distancias recorridas por los vehículos y su consumo de combustible. Además, también es necesario hacer frente a contingencias imprevistas, como la llegada de nuevos pedidos de recogida. Es decir, las rutas iniciales deben adaptarse a las necesidades que vayan surgiendo a lo largo del día.

Biogas Fuel Cell Esta empresa está especializada en la gestión y transformación de residuos orgánicos en biogás, para lo cual dispone de su propia planta de biogás con una zona de recepción de residuos. Los residuos son transportados tanto por contratistas externos como por los propios vehículos de la empresa, por lo que es necesario coordinar las llegadas de ambos tipos de vehículos. De ello se derivan dos problemas: organizar la recepción de los residuos en la planta y establecer las rutas para los vehículos propios. Además, se debe de tener en cuenta que estas rutas pretenden minimizar la distancia recorrida y deben cumplir los horarios marcados por el primer problema.

Mayores (mayores.es) Esta empresa presta servicios de atención a domicilio a personas mayores/dependientes y a sus familias. Para ello, el personal especializado visita a los usuarios en sus propios domicilios y realiza las tareas que les han sido asignadas. Por tanto, la empresa necesita determinar las rutas que deben seguir sus auxiliares, con el objetivo de mantener los horarios de los pacientes y reducir el tiempo de desplazamiento. Este problema presenta una característica importante, que consiste en que las rutas deben favorecer la asignación de las mismas auxiliares a los mismos usuarios, puesto que al estar trabajando con personas dependientes su bienestar y satisfacción es lo más importante. La herramienta propuesta debe ser capaz de obtener una planificación de horarios y rutas que se adapte dinámicamente a todo tipo de contingencias: solicitudes de cambios en los horarios, servicios cancelados, modificación de la duración de los servicios, etc.

Grupo On (seguridadon.es) Esta empresa se especializa en sistemas de seguridad que implican la instalación de dispositivos de alarma, tanto para empresas como para viviendas particulares. Un equipo de técnicos está constantemente desplazándose para la instalación de alarmas y su mantenimiento, lo cual incluye inspecciones periódicas y trabajos de reparación. La empresa se enfrenta a problemas de planificación debido a la necesidad de organizar las rutas de sus técnicos para cumplir las citas con cada cliente, con el objetivo de reducir los tiempos de desplazamiento y maximizar la prioridad de los clientes, satisfaciendo al mismo tiempo su disponibilidad.

Mugatra (mugatra.es) Esta empresa ofrece a sus clientes un conjunto de servicios que proporcionan una cobertura total en materia de seguridad y prevención de riesgos laborales. El servicio más importante es la vigilancia de la salud, cuyo objetivo es realizar evaluaciones periódicas, con el fin de valorar si el estado de salud del personal de la empresa es satisfactorio para el desempeño de sus funciones. El problema que presenta esta empresa es una combinación de tres problemas de planificación diferentes:

- Establecer las fechas de servicio para cada cliente.
- Decidir si utilizar unidades móviles o alquilar salas, lo cual implica programar el envío de equipos médicos.
- Programar los exámenes para todos los empleados de la empresa cliente.

El objetivo es minimizar los tiempos de desplazamiento entre clientes y maximizar el número de clientes que pueden ser atendidos en el mismo lugar.

Taprega (taprega.com) La empresa ofrece a sus clientes servicios en el ámbito de la seguridad laboral, especializándose en la prevención de riesgos laborales. En la empresa hay dos tipos de trabajadores: técnicos y comerciales. Los técnicos se encargan de realizar las inspecciones a los clientes. Los comerciales visitan a los clientes, o potenciales clientes, con el objetivo de hacer publicidad de la empresa y formalizar nuevos contratos. De este modo, la empresa se enfrenta a dos problemas de planificación: por un lado, la planificación de la agenda de sus técnicos y, por otro, la planificación de sus comerciales. En el problema de planificación de las rutas de los técnicos, los objetivos son minimizar los tiempos de desplazamiento y maximizar la urgencia de las visitas realizadas. En el problema de los representantes de ventas, el objetivo es minimizar los tiempos de viaje. Además, en ambos casos es necesario tener en cuenta una periodicidad en las visitas.

Esta tesis se centra en el problema planteado por Mayores, la empresa de asistencia a domicilio. Los servicios de atención a domicilio tienen como objetivo ayudar a personas mayores o dependientes a mantener o mejorar su calidad de vida sin necesidad de abandonar sus hogares, ya que estas personas suelen sentirse más cómodas si pueden continuar viviendo en sus casas, en lugar de tener que trasladarse a un centro especializado. La empresa Mayores lleva desde 1997 prestando servicios de atención a domicilio en A Coruña y alrededores.

Los principales objetivos de los servicios de atención a domicilio son:

- Mejorar la calidad de vida de sus usuarios.
- Favorecer la adquisición de habilidades que permitan un desarrollo más independiente en la vida diaria.
- Ayudar a los familiares del usuario en sus responsabilidades de cuidado.
- Evitar, o retrasar, el traslado del usuario a una residencia.

Los usuarios son personas que necesitan cuidados o ayuda para realizar determinadas tareas y acuden a la empresa para mejorar su calidad de vida. Los usuarios demandan un conjunto de servicios que deberán ser atendidos por las empleadas[14] de la empresa en sus domicilios. Durante estos servicios las empleadas deben realizar determinadas tareas, que pueden ser domésticas, personales o sanitarias. Algunas de estas actividades deben realizarse en un horario determinado, por ejemplo, levantarse y acostarse deben realizarse por la mañana y por la noche, respectivamente. Otras actividades pueden realizarse en cualquier momento del día, como limpiar el polvo o hacer la colada. En concreto, se debe disponer de la siguiente información para cada uno de los servicios a realizar: su duración, la franja horario en la que se debe llevar a cabo y el día de la semana en que debe realizarse.

A las empleadas de la empresa se le denominan auxiliares, que son las encargadas de visitar a los usuarios en sus domicilios para realizar las tareas asignadas. Las auxiliares solo pueden trabajar por contrato un número máximo de horas a la semana, considerando como tiempo de trabajo:

- El tiempo que emplean en llevar a cabo los servicios en los domicilios de los usuarios.
- El tiempo de desplazamiento entre servicios.
- Todos los descansos entre servicios (que se denominará tiempo libre) que puedan tener durante la jornada diaria, a excepción del de mayor duración si es de dos o más horas.

Los tiempos de desplazamiento pueden variar en función de la región de actividad. En las zonas urbanas, los usuarios están más concentrados, por lo que las auxiliares suelen poder desplazarse a pie o en transporte público. Lo contrario ocurre en las zonas rurales, donde los usuarios suelen estar muy repartidos y las auxiliares generalmente tienen que utilizar su propio vehículo. Cabe destacar que la empresa sólo cobra por las horas que las auxiliares están atendiendo a los usuarios. Por este motivo, los desplazamientos y los tiempos libres de las auxiliares pueden interpretarse como pérdidas económicas para la empresa.

La empresa tiene contratadas también a un conjunto de supervisoras, a las que se les ha asignado un número de usuarios y auxiliares. La asignación de usuarios se hace teniendo en cuenta su ubicación y no puede modificarse, ya que las supervisoras son las encargadas de estar en constante contacto con los familiares de los usuarios (redactan informes y monitorizan la evolución de los usuarios).

La empresa considera que la consistencia en la asignación de usuarios a auxiliares es muy importante, hasta el punto de que es preferible cambiar el horario de un servicio para que la auxiliar que lo realice sea siempre la misma. A veces incluso es necesario que un usuario sea atendido por más de una auxiliar. De este modo, las asignaciones anteriores y las preferencias entre usuarios y auxiliares, o problemas que hayan podido surgir entre ellos, se utilizan para obtener una lista de niveles de afinidad. La afinidad establece un nivel de compatibilidad entre usuarios y auxiliares, quedando estos niveles definidos de la siguiente manera:

Nivel 0. El usuario no debe ser atendido por esta auxiliar bajo ningún concepto. Esta situación puede darse porque la auxiliar no esté capacitada para atender al usuario o porque haya habido algún tipo de incidente entre ellos.

[14]Nótese que se habla de auxiliares en femenino puesto que las empleadas de la empresa on mujeres.

Nivel 1. La auxiliar podría atender al usuario sólo si no hay otra opción. Esto puede ocurrir cuando la auxiliar ha recibido alguna queja de poca importancia por parte del usuario.

Nivel 2. La auxiliar podría atender al usuario aunque todavía no se ha establecido el grado de compatibilidad entre ellos. En este nivel, el usuario no requiere ningún tipo de característica especial por parte de las auxiliares.

Nivel 3. La auxiliar aún no ha atendido al usuario pero, por las características de la misma, el supervisor piensa que podría ser una muy buena candidata para llevar a cabo sus servicios.

Nivel 4. El usuario fue atendido con éxito por la auxiliar en el pasado, ya fuera de forma esporádica o continuada.

Nivel 5. La auxiliar se encuentra atendiendo al usuario de forma continuada y de manera satisfactoria.

En la empresa, las supervisoras se encargan de organizar manualmente los planes de trabajo de todas las auxiliares que tienen asignadas. Es decir, asignan cada servicio requerido por un usuario, durante la semana, a una auxiliar y establecen las horas concretas en las que comenzarán dichos servicios.

La tesis se organiza como sigue:

Capítulo 1:

El Capítulo 1 se centra en el problema original propuesto por la empresa. El objetivo principal era actualizar los horarios semanales de las auxiliares, que se repiten en el tiempo hasta que surge la necesidad de modificarlos de nuevo. Los horarios deben ser actualizados para resolver un conjunto de posibles incidencias, que representan cambios permanentes en las planificaciones tales como: altas/bajas de usuarios, aumento/disminución del número de servicios requeridos y modificación de parámetros de los mismo (duración, franja horaria,...). Una característica especial de este problema es que en Mayores no están excesivamente interesados en modificar los horarios originales de las auxiliares, ya que para ellos lo realmente importante es el bienestar de los usuarios. Por ello, las incidencias deben resolverse modificando lo menos posible los horarios previos.

Este problema se abordó mediante el algoritmo Simulated Annealing (SA), un método metaheurístico de optimización inspirado en el proceso de templado de metales. Este método está integrado dentro de un algoritmo heurístico desarrollado a medida para resolver los incidentes a los que se enfrenta la empresa. El heurístico puede dividirse en tres etapas: fase inicial (para extraer la información principal), programación de servicios (para asignar los servicios a las auxiliares) y optimización (para mejorar los horarios). En la fase de optimización se utiliza el SA, que parte de una solución inicial y luego se mueve aleatoriamente por el espacio de soluciones para encontrar mejores horarios. En este capítulo se ofrece una descripción detallada del algoritmo heurístico, la metaheurística SA y los movimientos diseñados para explorar el espacio de soluciones.

Este algoritmo forma parte del núcleo de en una herramienta de software, la cual se desarrolló para proporcionar a la empresa un sistema de apoyo a la toma de decisiones que les ayude a obtener los horarios semanales de las auxiliares. En el capítulo puede verse una descripción del planificador de horarios de la herramienta así como un ejemplo de cómo el algoritmo resolvería un conjunto de incidencias (dar de baja a un usuario, aumentar la duración de los servicios de otro usuario y dar de alta a dos nuevos usuarios).

El contenido de este capítulo ha sido publicado en Méndez-Fernández *et al.* (2020).

Capítulo 2:
El Capítulo 2 presenta la formulación matemática (un modelo de programación lineal entera mixta (MILP)) de una versión más general del problema. La nueva versión del problema pretende obtener las soluciones desde cero, con el fin de conseguir los mejores horarios posibles. Es decir, en lugar de partir de un horario inicial que no debe modificarse excesivamente, existe total libertad para diseñar un horario óptimo para las auxiliares. Una característica muy importante del problema es que si el descanso diario más grande de una auxiliar tiene una duración de 2 horas o más, éste no se considera como tiempo de trabajo y no se pagará, lo que añade una dificultad considerable al problema.

Las principales variables de decisión son las que determinan las rutas que seguirán las auxiliares y aquellas que establecen las horas en las que deben realizarse los servicios. Se consideran dos funciones objetivo diferentes: el bienestar de los usuarios, que evalúa el grado de satisfacción de los usuarios con las planificaciones, y el coste de la planificación, que cuantifica la carga de trabajo de las auxiliares. Por un lado, el bienestar está compuesto por la afinidad entre los usuarios y las auxiliares que los atienden, y la penalización por realizar servicios fuera de su ventana de tiempo preferida. Por otro lado, el coste de los horarios está compuesto por las horas extras de las auxiliares y su tiempo total de trabajo. El modelo incluye restricciones relacionadas con los requisitos de la empresa (ventanas de tiempo de servicios y auxiliares o tiempo máximo de trabajo diario), restricciones de ruta (inicio y fin en los servicios ficticios, que la hora de inicio entre servicios consecutivos debe permitir que la auxiliar se desplace entre ellos ellos o que los servicios sólo pueden tener un servicio anterior) y restricciones relacionadas con los objetivos (penalización causada por no respetar la ventana preferida de los servicios, duración del descanso no remunerado o bien considerar las horas extra de las auxiliares).

Además, se muestra un ejemplo que detalla el funcionamiento del problema, mostrando cómo los valores de la función objetivo pueden cambiar según la planificación. El capítulo concluye con la explicación de los tres posibles enfoques propuestos para abordar el problema: priorizar el bienestar sobre el coste, priorizar el coste sobre el bienestar y una alternativa biobjetivo.

La formulación matemática del HCSP descrita en este capítulo se incluye en Méndez-Fernández *et al.* (2023a) (aceptado para publicación) y en el artículo en desarrollo Méndez-Fernández *et al.* (2023c). Este modelo también describe la versión biobjetivo del problema, por lo que también aparecerá en Méndez-Fernández *et al.* (2023b).

Capítulo 3:
El Capítulo 3 presenta una descripción exhaustiva de la metaheurística utilizada para resolver el problema, el algoritmo Adaptive Large Neighborhood Search (ALNS), que se basa en el principio de destrucción y reparación.

En primer lugar, se presenta el esquema general del ALNS. El algoritmo consiste en eliminar servicios de los horarios de las auxiliares, de acuerdo con un operador de eliminación; posteriormente, se utiliza un operador de inserción para volver a colocar los servicios en las rutas. La planificación reparada se utiliza entonces para actualizar la solución actual, teniendo en cuenta un criterio de aceptación. Los operadores de eliminación e inserción se eligen al azar en función de sus probabilidades, que aumentan cuando los operadores resultan en una mejora de la solución. Este proceso de destrucción y reparación se repite hasta que se cumple un cierto criterio de parada. Tras describir el funcionamiento del ALNS, se describen los operadores de eliminación e inserción, proporcionando su pseudocódigo y ejemplos para facilitar su comprensión.

Es importante destacar que los operadores modifican las rutas pero, para obtener soluciones completas y poder evaluar la función objetivo, es necesario establecer los horarios de los servicios. Es decir, al reparar la solución con los operadores de inserción, se debe fijar la hora a la que comenzará cada servicio de las rutas. Para ello, dada una ruta, se proponen tres métodos diferentes:

- Resolver el problema de planificación de horarios mediante programación por restricciones.
- Dos aproximaciones heurísticas: una para obtener horarios en el caso de que se priorice el bienestar de los usuarios sobre el coste, y otra que obtenga los horarios pero priorizando el coste sobre el bienestar.

Para finalizar el capítulo, se presenta el enfoque de programación por restricciones utilizado para obtener los horarios de los servicios, describiendo la función objetivo a optimizar y las restricciones que debe cumplir el horario.

El algoritmo ALNS presentado en este capítulo se incluye en Méndez-Fernández *et al.* (2023a) y también aparecerá en el artículo en desarrollo Méndez-Fernández *et al.* (2023c). El método metaheurístico también se utilizará en Méndez-Fernández *et al.* (2023b) para resolver la versión biobjetivo del HCSP.

Capítulo 4:
En el Capítulo 4 se describe el algoritmo heurístico diseñado para obtener el horario de una ruta con el fin de priorizar el bienestar de los usuarios sobre el coste de la planificación. El método solamente trabaja con una ruta, por lo que el bienestar se corresponde con la penalización de las ventanas de tiempo preferidas (ya que la afinidad está fijada) y el coste es el tiempo de trabajo de la auxiliar (las horas extras sólo se pueden calcular de forma semanal).

En primer lugar, se presenta una descripción básica del algoritmo. El método puede dividirse en tres fases. En la primera se obtienen las horas de inicio más tempranas y más tardías de los servicios, así como los bloques de servicios que tienen ventanas de tiempo preferidas conflictivas (i.e. aquellas ventanas que se intersecan). El segundo paso consiste en obtener el horario de la ruta que tiene la menor penalización por ventana de tiempo preferida. Por último, en el tercer paso se actualiza el horario para reducir su coste sin empeorar el bienestar. El funcionamiento de cada uno de los tres pasos se ilustra con un ejemplo basado en una ruta con 6 servicios. Finalmente, se introducen algunas funciones auxiliares en el Apéndice 4.A para facilitar la comprensión del algoritmo.

El algoritmo de planificación heurística propuesto en este capítulo aparece en Méndez-Fernández *et al.* (2023a). También será útil para el HCSP biobjetivo incluido en el trabajo en desarrollo Méndez-Fernández *et al.* (2023b).

Capítulo 5:
En el Capítulo 5 se presenta el algoritmo heurístico diseñado para obtener el horario de una ruta con el fin de priorizar el coste del horario sobre el bienestar de los usuarios. Al igual que en el capítulo anterior, el método sólo trabaja con una ruta, por lo que el bienestar coincide con la penalización por ventanas de tiempo preferidas y el coste es el tiempo de trabajo de la auxiliar para ese horario.

El capítulo se centra en los principales elementos del algoritmo, mientras que las funciones complementarias se pueden encontrar en el Apéndice 5.A. El algoritmo comienza calculando las horas de inicio más tempranas y más tardías de los servicios. Seguidamente, se obtienen los horarios que presentan un coste mínimo. Tras ello, estos horarios se actualizan para mejorar la penalización de ventana de tiempo preferida sin empeorar su coste, eligiendo el horario con mejor penalización. A lo largo de todo el capítulo se utiliza un ejemplo para ilustrar el funcionamiento del algoritmo.

El algoritmo heurístico de planificación presentado en este capítulo dará lugar a Méndez-Fernández *et al.* (2023c), que está actualmente en desarrollo. También aparecerá en Méndez-Fernández *et al.* (2023b).

Capítulo 6:

El Capítulo 6 está relacionado con la versión biobjetivo del HCSP, es decir, el problema que considera ambos objetivos simultáneamente. La solución de un problema de este tipo se denomina frontera de Pareto y es un conjunto compuesto por soluciones no dominadas (i.e. soluciones que no pueden mejorarse sin empeorar, al menos, uno de los objetivos). Dicha frontera de Pareto puede obtenerse de forma exacta empleando, por ejemplo, el método clásico conocido como Epsilon Constraint.

La primera parte del capítulo se centra en el método AUGMECON2, una versión mejorada del método Epsilon Constraint que se utilizará para abordar el HCSP biobjetivo. Dicho método consiste en, dado un conjunto de puntos que dividen el rango de uno de los objetivos, resolver iterativamente una versión del HCSP que fija dicho objetivo al valor dado por los puntos considerados.

La segunda del capítulo parte describe un algoritmo metaheurístico desarrollado a medida para obtener aproximaciones de la frontera de Pareto. El algoritmo se basa en las metodologías presentadas en los capítulos 3, 4 y 5. Por lo tanto, el enfoque principal del capítulo son las nuevas funcionalidades desarrolladas para resolver el problema biobjetivo, utilizando un ejemplo para ilustrarlas. El algoritmo biobjetivo puede dividirse en tres pasos. En el primero se inicializa el conjunto de soluciones no dominadas. En el segundo paso se obtienen soluciones compuestas por diferentes rutas. Por último, en el tercer paso se obtienen las soluciones no dominadas. Para concluir el capítulo, en el Apéndice 6.A se muestra información adicional que completa la descripción del algoritmo.

Los contenidos descritos en este capítulo se resumen en el artículo en desarrollo relacionado con el HCSP biobjetivo, Méndez-Fernández *et al.* (2023b).

Capítulo 7:

El Capítulo 7 presenta el estudio computacional llevado a cabo para evaluar los métodos de resolución descritos en los capítulos anteriores. Para ello, en los experimentos numéricos se utilizan dos tipos de conjuntos de datos diferentes: las instancias de Solomon y los datos reales proporcionados por la empresa.

El primer paso para evaluar los enfoques jerárquicos es utilizar un solver del estado del arte (Gurobi) para resolver directamente la formulación MILP del problema, utilizando para ello las instancias de Solomon. El objetivo de estas pruebas es tener las soluciones exactas y ver el tamaño de instancia para el cual el método exacto ya no es capaz de encontrar soluciones (o no garantiza optimalidad global). Tras ello, se analizan los resultados obtenidos con diferentes configuraciones del ALNS (combinadas con los algoritmos heurísticos de planificación de horarios descritos en los capítulos 4 y 5), con el fin de evaluar el algoritmo metaheurístico y obtener la mejor configuración de parámetros.

Posteriormente, se utiliza el ALNS para resolver instancias reales y se comparan las soluciones obtenidas con los horarios utilizados por la empresa. En el caso del enfoque jerárquico que prioriza el bienestar de los usuarios sobre el coste de la programación, también se evalúa el rendimiento del algoritmo ALNS combinado con la programación con restricciones.

Los resultados relativos al enfoque jerárquico que prioriza el bienestar sobre el coste se incluyen en Méndez-Fernández *et al.* (2023a) y los del enfoque jerárquico que prioriza el coste sobre el bienestar se recogen en el artículo Méndez-Fernández *et al.* (2023c).

En cuanto al problema biobjetivo, este se estudia en primer lugar resolviendo las instancias de Solomon con el método AUGMECON2, para analizar los tiempos computacionales que necesita este método exacto. A continuación, se evalúa el comportamiento del algoritmo metaheurístico biobjetivo estudiando los diferentes parámetros que intervienen en su configuración. Para ello, se comparan las soluciones del algoritmo con las dadas por el método AUGMECON2. Finalmente, se presenta la frontera de Pareto de ciertas instancias de Solomon y para una instancia real.

Los resultados de la versión biobjetivo del HCSP se resumirán en el documento de trabajo Méndez-Fernández *et al.* (2023b).

Bibliography

Ait Haddadene, S. R., Labadie, N., & Prodhon, C. 2019. Bicriteria Vehicle Routing Problem with Preferences and Timing Constraints in Home Health Care Services. *Algorithms*, **12**(8).

Akjiratikarl, C., Yenradee, P., & Drake, P. R. 2007. PSO-based algorithm for home care worker scheduling in the UK. *Computers & Industrial Engineering*, **53**(4), 559–583.

Alves, F., Costa, L., Rocha, A. M. A. C., Pereira, A. I., & Leitao, P. 2019. A multi-objective approach to the optimization of home care visits scheduling.

Bachouch, R. B., Guinet, A., & Hajri-Gabouj, S. 2011. A Decision-Making Tool for Home Health Care Nurses' Planning. *Supply Chain Forum: An International Journal*, **12**(1), 14–20.

Bard, J. F., Shao, Y., & Jarrah, A. I. 2014. A sequential GRASP for the therapist routing and scheduling problem. *Journal of Scheduling*, **17**(2), 109–133.

Belhor, M., El-Amraoui, A., Jemai, A., & Delmotte, F. 2023. Multi-objective evolutionary approach based on K-means clustering for home health care routing and scheduling problem. *Expert Systems with Applications*, **213**, 119035.

Bertels, S., & Fahle, T. 2006. A hybrid setup for a hybrid scenario: Combining heuristics for the home health care problem. *Computers & Operations Research*, **33**(10), 2866–2890. Part Special Issue: Constraint Programming.

Bourdais, S., Galinier, P., & Pesant, G. 2003. hibiscus: A Constraint Programming Application to Staff Scheduling in Health Care. *Principles and Practice of Constraint Programming – CP 2003*, 153–167.

Braekers, K., Hartl, R. F., Parragh, S. N., & Tricoire, F. 2016. A bi-objective home care scheduling problem: Analyzing the trade-off between costs and client inconvenience. *European Journal of Operational Research*, **248**(2), 428–443.

Cappanera, P., & Scutellà, M. G. 2015. Joint Assignment, Scheduling, and Routing Models to Home Care Optimization: A Pattern-Based Approach. *Transportation Science*, **49**(4), 830–852.

Cappanera, P., Scutellà, M. G., Nervi, F., & Galli, L. 2018. Demand uncertainty in robust Home Care optimization. *Omega*, **80**, 95–110.

Carello, G., & Lanzarone, E. 2014. A cardinality-constrained robust model for the assignment problem in Home Care services. *European Journal of Operational Research*, **236**(2), 748–762.

Carello, G., Lanzarone, E., & Mattia, S. 2018. Trade-off between stakeholders' goals in the home care nurse-to-patient assignment problem. *Operations Research for Health Care*, **16**, 29–40.

Chaieb, M., & Ben Sassi, D. 2021. Measuring and evaluating the Home Health Care Scheduling Problem with Simultaneous Pick-up and Delivery with Time Window using a Tabu Search metaheuristic solution. *Applied Soft Computing*, **113**, 107957.

Chaieb, M., Jemai, J., & Mellouli, K. 2020. A decomposition - construction approach for solving the home health care scheduling problem. *Health Care Management Science*, **23**(2), 264–286.

Chan, W. T., & Hu, H. 2002. Constraint Programming Approach to Precast Production Scheduling. *Journal of Construction Engineering and Management*, **128**(6), 513–521.

Cissé, M., Yalçındağ, S., Kergosien, Y., Şahin, E., Lenté, C., & Matta, A. 2017. OR problems related to Home Health Care: A review of relevant routing and scheduling problems. *Operations Research for Health Care*, **13-14**, 1–22.

Deb, K., Chaudhuri, S., & Miettinen, K. 2006. Towards Estimating Nadir Objective Vector Using Evolutionary Approaches. *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, 643–650.

Decerle, J., Grunder, O., Hassani, A. H. E., & Barakat, O. 2018a. Impact of the workload definition on the multi-objective home health care problem. *IFAC-PapersOnLine*, **51**(11), 346–351.

Decerle, J., Grunder, O., El Hassani, A. H., & Barakat, O. 2021. A matheuristic-based approach for the multi-depot home health care assignment, routing and scheduling problem. *RAIRO-Oper. Res.*, **55**, S1013–S1036.

Decerle, J., Grunder, O., Hajjam El Hassani, A., & Barakat, O. 2018b. A memetic algorithm for a home health care routing and scheduling problem. *Operations Research for Health Care*, **16**, 59–71.

Decerle, J., Grunder, O., Hajjam El Hassani, A., & Barakat, O. 2019a. A hybrid memetic-ant colony optimization algorithm for the home health care problem with time window, synchronization and working time balancing. *Swarm and Evolutionary Computation*, **46**, 171–183.

Decerle, J., Grunder, O., Hajjam El Hassani, A., & Barakat, O. 2019b. A memetic algorithm for multi-objective optimization of the home health care problem. *Swarm and Evolutionary Computation*, **44**, 712–727.

Dhingra, S. L., Wang, L., Ma, W., Wang, L., Ren, Y., & Yu, C. 2021. Enabling In-Depot Automated Routing and Recharging Scheduling for Automated Electric Bus Transit Systems. *Journal of Advanced Transportation*, **2021**, 1–15.

Di Mascolo, M., Martinez, C., & Espinouse, M. L. 2021. Routing and scheduling in Home Health Care: A literature survey and bibliometric analysis. *Computers & Industrial Engineering*, **158**, 107255.

Dimitsas, A., Gogos, C., Valouxis, C., Tzallas, A., & Alefragis, P. 2022. A Pragmatic Approach for Solving the Sports Scheduling Problem. *In Proc. 13th Int. Conf. Pract. Theory Autom. Timetabling*, **3**, 195–207.

Durillo, J. J., & Nebro, A. J. 2011. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, **42**(10), 760–771.

Efthymiou, N., & Yorke-Smith, N. 2023. Predicting the Optimal Period for Cyclic Hoist Scheduling Problems. *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 238–253.

Erdem, M., & Bulkan, S. 2017. A two-stage solution approach for the large-scale home healthcare routing and scheduling problem. *South African Journal of Industrial Engineering*, **28**(12), 133 – 149.

Euchi, J., Zidi, S., & Laouamer, L. 2020. A Hybrid Approach to Solve the Vehicle Routing Problem with Time Windows and Synchronized Visits In-Home Health Care. *Arabian Journal for Science and Engineering*, **45**(12), 10637–10652.

Euchi, J., Masmoudi, M., & Siarry, P. 2022. Home health care routing and scheduling problems: A literature review. *4OR*, **20**(3), 351–389.

Fathollahi-Fard, A. M., Hajiaghaei-Keshteli, M., & Tavakkoli-Moghaddam, R. 2018. A bi-objective green home health care routing problem. *Journal of Cleaner Production*, **200**, 423–443.

Fathollahi-Fard, A. M., Govindan, K., Hajiaghaei-Keshteli, M., & Ahmadi, A. 2019. A green home health care supply chain: New modified simulated annealing algorithms. *Journal of Cleaner Production*, **240**, 118200.

Fathollahi-Fard, A. M., Ahmadi, A., Goodarzian, F., & Cheikhrouhou, N. 2020. A bi-objective home healthcare routing and scheduling problem considering patients' satisfaction in a fuzzy environment. *Applied Soft Computing*, **93**, 106385.

Fazel Zarandi, M. H., Sadat Asl, A. A., Sotudian, S., & Castillo, O. 2020. A state of the art review of intelligent scheduling. *Artificial Intelligence Review*, **53**(1), 501–593.

Fikar, C., & Hirsch, P. 2017. Home health care routing and scheduling: A review. *Computers & Operations Research*, **77**, 86–95.

Focacci, F., Lodi, A., & Milano, M. 2002. Mathematical Programming Techniques in Constraint Programming: A Short Overview. *Journal of Heuristics*, **8**(1), 7–17.

Frifita, S., & Masmoudi, M. 2020. VNS methods for home care routing and scheduling problem with temporal dependencies, and multiple structures and specialties. *International Transactions in Operational Research*, **27**(1), 291–313.

Garaix, T., Gondran, M., Lacomme, P., Mura, E., & Tchernev, N. 2018. Workforce Scheduling Linear Programming Formulation. *IFAC-PapersOnLine*, **51**(06), 264–269.

Grenouilleau, F., Legrain, A., Lahrichi, N., & Rousseau, L.-M. 2019. A set partitioning heuristic for the home health care routing and scheduling problem. *European Journal of Operational Research*, **275**(1), 295–303.

Grenouilleau, F., Lahrichi, N., & Rousseau, L.-M. 2020. New decomposition methods for home care scheduling with predefined visits. *Computers & Operations Research*, **115**, 104855.

Gurobi Optimization, LLC. 2021. *Gurobi Optimizer Reference Manual.*

Habibnejad-Ledari, H., Rabbani, M., & Ghorbani-Kutenaie, N. 2019. Solving a multi-objective model toward home care staff planning considering cross-training and staff preferences by NSGA-II and NRGA. *Scientia Iranica*, **26**(5), 2919–2935.

Haddadene, S. A., Labadie, N., & Prodhon, C. 2016. NSGAII enhanced with a local search for the vehicle routing problem with time windows and synchronization constraints. *IFAC-PapersOnLine*, **49**(12), 1198–1203.

Haimes, Y., Lasdon, L., & Wismer, D. 1971. On a bicriterion formation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, 296–297.

Ham, A. M. 2018. Integrated scheduling of m-truck, m-drone, and m-depot constrained by time-window, drop-pickup, and m-visit using constraint programming. *Transportation Research Part C: Emerging Technologies*, **91**, 1–14.

Kandakoglu, A., Sauré, A., Michalowski, W., Aquino, M., Graham, J., & McCormick, B. 2020. A decision support system for home dialysis visit scheduling and nurse routing. *Decision Support Systems*, **130**, 113224.

Kergosien, Y., Lenté, C., & Billaut, J. C. 2009. Home health care problem: An extended multiple Traveling Salesman Problem. *4th Multidisciplinary International Conference on Scheduling: Theory and Applications*, 85–92.

Khodabandeh, P., Kayvanfar, V., Rafiee, M., & Werner, F. 2021. A Bi-Objective Home Health Care Routing and Scheduling Model with Considering Nurse Downgrading Costs. *International Journal of Environmental Research and Public Health*, **18**(3).

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. 1983. Optimization by Simulated Annealing. *Science*, **220**(4598), 671–680.

Lahrichi, N., Lanzarone, E., & Yalçındağ, S. 2022. A First Route Second Assign decomposition to enforce continuity of care in home health care. *Expert Systems with Applications*, **193**, 116442.

Lenstra, J. K., & Kan, A. H. G. R. 1981. Complexity of vehicle routing and scheduling problems. *Networks*, **11**(2), 221–227.

Li, Y., Ye, C., Wang, H., Wang, F., & Xu, X. 2022. A discrete multi-objective grey wolf optimizer for the home health care routing and scheduling problem with priorities and uncertainty. *Computers & Industrial Engineering*, **169**, 108256.

Lin, C. C., Hung, L. P., Liu, W. Y., & Tsai, M. C. 2018. Jointly rostering, routing, and rerostering for home health care services: A harmony search approach with genetic, saturation, inheritance, and immigrant schemes. *Computers & Industrial Engineering*, **115**, 151–166.

Liu, M., Yang, D., Su, Q., & Xu, L. 2018. Bi-objective approaches for home healthcare medical team planning and scheduling problem. *Computational and Applied Mathematics*, **37**(4), 4443–4474.

Liu, R., Yuan, B., & Jiang, Z. 2017. Mathematical model and exact algorithm for the home care worker scheduling and routing problem with lunch break requirements. *International Journal of Production Research*, **55**(2), 558–575.

Liu, W., Dridi, M., Fei, H., & El Hassani, A. H. 2021a. Hybrid metaheuristics for solving a home health care routing and scheduling problem with time windows, synchronized visits and lunch breaks. *Expert Systems with Applications*, **183**, 115307.

Liu, W., Dridi, M., Fei, H., & El Hassani, A. H. 2021b. Solving a multi-period home health care routing and scheduling problem using an efficient matheuristic. *Computers & Industrial Engineering*, **162**, 107721.

Luna, F., Cervantes, A., Isasi, P., & Valenzuela-Valdés, J. F. 2018. Grid-enabled evolution strategies for large-scale home care crew scheduling. *Cluster Computing*, **21**(2), 1261–1273.

Ma, X., Fu, Y., Gao, K., Sadollah, A., & Wang, K. 2022. Integration routing and scheduling for multiple home health care centers using a multi-objective cooperation evolutionary algorithm with stochastic simulation. *Swarm and Evolutionary Computation*, **75**, 101175.

Malagodi, L., Lanzarone, E., & Matta, A. 2021. Home care vehicle routing problem with chargeable overtime and strict and soft preference matching. *Health Care Management Science*, **24**(1), 140–159.

Manavizadeh, N., Farrokhi-Asl, H., & Beiraghdar, P. 2020. Using a metaheuristic algorithm for solving a home health care routing and scheduling problem. *Journal of Project Management*, **5**(1), 27–40.

Mankowska, D. S., Meisel, F., & Bierwirth, C. 2014. The home health care routing and scheduling problem with interdependent services. *Health Care Management Science*, **17**(1), 15–30.

Martin, E., Cervantes, A., Saez, Y., & Isasi, P. 2020. IACS-HCSP: Improved ant colony optimization for large-scale home care scheduling problems. *Expert Systems with Applications*, **142**, 112994.

Mavrotas, G., & Florios, K. 2013. An Improved Version of the Augmented ϵ-Constraint Method (AUGMECON2) for Finding the Exact Pareto Set in Multi-Objective Integer Programming Problems. *Appl. Math. Comput.*, **219**(18), 9652–9669.

Maya Duque, P. A., Castro, M., Sörensen, K., & Goos, P. 2015. Home care service planning. The case of Landelijke Thuiszorg. *European Journal of Operational Research*, **243**(1), 292–301.

Milburn, A. B., & Spicer, J. 2013. Multi-objective home health nurse routing with remote monitoring devices. *International Journal of Planning and Scheduling*, **1**(4), 242–263.

Montemanni, R., & Dell'Amico, M. 2023. Solving the Parallel Drone Scheduling Traveling Salesman Problem via Constraint Programming. *Algorithms*, **16**(1).

Mosquera, F., Smet, P., & Vanden Berghe, G. 2019. Flexible home care scheduling. *Omega*, **83**, 80–95.

Moussavi, S., Mahdjoub, M., & Grunder, O. 2019. A matheuristic approach to the integration of worker assignment and vehicle routing problems: Application to home healthcare scheduling. *Expert Systems with Applications*, **125**, 317–332.

Méndez-Fernández, I., Lorenzo-Freire, S., García-Jurado, I., Costa, J., & Carpente, L. 2020. A heuristic approach to the task planning problem in a home care business. *Health Care Management Science*, **23**(4), 556–570.

Méndez-Fernández, I., Lorenzo-Freire, S., & González-Rueda, A. M. 2023a. An Adaptive Large Neighbourhood Search algorithm for a real-world Home Care Scheduling Problem with time windows and dynamic breaks. *Computers & Operations Research.*

Méndez-Fernández, I., Lorenzo-Freire, S., & González-Rueda, A. M. 2023b. A biobjective routing and scheduling problem in a home care business. *Working paper.*

Méndez-Fernández, I., Lorenzo-Freire, S., & González-Rueda, A. M. 2023c. A metaheuristic for solving Home Care Scheduling Problems from scratch. *Working paper.*

Nasir, J. A., & Kuo, Y. H. 2020. A decision support framework for home health care transportation with simultaneous multi-vehicle routing and staff scheduling synchronization. *Decision Support Systems*, **138**, 113361.

Nickel, S., Schröder, M., & Steeg, J. 2012. Mid-term and short-term planning support for home health care services. *European Journal of Operational Research*, **219**(3), 574–587.

Oladzad-Abbasabady, N., & Tavakkoli-Moghaddam, R. 2022. Dynamic routing-scheduling problem for home health care considering caregiver-patient compatibility. *Computers & Operations Research*, **148**, 106000.

Perron, L., & Furnon, V. 2022. *OR-Tools*.

Pisinger, D., & Ropke, S. 2007. A general heuristic for vehicle routing problems. *Computers & Operations Research*, **34**(8), 2403–2435.

Rest, K. D., & Hirsch, P. 2016. Daily scheduling of home health care services using time-dependent public transport. *Flexible Services and Manufacturing Journal*, **28**(3), 495–525.

Riazi, S., Wigström, O., Bengtsson, K., & Lennartson, B. 2019. A Column Generation-Based Gossip Algorithm for Home Healthcare Routing and Scheduling Problems. *IEEE Transactions on Automation Science and Engineering*, **16**(1), 127–137.

Ropke, S., & Pisinger, D. 2006. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, **40**(4), 455–472.

Rossi, F., van Beek, P., & Walsh, T. (eds). 2006. *Handbook of Constraint Programming*. Foundations of Artificial Intelligence, vol. 2. Elsevier.

Schweitzer, F., Bitsch, G., & Louw, L. 2023. Choosing Solution Strategies for Scheduling Automated Guided Vehicles in Production Using Machine Learning. *Applied Sciences*, **13**(2).

Solomon, M. M. 1987. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, **35**(2), 254–265.

Taieb, S. H., Loukil, T., & Mhamedi, A. E. 2019. Home (Health)-Care Routing and Scheduling Problem. *2019 International Colloquium on Logistics and Supply Chain Management (LOGISTIQUA)*, 1–6.

Teichteil-Königsbuch, F., Povéda, G., de Garibay Barba, G. G., Luchterhand, T., & Thiébaux, S. 2023. Fast and Robust Resource-Constrained Scheduling with Graph Neural Networks.

Trautsamwieser, A., & Hirsch, P. 2011. Optimization of daily scheduling for home health care services. *Journal of Applied Operational Research*, **3**(3), 124–136.

Trilling, L., Guinet, A., & Magny, D. L. 2006. Nurse scheduling using integer linear programming and constraint programming. *IFAC Proceedings Volumes*, **39**(3), 671–676.

Van Rossum, G., & Drake, F. L. 2009. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace.

Vieira, B., de Armas, J., & Ramalhinho, H. 2022. Optimizing an integrated home care problem: A heuristic-based decision-support system. *Engineering Applications of Artificial Intelligence*, **114**, 105062.

Wang, H., He, Y., Li, Y., & Wang, F. 2020. Study on the Home Health Caregiver Scheduling Problem under a Resource Sharing Mode considering Differences in Working Time and Customer Satisfaction. *Discrete Dynamics in Nature and Society*, **2020**.

Weil, G., Heus, K., Francois, P., & Poujade, M. 1995. Constraint programming for nurse scheduling. *IEEE Engineering in Medicine and Biology Magazine*, **14**(4), 417–422.

Xiang, T., Li, Y., & Szeto, W. Y. 2023. The daily routing and scheduling problem of home health care: based on costs and participants' preference satisfaction. *International Transactions in Operational Research*, **30**(1), 39–69.

Yang, M., Ni, Y., & Yang, L. 2021. A multi-objective consistent home healthcare routing and scheduling problem in an uncertain environment. *Computers & Industrial Engineering*, **160**, 107560.

Zhan, Y., & Wan, G. 2018. Vehicle routing and appointment scheduling with team assignment for home services. *Computers & Operations Research*, **100**, 1–11.