Trabajo Fin de Máster

Modelización y heurísticas para un problema de planificación de tareas en una empresa de atención a personas dependientes

Isabel Méndez Fernández

Máster en Técnicas Estadísticas

Curso 2017-2018

Propuesta de Trabajo Fin de Máster

Título en galego: Modelización e heurísticas para un problema de planificación de tarefas nunha empresa de atención a persoas dependentes

Título en español: Modelización y heurísticas para un problema de planificación de tareas en una empresa de atención a personas dependientes

English title: Modelling and heuristics for a task planning problem in a care at home business

Modalidad: Modalidad A

Autor/a: Isabel Méndez Fernández, Universidad de Santiago de Compostela

Director/a: Ignacio García Jurado, Universidade da Coruña

Breve resumen del trabajo:

Este TFM se enmarca dentro del proyecto Innterconecta "Generación, Gestión e Integración de Rutas en OLAP (GIRO)", ITC-20151247 (EXP 00082214), cuya investigadora principal es Nieves Rodríguez Brisaboa. En su equipo colaboran Isabel Méndez Fernández e Ignacio García Jurado. Una de las empresas que participan en GIRO es la empresa de atención a personas dependientes MAYORES S.L. Dicha empresa desea disponer de una herramienta para la planificación de los horarios de sus trabajadores, de modo que se cubran las necesidades y se respeten las preferencias de sus clientes, minimizando los tiempos improductivos de sus trabajadores (tiempos de desplazamiento, tiempos de espera entre pares de servicios consecutivos, etc.) Es muy importante que la herramienta sea ágil y rápida porque, por la naturaleza de los servicios que presta la empresa, ésta se enfrenta con considerable frecuencia a imprevistos y cambios que requieren una rápida reacción. Los objetivos de este TFM son:

- Estudiar detenidamente el problema que desea resolver la empresa en todos sus detalles y características específicas.
- Modelizar el problema haciendo uso de la programación lineal entera multiobjetivo.
- Proponer y comparar diversas heurísticas para abordar el problema y encontrar buenas soluciones en tiempos razonables.

Recomendaciones:

Otras observaciones:

Ignacio García Jurado, Catedrático de la Universidade da Coruña, informa que el Trabajo Fin de Máster titulado

Modelización y heurísticas para un problema de planificación de tareas en una empresa de atención a personas dependientes

fue realizado bajo su dirección por Isabel Méndez Fernández para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, da su conformidad para su presentación y defensa ante un tribunal.

En A Coruña, a 2 de Julio de 2018.

El director:

Ignacio García Jurado

La autora:

Isabel Méndez Fernández

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Economóa y Competitividad a través del proyecto MTM2014-53395-C3-1-P, por el Centro para el Desarrollo Tecnológico Industrial a través del proyecto ITC-20151247, y por la Xunta de Galicia mediante las ayudas para Grupos de Referencia Competitiva ED431C-2016-015 y Centro Singular de Investigación de Galicia ED431G/01.

Índice general

Resumen

Resumen en español

Este trabajo se centra en el estudio de un problema relacionado con la planificación de tareas en una empresa de ayuda a domicilio, Mayores, situada en la provincia de A Coruña. Esta empresa permite a personas mayores y/o dependientes continuar viviendo en sus casas, a pesar de pasar por una situación de dependencia o necesidad de apoyo en distintas áreas de su vida diaria.

Para ello, se han estudiado con detalle las características más destacadas del problema que Mayores desea resolver, y el estado del arte de problemas similares. A partir de ello, se ha modelado dicho problema como uno de programación lineal entera donde se recogen todas las propiedades que éste presenta. Debido al gran tamaño de los problemas reales que se deben afrontar, se diseña un algoritmo basado en técnicas heurísticas, que permite dar solución a las incidencias que surgen diariamente en la empresa.

El algoritmo diseñado, se valida mediante la resolución de una batería de ejemplos de pequeña dimensión. En ella, se compara el comportamiento de las soluciones que presenta el algoritmo, con aquellas obtenidas empleando el solver Gurobi para resolver el problema exacto.

Por último, se presentan una serie de casos de uso, en los cuales se utiliza el algoritmo diseñado, para obtener planificaciones de forma automática.

English abstract

This work focuses on the study of a task planning problem in a care at home business called Mayores, located in the province of A Coruña, which allows elderly and/or dependent people to continue living in their homes, in spite of going through a situation of dependency or need for support in different areas of their daily lives.

To this end, we have studied in detail the most outstanding characteristics of the problem that Mayores wishes to solve and, also, various similar problems that have been analised in the literature. Based on this, we modeled a linear programming problem which includes all the properties that our problem presents. Due to the large size of the real problems that must be faced, we desing an algorithms based on heuristic techniques to provide solutions to the incidents that arise daily in the company.

The designed algorithm is validated by the resolution of a battery of examples of small dimension. We compare the behavior of the solutions presented by the algorithm with those obtained using the Gurobi solver to solve the exact problem.

Finally, a series of real-like examples are presented, in which the designed algorithm is used to obtain the automatic planning that solves said examples.

Prefacio

Este trabajo se ha realizado gracias al proyecto Innterconecta "Generación, Gestión e Integración de Rutas en OLAP (GIRO)", ITC-20151247 (EXP 00082214), en el cual seis empresas Gesuga, Biogas Fuel Cell, Grupo On, Mayores, Taprega y Mugatra se unen formando un consorcio y, así, abordar de forma colaborativa la financiación de los trabajos de investigación necesarios para desarrollar las herramientas informáticas que requieren para optimizar sus procesos de negocio.

En el proyecto participan un amplio grupo de investigadores, tanto del CITIC como de la UDC, formado por informáticos y matemáticos, puesto que es necesario desarrollar conjuntamente las aplicaciones informáticas, para digitalizar el trabajo de las empresas, y los algoritmos más adecuados para la resolución de los problemas que presentan.

Todas las empresas del consorcio, aunque tengan ámbitos de trabajo muy dispares, tienen una característica común, y es que sus empleados deben desplazarse constantemente durante su jornada para llevar a cabo las tareas que tienen asignadas. Por tanto, se pretende diseñar herramientas que realicen el cálculo de rutas/horarios con restricciones de forma automática, basándose en técnicas de investigación operativa, tanto obtenidas mediante un estudio del estado del arte, como desarrolladas específicamente para cada empresa.

En particular, este trabajo se centra en Mayores, una empresa que se encarga del cuidado y la atención de personas mayores, tanto en sus domicilios, como en el Centro de Día del que disponen. Esta empresa no sólo trabaja para particulares, sino que también ofrece su servicio a entidades públicas, no lucrativas, y empresas.

El problema que presenta Mayores es uno de planificación de horarios, en el cual se desean determinar las rutas que deben seguir las empleadas de la empresa, y el horario en que deben realizar los servicios que tienen asignados.

El funcionamiento de Mayores y la descripción del problema que quiere resolver se presentan de forma minuciosa en el Capítulo 1 de este trabajo, detallando los conceptos fundamentales que se deben conocer para comprender la actividad que realiza la empresa, así como los objetivos que se desean alcanzar mediante la automatización de las replanificaciones.

En el Capítulo 2, se presenta el estudio del estado del arte llevado a cabo durante la realización de este trabajo. En él se definen los problemas de la literatura que más se asemejan al problema que se debe resolver, como pueden ser el *Vehicle Routing Problem With Time Windows*, que se basa en la planificación de rutas con ventanas de tiempo, o los *Home Care Scheduling Problems*, que estudian problemas de atención a domicilio.

En base a que los problemas estudiados en la literatura no se adaptan por completo al de Mayores, en el Capítulo 3 se modeliza el problema en estudio como uno de programación lineal entera. Para ello se necesita usar un número muy elevado de variables y restricciones que hacen irresolubles los problemas que se necesitan resolver en la práctica. En consecuencia, se necesita diseñar un método heurístico para poder afrontar tales problemas.

En el Capítulo 4, se describe el algoritmo metaheurístico propuesto para resolver el problema de planificación de Mayores. En primer lugar, se define el método *simulated annealing*, seguidamente, se presenta la metodología empleada para obtener el algoritmo definitivo y, por último, se detalla el funcionamiento de las diferentes fases en las que se divide el algoritmo diseñado.

Con el objetivo de estudiar cómo de bueno es el algoritmo propuesto, en el Capítulo 5 se compara

dicho algoritmo con las soluciones exactas obtenidas a partir del problema de programación lineal entera. En primer lugar, se describen las características de los ejemplos resueltos, así como, los parámetros utilizados tanto en el algoritmo como en la obtención de la solución exacta. Seguidamente se comparan ambas técnicas teniendo en cuenta los valores que toman en la función objetivo las soluciones obtenidas por ambos métodos, y los tiempos de computación necesarios para resolver cada uno de los ejemplos considerados.

El Capítulo 6 presenta una breve descripción del funcionamiento de la herramienta informática desarrollada para Mayores y se presentan unos casos de uso en los cuales se utiliza el algoritmo diseñado para resolver un conjunto de incidencias de forma automática.

Por último, en el Capítulo 7 se comentan las conclusiones del trabajo, así como una concisa discusión del posible trabajo que se puede realizar en el futuro sobre la problemática de Mayores.

El trabajo finaliza con dos anexos. En el primero se presenta el pseudocódigo del algoritmo diseñado para resolver el problema en estudio, describiendo con detalle las funciones más importantes que intervienen en él. En el segundo anexo se recoge el código en Java empleado para resolver, utilizando el solver Gurobi, el problema de programación lineal definido en el capítulo 3.

Capítulo 1

La empresa Mayores

En este capítulo se describe, en primer lugar, la actividad que realiza la empresa Mayores, detallando los objetivos que quiere alcanzar y, seguidamente, se definen los conceptos más importantes necesarios para comprender el funcionamiento interno de la empresa y así poder detallar cuál es la situación de ésta en la actualidad.

Por último, se establece el problema que quiere resolver Mayores y que será objeto de estudio de este trabajo, especificando los elementos que se deben tener en cuenta durante su resolución y aquellos objetivos que se desean optimizar durante la misma.

1.1. Introducción

Mayores comienza su actividad en 1997 como empresa de ayuda a domicilio atendiendo a las personas dependientes y a sus familias. La ayuda a domicilio es un servicio de carácter preventivo y normalizador dirigido a las personas y familiares que presentan problemas para la realización de las actividades de la vida diaria, proporcionándoles atención directa en el propio hogar mediante un plan de intervención específico que favorece la permanencia de la persona dependiente en su entorno habitual.

El servicio de ayuda a domicilio está orientado a alcanzar los siguientes objetivos:

- Proporcionar la atención adecuada a personas y familias con dificultades en su autonomía.
- Prevenir situaciones de deterioro personal y social.
- Favorecer la adquisición de habilidades que permitan un desarrollo más autónomo en la vida diaria.
- Posibilitar la integración en el ámbito habitual de convivencia.
- Apoyar a los familiares en sus responsabilidades de atención.
- Evitar o atrasar, mientras no resulte imprescindible, el ingreso de la persona en una residencia.

El principal objetivo de Mayores es mejorar la calidad de vida de las personas dependientes a las que presta servicio facilitando que tengan la máxima autonomía posible para que puedan seguir viviendo en sus casas y, de algún modo, liberar también a sus familias de cierta carga. Para conseguirlo realizan todo tipo de tareas del hogar, atención sobre los pacientes y acompañamiento; destacando por ofrecer soluciones a medida según las necesidades de cada persona dependiente. Sus profesionales, que están en continua formación, saben cómo actuar ante enfermedades que necesitan atenciones tan diferentes como el párkinson, el alzhéimer, la artrosis o la artritis. También cabe señalar que desde la empresa existe un alto compromiso por compartir esta experiencia mediante la investigación y participación en proyectos innovadores en colaboración tanto con entidades públicas como privadas.

1.2. Funcionamiento de la empresa

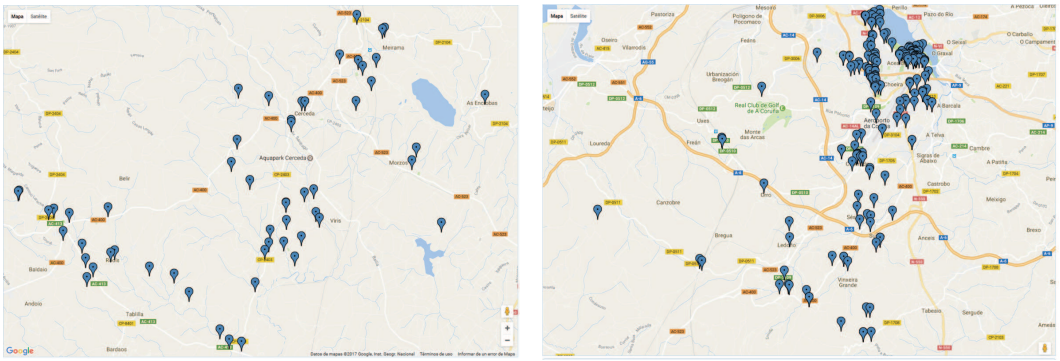
A continuación se definen los conceptos más importantes necesarios para comprender el funcionamiento de la empresa. Así mismo, también se describe cómo el personal de Mayores realiza la planificación de las visitas en la actualidad.

1.2.1. Conceptos básicos

Las nociones básicas, indispensables para poder estudiar con detalle el problema que la empresa quiere resolver, se presentan a continuación.

Usuario Los usuarios son aquellas personas que necesitan cuidados o ayuda en la realización de determinadas tareas y, para mejorar su calidad de vida, recurren a la empresa.

El número de usuarios es muy variable dependiendo de la zona de actuación. En zonas urbanas (como la que se muestra en la figura 1.1b) suelen tenerse más usuarios y con localizaciones mayormente concentradas; en cambio, en zonas rurales (como la que se muestra en la figura 1.1a) se tienen menos usuarios cuya localización suele ser más dispersa.



(a) Cerceda (b) Culleredo

Figura 1.1: Localización de usuarios

Auxiliar Las auxiliares son las personas encargadas de visitar los domicilios de los usuarios y realizar las tareas que les han sido asignadas. Destacar que se utiliza el femenino para referirse a las empleadas de Mayores puesto que, prácticamente su totalidad, está compuesta por mujeres.

Tipo de servicio Tareas que la auxiliar debe realizar cuando se encuentra atendiendo al usuario (por ejemplo: ayudar a mantener la higiene personal, supervisar la medicación prescrita, realizar tareas domésticas, etc.). Los tipos de servicio se dividen en dos categorías:

- Con restricción horaria: algunos cometidos deben realizarse en cierto abanico de horas. Un ejemplo de esto es levantar y acostar al usuario, puesto que estas tareas deben realizarse por la mañana y por la noche.
- Sin restricción horaria: otras tareas pueden realizarse a cualquier hora del día, como por ejemplo limpiar el polvo o poner la lavadora.

Plan de intervención Este plan consiste en definir, para cada usuario, las horas semanales que requiere asistencia de una auxiliar y el tipo de servicio a realizar. Para cada día de la semana se deben especificar las duraciones de los servicios que necesita el usuario (teniendo en cuenta que un día se divide en 4 franjas: mañana, mediodía, tarde y noche; aunque también es posible definir el tramo del día como indistinto) y las actividades a realizar en cada uno de ellos. En

el plan de intervención se determina si el usuario también recibirá a la auxiliar los días festivos (pudiendo ser entre semana y en fines de semana).

El plan de intervención tiene una periodicidad, normalmente semanal, que se repite a lo largo del tiempo. Pero además se tienen servicios excepcionales donde el usuario requiere de una auxiliar, para la realización de ciertas tareas, en un horario ajeno al plan de intervención.

Horario de disponibilidad del usuario El usuario determina un horario en que puede recibir a la auxiliar, denominadas horas disponibles, y dentro de esta ventana se pueden definir unas horas óptimas, que son aquellas que más favorecen al usuario. Un ejemplo de esto es que haya que levantar y dar el desayuno a un usuario cuyas horas disponibles son de 7:00 a 10:00, pero prefiere que se le atienda a las 9:00.

Plan de trabajo El plan de trabajo consiste en concretar el plan de intervención, es decir, a cada usuario se le determina la auxiliar (o auxiliares) que lo atenderá, el horario en que se llevarán a cabo los servicios y las tareas a realizar durante los mismos.

Contrato de las auxiliares Las auxiliares tienen contratos en los que se determina el número de horas que deben trabajar a la semana. Estos contratos generan una amplia variedad de jornadas máximas permitidas que, además, son susceptibles de ser ampliadas o reducidas dependiendo de las necesidades de la empresa y los usuarios.

Las auxiliares pueden estar contratadas unicamente para los fines de semana, para los días festivos, solamente para los días de semana/laborables, etc. Por tanto, este aspecto es de vital importancia a la hora de asignar los servicios a las auxiliares.

Restricciones en el horario Las auxiliares necesitan tener descansos, los cuales se rigen por las siguientes normas:

- No se puede trabajar más de 6 horas seguidas sin tomar un descanso de 30 minutos (pagado).
- Si una auxiliar tiene un hueco en el horario de más de 2 horas se considera que trabaja a jornada partida y el lapso de tiempo no se paga.
- Si una auxiliar tiene varios huecos de más de 2 horas en la jornada laboral el único que no se paga es el que tiene mayor duración.

Organización La empresa dispone de unas coordinadoras que tienen asignados unos usuarios y unas auxiliares. Dicha asignación se realiza teniendo en cuenta la ubicación de los usuarios, puesto que las coordinadoras suelen trabajar por zonas o barrios. La asignación de usuarios no se puede modificar puesto que las coordinadoras están en contacto constante con los familiares de los mismos (se redactan informes, hay un seguimiento de los usuarios, etc).

Para atender a estos usuarios, las coordinadoras solamente trabajan con el grupo de auxiliares que tienen a su cargo, cuyo tamaño depende de la demanda de servicios que presentan los usuarios que tienen a su cargo.

Las coordinadoras son las encargadas de organizar manualmente los planes de trabajo de las auxiliares, de modo que se satisfagan las necesidades de todos los usuarios asignados a ellas. Es decir, planifican los servicios que requieren los usuarios, asignándolos a las auxiliares y especificando el horario en que se llevarán a cabo.

Tiempos de desplazamiento Las auxiliares invierten una parte de su jornada laboral en desplazarse de la vivienda de un usuario a la del siguiente. Mayores solamente cobra las horas en que las auxiliares están con los usuarios, de modo que, el tiempo invertido en desplazamientos proporciona perdidas a la empresa.

Los tiempos varían mucho dependiendo de la zona de actuación. En una zona rural, como la de la figura 1.1a, los usuarios están muy dispersos y básicamente se requiere que las auxiliares

utilicen como medio de transporte su vehículo propio. En zonas urbanas, como la que se refleja en la figura 1.1b, los usuarios están más concentrados y muchas veces el desplazamiento de las auxiliares se realiza a pie.

Incidencias En ocasiones no puede llevarse a cabo el plan de trabajo, lo cual puede deberse a dos motivos:

- Los usuarios no pueden recibir a las auxiliares, por ejemplo: el usuario está hospitalizado, de vacaciones, etc. En este caso la auxiliar que debería de atender al usuario queda libre. Si la incidencia no se avisa con cierto tiempo de antelación al usuario se le cobra el servicio de todas formas.
- Las auxiliares no están disponibles, esto puede darse en multitud de ocasiones puesto que las auxiliares pueden ponerse enfermas, tomar vacaciones, etc. En este caso el usuario necesita que se le asigne otra auxiliar ya que no pueden quedar servicios desatendidos.

Las coordinadoras son las encargadas de resolver las incidencias que le llegan, ya sean producidas por los usuarios o por las auxiliares que tienen asignadas. Estas incidencias suelen llegarle a la coordinadora durante el día, puesto que de esta forma es más sencillo ponerse en contacto con ella.

Bolsa de horas En esta bolsa se lleva la cuenta de las horas que ha trabajado de más (por ejemplo: cubrir por una compañera que un día no estaba) o de menos (por ejemplo: quedar libre un día por incidencia del usuario) cada una de las auxiliares. De forma que, si la auxiliar ha trabajado más de lo que se especificaba en su contrato, estas horas extras se suman a la cantidad que tenía acumulada en su bolsa de horas y, si por el contrario, la jornada trabajada de la auxiliar no alcanza la estipulada en su contrato, las horas no completadas se restan de la cantidad almacenada. Así mismo, la bolsa de horas se pone a cero de forma trimestral.

1.2.2. Situación actual de la empresa

En la actualidad las coordinadoras son las encargadas de realizar toda la planificación de forma manual, de modo que determinan las asignaciones de usuarios a auxiliares, así como, los horarios en los cuales se realizarán las visitas.

En el momento en que un nuevo usuario, o sus familiares, se ponen en contacto con la empresa éstos deben especificar el plan de intervención del mismo, es decir, el número de horas que requiere de la asistencia de una auxiliar y las tareas a realizar en cada una de las visitas. Tras ello el usuario pasa a estar *pendiente de alta* y cuando la coordinadora a la que dicho usuario ha sido asignado (normalmente esta asignación se realiza por zonas) lo considera oportuno le da de *alta* y determina el plan de trabajo. Para ello, la coordinadora estudia el horario de disponibilidad de los servicios del usuario y la planificación actual de las auxiliares de las que dispone, con el fin de determinar la auxiliar (o auxiliares) más adecuadas para atender al usuario y establecer el horario en que se realizará cada uno de dichos servicios.

Las coordinadoras también son las encargadas de replanificar los horarios de las auxiliares en el caso de que un usuario se dé de baja del sistema, puesto que la auxiliar (o auxiliares) que lo atendían pueden pasar a tener huecos en su agenda, los cuales deben ser eliminados. Estos huecos pueden ser cubiertos mediante la asignación de otras visitas a las auxiliares, mediante la modificación de los horarios de los servicios que realizan las auxiliares, o pueden quedar descubiertos dando lugar a la reducción de las horas estipuladas en el contrato de las auxiliares.

En el momento en que un usuario produce una incidencia puntual (ausencia por hospitalización, vacaciones, etc), la auxiliar que debería realizar la visita se queda libre durante el período de tiempo en que debería realizarse dicha visita. Por tanto, la coordinadora encargada de ese usuario y de dicha auxiliar tiene que decidir la mejor forma de resolver esta incidencia, teniendo en cuenta la bolsa de horas, el resto de incidencias, la planificación de la auxiliar, los tiempos de desplazamiento empleados

en ir de un usuario a otro, etc. Es decir, la coordinadora debe determinar si a la auxiliar le puede asignar otra visita o si, en cambio, se tiene que dejar el hueco libre.

Si la incidencia a tratar es producida por una auxiliar (enfermedad, vacaciones, día libre, etc) la coordinadora deberá reasignar todas las visitas que tenía asignada dicha auxiliar, pasando a ser realizados por alguna de las auxiliares de las que dispone en ese momento. Para ello debe de tener en cuenta la posible existencia de otras incidencias, la bolsa de horas, las planificaciones previas, los tiempos de desplazamiento entre usuarios y que todos los usuarios deben ser atenidos. Las auxiliares que han trabajado horas de menos, o que tengan huecos en su planificación, serán las seleccionadas para realizar dichos servicios.

1.3. Problema que presenta la empresa

El objetivo de la empresa es que las replanificaciones, que actualmente son llevadas a cabo por las coordinadoras de forma manual, se realicen de forma automática mediante el uso de una aplicación informática. Cabe recalcar que, en ningún momento, la empresa pretende prescindir de las coordinadoras, simplemente facilitar su trabajo al proporcionarles una herramienta de planificación automática. Las coordinadoras deben estar presentes para comprobar las planificaciones obtenidas de forma automática y, en caso de que sea necesario, realizar las modificaciones pertinentes.

La empresa presenta dos problemas de planificación de horarios a resolver, que se detallan a continuación.

1.3.1. Descripción del problema

El primer problema que la empresa quiere resolver, y en el que se centra este trabajo, es una *replanificación general*, cuyo objetivo consiste en obtener una modificación de la planificación previa para así solventar la aparición de una incidencia (o conjunto de incidencias) de efecto general importante. Las modificaciones de las planificaciones se llevan a cabo cuando se da alguno de los siguientes casos:

- Se quiere dar de alta a un usuario. En este caso, la coordinadora determina que se va a incorporar un nuevo usuario a las planificaciones, de modo que coloca cada uno de sus servicios en el horario que considera más apropiado dentro de la planificación de la auxiliar más adecuada para ello.
- Se quiere dar de baja a un usuario. En este caso, un usuario se da de baja del servicio (ya sea por ingreso en hospital, vacaciones, fallecimiento, etc.) y la coordinadora elimina los servicios asignados a dicho usuario de las planificaciones de las auxiliares que lo atendían. Al suprimir estos servicios pueden generarse huecos en las jornadas de las auxiliares; por tanto, la coordinadora debe reajustar los horarios en los que las auxiliares llevan a cabo los servicios para poder eliminar dichos huecos.
- Modificaciones en el número de servicios. En este caso, un usuario que ya está siendo atendido altera el número de servicios que requiere, pudiendo darse dos casos:
  1. Aumento de servicios, el usuario amplía el número de servicios que requiere, en cuyo caso la coordinadora coloca estos nuevos servicios en la planificación de la auxiliar que considera más adecuada para llevarlos a cabo.
  2. Reducción de servicios, el usuario disminuye el número de servicios que requiere, en cuyo caso la coordinadora elimina los servicios sobrantes y modifica los horarios de la auxiliar (o auxiliares) que lo atendía con el fin de eliminar posibles huecos en su planificación.
- Modificación de alguna característica de un servicio, como puede ser aumento (disminución) de la duración del servicio o cambios en la ventana disponible del servicio. La coordinadora, en estos casos, debe modificar las planificaciones de las auxiliares con el objetivo de que todos los servicios estén correctamente planificados.

- Combinación de casos. Una coordinadora puede decidir combinar los casos anteriores si lo cree conveniente. Por ejemplo, puede dar de alta a un nuevo usuario y al mismo tiempo dar de baja a otro, o también puede dar de baja a un usuario y ampliar el número de servicios que requiere otro usuario.
- Repetición de casos. Una coordinadora puede modificar la planificación con el fin de resolver una única incidencia, pero para varios usuarios. Por ejemplo, se puede dar de baja (o de alta) a varios usuarios, se puede aumentar el número de servicios de varios usuarios que lo requieran, etc.

En este caso la nueva planificación se obtiene con el fin de repetirse semanalmente, comenzando en el instante en que se da la incidencia y finalizando en el momento en que se vuelva a producir otra incidencia de este tipo (momento en el que será necesario volver a obtener una nueva planificación).

El segundo problema a resolver, y que puede ser objetivo de estudio en el futuro, es una *replanificación puntual*, que consiste en obtener una replanificación para solventar una incidencia (o varias) de efecto puntual. Es decir, incidencias que se producen de forma esporádica, como pueden ser vacaciones de las auxiliares o ausencias puntuales de los usuarios, y que tras el paso del tiempo desaparecen. La solución para este problema solamente se aplica en las fechas en que se presenta dicha incidencia, de modo que en el momento en que la incidencia desaparece se vuelve a utilizar la planificación original.

En ambos problemas la coordinadora supervisará la planificación obtenida de forma automática y, en caso de que lo crea conveniente, puede realizar modificaciones en los horarios de los servicios y en las asignaciones de forma manual.

1.3.2. Elementos a tener en cuenta

Los problemas definidos en la subsección 1.3.1 deben tener en cuenta los elementos que se describen a continuación para que las soluciones que proporcionen sean las más adecuadas para la empresa. Estas características son las que se utilizarán para determinar la modelización matemática del problema y para la implementación de algoritmos heurísticos.

Replanificación El objetivo de este problema no es la creación de una nueva planificación de los horarios de las auxiliares, sino la modificación (realizando el menor número de cambios posible) del plan previo para solventar las incidencias que se presenten.

Jornada laboral La jornada laboral de cada auxiliar se corresponde con el número de horas que ha trabajado y consta de los siguientes elementos:

- La duración de todos los servicios que lleva a cabo a lo largo del día.
- Los tiempos empleados en desplazarse entre las viviendas de dos usuarios con servicios consecutivos, es decir, servicios entre los cuales la auxiliar solamente tiene tiempo para desplazarse de uno a otro.
- Los tiempos de descanso (los tiempos en que la auxiliar no está ni realizando un servicio ni desplazándose hacia la vivienda del siguiente usuario al que tiene que atender) que tiene la auxiliar durante su jornada diaria, cuya duración es inferior a 2 horas.
- Los tiempos de descanso que tiene la auxiliar durante su jornada diaria cuya duración es mayor de 2 horas, siempre y cuando haya algún otro descanso de mayor duración.

De modo que los siguientes tiempos no se consideran a la hora de calcular las jornadas de trabajo de las auxiliares:

- El mayor tiempo de descanso en la planificación diaria de la auxiliar, siempre y cuando dicho descanso tenga una duración mayor o igual a 2 horas. En este caso, tampoco se considera para el cálculo de la jornada, el tiempo de desplazamiento entre los servicios que generan el descanso.

- El tiempo que invierte la auxiliar en desplazarse desde su domicilio hasta la vivienda del primer usuario al que debe atender.
- El tiempo que emplea la auxiliar en desplazarse desde la vivienda del último usuario que debe atender hasta su domicilio personal.

Bolsa de horas La bolsa de horas es un aspecto muy importante en la modelización del problema, puesto que es necesario tener en cuenta el número de horas que cada auxiliar ha trabajado de más o de menos a la hora de obtener las nuevas soluciones. Esto se debe a que, en caso de que una auxiliar haya trabajado horas de más, la solución puede otorgarle descansos, lo que implicaría una reducción en su bolsa de horas. Mientras que, si una auxiliar ha trabajado de menos, la solución puede asignarle más visitas, sin necesidad de modificar su contrato, lo que acarrearía que las horas que ha trabajado de menos disminuyan.

Cambios en las asignaciones La empresa considera muy importante la fidelidad de las asignaciones de usuarios a auxiliares, llegándose al caso de que es preferible modificar el horario de una visita con el fin de que la auxiliar que la realice pueda ser siempre la misma. Por tanto, es de vital importancia no modificar, siempre que sea posible, las asignaciones de auxiliares a usuarios.

Afinidades Como se explica en el punto anterior, es muy importante minimizar el número de cambios en las asignaciones de auxiliares a los pacientes, pero en ocasiones es inevitable que un usuario sea atendido por más de una auxiliar. La empresa puede utilizar el histórico de las asignaciones, y las posibles preferencias o problemas que hayan surgido, para obtener un código de afinidad para cada asignación auxiliar-usuario.

Los niveles de afinidad considerados son los siguientes:

- La auxiliar ha atendido de forma satisfactoria al usuario:

  Nivel 5 El nivel máximo de afinidad se considera cuando la auxiliar atiende, de forma continuada, al usuario.

  Nivel 4 Este nivel de afinidad se da cuando la auxiliar atendió (de forma esporádica o continuada) al usuario en el pasado pero dejó de hacerlo, ya sea por cambio de auxiliar, baja, hospitalización, etc.

- La auxiliar no ha atendido al usuario en el pasado:

  Nivel 3 Este nivel de afinidad se tiene cuando la auxiliar posee algún tipo de característica necesaria para atender al usuario. Estas características pueden ser: capacidad para administrar medicación, conocimientos para utilizar cierto tipo de maquinaria, etc.

  Nivel 2 Este nivel es el término medio a priori, se da en casos en los que la auxiliar y el usuario no han tenido interacción alguna en el pasado y el usuario no requiere de ninguna característica especial de la auxiliar.

- La auxiliar ha recibido quejas por parte del usuario:

  Nivel 1 Este nivel de afinidad se tiene cuando la auxiliar que atiende (o atendía) a un usuario recibe algún tipo de queja leve por parte de éste. De modo que es recomendable que la auxiliar no atienda al usuario en el futuro, aunque podría utilizarse como último recurso.

  Nivel 0 Este nivel de afinidad se da en las ocasiones en las que un usuario ha vetado a una auxiliar, es decir, ha habido quejas o enfrentamientos graves, y por tanto no quiere que dicha auxiliar lo atienda en el futuro.

De este modo se debe integrar este código en el algoritmo de resolución de forma que se realicen las asignaciones más favorables para los usuarios.

Incidencias La empresa se enfrenta a un gran número de incidencias lo que hace aumentar la complejidad del problema. El sistema debe ser capaz de solventar todas las incidencias que se presentan, así como, de permitir que la coordinadora decida que incidencias quiere resolver en un momento determinado.

Modificaciones de los contratos Las auxiliares tienen el número de horas que deben trabajar (normalmente es un máximo semanal, pero también puede ser diario) especificado en su contrato, que debería respetarse. Debido al gran número de altas y bajas de usuarios, la empresa realiza frecuentemente modificaciones en el número de horas estipuladas en los contratos de las auxiliares.

Tiempos de desplazamiento Las auxiliares están constantemente desplazándose entre las viviendas de los usuarios, de modo que, con el objetivo de minimizar las pérdidas de dinero de la empresa, la solución debe tratar de minimizar los desplazamientos de las auxiliares.

Por otro lado, para que una planificación sea correcta es necesario que la auxiliar a la que corresponde pueda llevarla a cabo respetando los horarios estipulados en ella. Por tanto, será necesario que entre el instante en que se debe finalizar un servicio y el instante en que debe comenzar el siguiente, la auxiliar tenga el tiempo suficiente como para poder desplazarse desde la vivienda del primer usuario hasta la del segundo.

Descansos de las auxiliares Las auxiliares, durante su jornada trabajada, pueden tener huecos en la planificación debido a las restricciones horarias de los servicios que deben realizar. Debido a esto es necesario que las planificaciones sean tales que el número de descansos entre servicios, y su duración, sean lo menor posible; puesto que de lo contrario se generarían más perdidas para la empresa.

Realización de servicios Todos los usuarios requieren de una serie de servicios que deben ser realizados por alguna auxiliar. Por tanto, las soluciones solamente serán aceptables cuando todos los servicios, de los usuarios que se estén contemplando en ese momento, se encuentran en la planificación de una auxiliar.

Plan de intervención Los usuarios tienen definido un plan de intervención en el que se determina el número de horas semanales que requiere asistencia de una auxiliar, así como, la duración de los servicios y si se desean realizar dichos servicios también en días festivos. Las soluciones obtenidas de forma automática deben cumplir con el plan de intervención, de modo que todos los servicios estén correctamente planificados (respetar su duración, día de la semana y su ventana disponible).

Limitaciones en el horario Como ya se comentaba anteriormente, los usuarios tienen unos horarios disponibles en los que deben ser atendidos y unos horarios óptimos que muestran sus preferencias. Estos horarios deben tenerse en cuenta a la hora de generar soluciones ya que la planificación obtenida debe respetar siempre los horarios disponibles impuestos por el usuario y, al mismo tiempo, debe tratar de maximizar los servicios que se realizan dentro de sus ventanas óptimas.

Modificaciones en los inicios de las visitas En ocasiones, es posible modificar el inicio de varias visitas consecutivas de una auxiliar de modo que, tras ello, se obtenga un hueco en el horario de la auxiliar en el cual se puede realizar otra visita. El sistema debe ser capaz de realizar estas modificaciones en la planificación de manera que, respetando las restricciones antes mencionadas, se optimice la jornada de las auxiliares.

1.3.3. Objetivos

Los objetivos que se desean optimizar durante la resolución del problema de replanificación general que presenta Mayores son los siguientes:

Minimizar desplazamientos. Las auxiliares se trasladan entre las viviendas de los auxiliares que atienden para cumplir con su planificación y por tanto, el tiempo que emplean en desplazamientos está dentro de su jornada laboral. De modo que uno de los objetivos de la empresa es que el sistema proporcione soluciones que minimicen el tiempo empleado en desplazamientos.

Maximizar afinidades. Como ya se ha comentado anteriormente, se trabaja con unas afinidades entre usuarios y auxiliares que determina cómo de recomendable es que una auxiliar se ocupe de un usuario. La empresa quiere que los usuarios estén satisfechos con las auxiliares; por lo tanto, será necesario maximizar la afinidad entre los usuarios y las auxiliares que los atienden.

Maximizar ventanas óptimas. Los usuarios determinan, para cada uno de sus servicios, un horario óptimo dentro del cual les gustaría que se llevara a cabo el mismo. Aunque esta restricción no es indispensable para que la planificación se considere correcta, es recomendable que los servicios se realicen dentro de las ventanas óptimas. Por ello se debe maximizar el número de servicios que están planificados dentro del horario óptimo.

Minimizar los cambios de las planificaciones. Se ha explicado con anterioridad que para la empresa es muy importante que a los usuarios siempre los atiendan las mismas auxiliares, salvo en el caso de que haya quejas, y en horarios consistentes Por tanto, es recomendable que las planificaciones, que se generen de forma automática, traten de minimizar el número de veces que se modifica la auxiliar que va a llevar a cabo alguno de los servicios de un usuario, y que se realicen el menor número de modificaciones en los horarios de los mismos.

Minimizar los descansos. Tal y como se explicaba en 1.3.2, todos los descansos (a excepción del más largo, siempre y cuando tenga una duración superior a dos horas) que realizan las auxiliares en un día, forman parte de su jornada laboral trabajada. Para optimizar la planificación de las auxiliares será necesario minimizar sus tiempos libres, es decir, los tiempos que no están realizando servicios, sin considerar entre éstos el mayor de los descansos de duración superior (o igual) a dos horas.

Capítulo 2

Estado del arte

El objetivo principal de este capítulo es presentar los trabajos de la literatura que han sido revisados para facilitar el abordaje del problema de Mayores.

En primer lugar, se define el problema de rutas de vehículos, debido a que una de las principales características del problema en estudio es que las auxiliares recorren rutas para llevar a cabo los servicios que tienen asignados en su planificación. Seguidamente, se presenta el problema de planificación de enfermeras y dos problemas de atención a domicilio, con el objetivo de buscar similitudes entre ellos y el problema de replanificación general en estudio. Para todos estos problemas se definen las variables que intervienen en ellos, las restricciones que los determinan y los objetivos a optimizar durante su resolución.

El capítulo finaliza con una breve comparación entre el problema de Mayores y los problemas estudiados, de modo que se determinen aquellos aspectos que pueden utilizarse durante la modelización del problema que nos atañe.

2.1. Vehicle Routing Problem

El *Vehicle Routing Problem* (VRP) tiene como objetivo principal determinar el conjunto óptimo de rutas que debe realizar una flota de vehículos para satisfacer la demanda de un conjunto de consumidores, o clientes, que se encuentran en ubicaciones geográficamente dispersas, minimizando los costes que genera la realización de esas rutas. Este problema aparece por primera vez en Dantzig & Ramser (1959); en este artículo se estudiaba la obtención de las rutas óptimas que debían seguir los camiones de una flota de reparto de gasolina, que tenían que abastecer a una gran cantidad de estaciones de servicio desde una terminal.

A continuación, se describe el modelo del problema, definiendo los conjuntos, parámetros, variables, restricciones y objetivos que lo determinan.

2.1.1. Conjuntos, parámetros y variables

Los conjuntos, parámetros y variables que intervienen en el problema son los siguientes:

- $G = (V, A)$ es un grafo donde:
  - $V = \{0, 1, ..., n, n+1\}$ es el conjunto de vértices representando las ubicaciones de los clientes $N = \{1, .., n\}$, donde el vértice 0 y el $n + 1$ representan el almacén de donde salen todos los vehículos.
  - $A = \{(i, j) / i \neq j, i, j \in V\}$ es el conjunto de arcos que unen los vértices. Asociada a éstos se tiene la matriz $C = (c_{ij})$ que representa los costes que suponen desplazarse desde el vértice $i \in V$ hasta el $j \in V$; este coste puede representar tiempo, distancia, dinero, etc.

- $M = \{1, ..., m\}$ representa el conjunto de vehículos disponibles en el almacén; estos vehículos se consideran iguales en cuanto a los costes que provocan.
- Q determina la capacidad máxima de cada vehículo (se consideran todos iguales).
- q_i establece la cantidad de producto que hay que entregar al cliente ($i \in V$).
- x_{ijk} es la variable binaria que toma el valor 1 si el vehículo $k \in M$ se desplaza desde la ubicación $i \in V$ a la $j \in V$ y 0 en cualquier otro caso.
- $\delta^+(i) = \{j/(i,j) \in A\}$ y $\delta^-(j) = \{i/(i,j) \in A\}$ los conjuntos de arcos de salida y entrada de los vértices.

2.1.2. Problema

Por último, se muestran las restricciones del problema y la función objetivo a optimizar.

1. Cada cliente debe ser visitado una única vez por un único vehículo

$$\sum_{k \in M} \sum_{j \in \delta^+(i)} x_{ijk} = 1 \quad i \in N$$

2. Si un vehículo visita a un cliente también debe abandonarlo

$$\sum_{i \in \delta^-(j)} x_{ijk} - \sum_{i \in \delta^+(j)} x_{jik} = 0 \quad j \in N \; k \in M$$

3. Los vehículos deben comenzar su ruta en el almacén

$$\sum_{i \in \delta^+(0)} x_{0jk} = 1 \quad k \in M$$

4. Los vehículos deben finalizar su ruta en el almacén

$$\sum_{i \in \delta^-(n+1)} x_{i(n+1)k} = 1 \quad k \in M$$

5. Los vehículos sólo pueden ser utilizados una vez

$$\sum_{j=0}^{n} x_{0jk} = 1 \quad k \in M$$

6. Debe respetarse la capacidad máxima de los camiones

$$\sum_{i \in N} q_i \sum_{j \in \delta^+(i)} x_{ijk} \leq Q \quad k \in M$$

El objetivo de este problema es minimizar los costes, es decir, minimizar la función:

$$\sum_{i=0}^{n+1} \sum_{j=0}^{n+1} \sum_{k=1}^{m} c_{ij} x_{ijk}$$

2.2. Vehicle routing problem with time windows

El *Vehicle Routing Problem with Time Windows* (VRPTW) es una generalización del VRP en la que cada cliente debe ser atendido dentro de una ventana de tiempo disponible. Por tanto, los vehículos no pueden llegar antes de que comience la ventana de tiempo y tampoco llegar más tarde del final de la misma. Este problema es estudiado por primera vez cuando Pullen & Webb (1967) planifican las rutas de los vehículos del servicio postal del área central de Londres.

A continuación, se describe el modelo del problema, definiendo los elementos, restricciones y objetivos que lo determinan.

2.2.1. Conjuntos, parámetros y variables

Los conjuntos, parámetros y variables que intervienen en el problema son los siguientes:

- $G = (V, A)$ es un grafo donde:
  - $V = \{0, 1, ..., n, n+1\}$ es el conjunto de vértices representando las ubicaciones de los clientes $N = \{1, .., n\}$, donde el vértice 0 y el $n + 1$ representan el almacén de donde salen todos los vehículos.
  - $A = \{(i, j) \ / \ i \neq j,\ i, j \in V\}$ es el conjunto de arcos que une los vértices, asociado a estos arcos se tiene la matriz $C = (c_{ij})$ que representa los costes que supone desplazarse desde el vértice $i \in V$ hasta el $j \in V$; este coste puede representar tiempo, distancia, dinero, etc.
- $M = \{1, ..., m\}$ representa el conjunto de vehículos disponibles en el almacén, estos vehículos se consideran iguales en cuanto a los costes que provocan.
- s_i es el tiempo de servicio del cliente $i \in V$, es decir, el tiempo que se emplea en servir al cliente, de modo que $s_0 = s_{n+1} = 0$ puesto que el nodo 0 y el $n + 1$ representan al almacén.
- Q determina la capacidad máxima de cada vehículo (se consideran todos los vehículos iguales).
- q_i establece la cantidad de producto que hay que entregar al cliente $i \in V$.
- t_{ij} es el tiempo que tarda un vehículo en ir desde la ubicación del cliente $i \in N$ a la de $j \in N$.
- $[a_i, b_i]$ especifica la ventana de tiempo dentro de la cual debe ser servido el cliente $i \in N$. Si no se tienen restricciones en la disponibilidad de los vehículos se puede tomar $a_0 = \text{mín}_{i \in N}\, a_i - t_{0i}$, $b_0 = \text{máx}_{i \in N}\, b_i - t_{0i}$, $a_{n+1} = \text{mín}_{i \in N}\, a_i + s_i + t_{in+1}$ y $b_{n+1} = \text{máx}_{i \in N}\, b_i + s_i + t_{in+1}$
- x_{ijk} es la variable binaria que toma el valor 1 si el vehículo $k \in M$ se desplaza desde la ubicación $i \in V$ a la $j \in V$ y 0 en cualquier otro caso.
- w_{ik} es la variable que determina el instante de tiempo en que el vehículo $k \in M$ comienza a servir al cliente $i \in V$.
- $\delta^+(i) = \{j/(i, j) \in A\}$ y $\delta^-(j) = \{i/(i, j) \in A\}$ los conjuntos de arcos de salida y entrada de los vértices.

2.2.2. Problema

Por último, se presentan las restricciones del problema y la función objetivo a optimizar.

1. Cada cada cliente debe ser visitado una única vez por un único vehículo

$$\sum_{k \in M} \sum_{j \in \delta^+(i)} x_{ijk} = 1 \quad i \in N$$

2. Si un vehículo visita a un cliente también debe abandonarlo

$$\sum_{i\in\delta^-(j)} x_{ijk} - \sum_{i\in\delta^+(j)} x_{jik} = 0 \quad j \in N \; k \in M$$

3. Los vehículos deben comenzar su ruta en el almacén

$$\sum_{i\in\delta^+(0)} x_{0jk} = 1 \quad k \in M$$

4. Los vehículos deben finalizar su ruta en el almacén

$$\sum_{i\in\delta^-(n+1)} x_{i(n+1)k} = 1 \quad k \in M$$

5. Debe respetarse la capacidad máxima de los camiones

$$\sum_{i\in N} q_i \sum_{j\in\delta^+(i)} x_{ijk} \leq Q \quad k \in M$$

6. Deben respetarse las ventanas de tiempo de los clientes

$$a_i \leq w_{ik} \leq b_i \quad i \in V \; k \in M$$

7. Las variables que determinan el inicio del servicio deben ser consistentes, es decir, deben respetar la duración de los servicios y el tiempo empleado en desplazarse entre dos servicios consecutivos

$$x_{ijk}(w_{ik} + s_i + t_{ij} - w_{jk}) = 0 \quad (i,j) \in A \; k \in M$$

La función objetivo de este problema de minimización de costes es

$$\sum_{i=0}^{n+1}\sum_{j=0}^{n+1}\sum_{k=1}^{m} c_{ij}x_{ijk}$$

2.3. Nurse Scheduling Problem

El *Nurse Schedulling Problem*, Dowsland (1998), es un problema de asignación que consiste en determinar la planificación de turnos[1] que se debe asignar a cada enfermera para la semana considerada, con el fin de que en todos los turnos se tenga un número mínimo de enfermeras de determinado nivel.

A continuación, se describe el modelo del problema, definiendo los conjuntos, parámetros, variables, restricciones y objetivos que lo determinan.

2.3.1. Conjuntos, parámetros y variables

A continuación de definen los conjuntos, parámetros y variables que intervienen en el problema.

- $N = \{1, ..., n\}$ es el conjunto de enfermeras disponibles.
- $R = \{1, ..., r\}$ es el conjunto de niveles que presentan las enfermeras.

[1] La organización de los turnos de hospital es compleja, puesto que se tienen distintos tipos de servicio y una amplia variedad de turnos, pero en este problema se suponen conocidas todas las posibles planificaciones de turnos que pueden asignarse a una enfermera durante una semana.

- $K = \{1, ..., k\}$ es el conjunto de posibles turnos.
- G_s es el conjunto de enfermeras de nivel $s \in R$ o superior.
- R_{ls} determina el mínimo número de enfermeras de nivel $s \in R$ que se necesitan en el turno $l \in K$.
- F_i fija el conjunto de posibles planificaciones de turnos que puede realizar la enfermera $i \in N$.
- a_{il} es una variable binaria que toma el valor 1 si la planificación j cubre el turno $l \in K$.
- p_{ij} especifica el coste de penalización asignado a que la enfermera $i \in N$ realice la planificación $j \in F_i$.
- x_{ij} es la variable binaria de decisión que toma el valor 1 si la enfermera $i \in N$ realiza la planificación $j \in F_i$ y el valor 0 en caso contrario.

2.3.2. Problema

El objetivo del NSP es minimizar los costes de asignar las enfermeras a las planificaciones, es decir, minimizar la función siguiente.

$$\sum_{i=1}^{n} \sum_{j \in F_i} p_{ij} x_{ij}$$

Las restricciones a tener en cuenta son las siguientes:

1. Cada enfermera solamente puede realizar una planificación

$$\sum_{j \in F_i} x_{ij} = 1 \quad \forall i \in N$$

2. En cada turno debe haber el número suficiente de enfermera con el nivel que se requiere

$$\sum_{i \in G_s} \sum_{j \in F_i} a_{jl} x_{ij} \geq R_{js} \quad \forall l \in K, \forall s \in R$$

2.4. Home care scheduling problems

En esta sección se describen brevemente los problemas encontrados en la literatura que se asemejan más al de Mayores. Estos problemas, *Home care scheduling problems* (problemas de atención a domicilio), se basan en el estudio de casos en los que las enfermeras deben desplazarse a las viviendas de los pacientes para atenderles.

2.4.1. Multimodal Home-Healthcare Scheduling

En Rendl *et al.* (2012), se describe el *Multimodal Home-Healthcare Scheduling Problem* (MHHS), cuyo objetivo es encontrar una asignación de enfermeras a pacientes y determinar los recorridos que deben seguir dichas enfermeras, donde el término multimodal se refiere a que las enfermeras tienen diferentes medios de transporte. Este problema se modela como una combinación del VRPTW y del NSP.

A continuación, se describe el modelo del problema, definiendo los elementos, restricciones y objetivos que lo determinan.

Conjuntos, parámetros y variables

Los conjuntos, parámetros y variables que intervienen en el problema se definen a continuación.

- $\mathcal{N} = \{1, ..., N\}$ es el conjunto de enfermeras. Para cada $n \in \mathcal{N}$ se tiene que h_{min}, h_{max} son las horas de trabajo mínima y máximas diarias.
- $TW_n = \{[a_{n1}, b_{n1}], ..., [a_{nm}, b_{nm}]\}$ representa el conjunto de ventanas de tiempo disponibles de una enfermera $n \in \mathcal{N}$.
- $\mathcal{C} = \{1, ..., C\}$ es el conjunto de clientes.
- $\mathcal{T} = \{1, ..., T\}$ representa el conjunto de puntos que dividen el día en T unidades de τ minutos.
- $\mathcal{J} = \{1, ..., J\}$ es el conjunto de trabajos. Para cada trabajo $j \in \mathcal{J}$ se tiene asociada una ventana de tiempo $[s_j, e_j]$, donde s_j es el tiempo más temprano para empezar j y e_j es el tiempo más tardío para empezar j. También se tiene $s_j \leq t_j \leq e_j$, donde t_j es el tiempo preferido por el usuario. La duración de un trabajo es dur_j. Tanto s_j, e_j como t_j son elementos de $\mathcal{T}$, mientras que $dur_j \in \mathbb{N}$.

  Los trabajos se diferencian en dos grupos, $\mathcal{J}^{pre}$ que son los trabajos fijados (es decir, trabajos que tienen tienen confirmado su horario y la enfermera asignada a ellos, como pueden ser reuniones, trabajos administrativos, etc) y $\mathcal{J}^{\overline{pre}}$ que son los trabajos no fijados (es decir, aquellos que se pueden reasignar). En estos casos $alloc(j)$ determina la enfermera asignada al trabajo fijado $j \in \mathcal{J}^{pre}$ y $cust(j)$ es el cliente asociado al trabajo no fijado $j \in \mathcal{J}^{\overline{pre}}$.
- $\mathcal{Q}$ es el conjunto ordenado de cualificaciones. Todas las enfermeras y los trabajos tienen asignada una cualificación, una enfermera puede realizar un trabajo sólo si su cualificación es igual o superior a la del trabajo.
- $\mathcal{F}$ determina el conjunto de causas de rechazo. $\mathcal{A}_c$ son los motivos por los cuales un cliente puede rechazar a una enfermera y $\mathcal{A}_n$ son los motivos por los cuales una enfermera puede rechazar a un cliente.
- $\mathcal{P}$ es el conjunto de los modos de transporte de las enfermeras y $p_n \in \mathcal{P}$ es el modo de trasporte preferido por la enfermera n. Las viviendas de la enfermera n y del paciente c se denotan por $loc(n)$ y $loc(c)$ respectivamente. El tiempo de transporte necesario para ir de $loc(a)$ a $loc(b)$ utilizando el medio de transporte p se almacena en la matriz $tt^p_{ab} \in \mathcal{T}$, donde $a, b \in \mathcal{C} \cup \mathcal{N}$ (haciendo $tt^p_{ab} \times \tau$ se obtienen minutos).
- La solución del problema es $\sigma = (\mathcal{R}, \mathcal{S})$, donde $\mathcal{R}$ es el conjunto de rutas y $\mathcal{S}$ es una aplicación que a cada trabajo le asigna su tiempo de inicio. A cada enfermera se le asigna una sola ruta, $R^n \in \mathcal{R}$, por tanto el número de rutas es igual al número de enfermeras ($|\mathcal{R}| = |\mathcal{N}|$) y cada ruta comienza y termina en la vivienda de la enfermera, $loc(n)$. R^n_k denota el k-ésimo trabajo de la ruta de la enfermera n y $t(R^n_k)$ define su tiempo de inicio. Si $R^n = \emptyset$ entonces la enfermera n no trabaja ese día.

Problema

Las restricciones que definen el problema, que se describen a continuación, se dividen en dos clases: restricciones fuertes (1-6), que deben respetarse siempre, y restricciones débiles (7-10), cuyo cumplimiento no es esencial para obtener soluciones factibles pero que es conveniente que este incumplimiento sea mínimo.

1. Todos los trabajos deben ser servidos por una enfermera

$$\exists n \in \mathcal{N} \; talque \; j \in R^n, \quad \forall j \in \mathcal{J}$$

2. Cada trabajo debe ser realizado una sola vez

$$R^n \cap R^m = \emptyset \quad \forall n, m \in \mathcal{N} \; con \; n \neq m$$

3. Las enfermeras deben estar cualificadas para el trabajo

$$\forall j \in \mathcal{J}, \; q_j \leq q_n \quad \forall n \in \mathcal{N}$$

4. Los tiempos de inicio de dos trabajos consecutivos tienen que ser tales que sea posible trasladarse de uno a otro

$$t(R_i^n) + dur_{R_i^n} + tt^{p_n}_{R_i^n R_{i+1}^n} \leq t(R_{i+1}^n) \quad \forall n \in \mathcal{N}, \; \forall i \; con \; 1 \leq i \leq |R^n| - 1$$

5. Una enfermera sólo trabaja durante su jornada

$$\forall n \in \mathcal{N}, \; \forall j \in \mathcal{R}^n, \; \exists tw \in TW_n \; talque \; \{t(j), ..., t(j) + dur_j\} \subseteq tw$$

6. Los trabajos fijados se asignan a la enfermera adecuada

$$\forall j \in \mathcal{J}^{\overline{pre}}, \; n = alloc(j), \; entonces \; j \in R^n$$

7. Las cualificaciones de las enfermeras y sus trabajos asignados deben ser iguales, esta restricción no es imprescindible (puesto que se tiene la 3), pero se añade ya que así se trata de no "desperdiciar" una enfermera asignándola a servicios de menor nivel que el suyo

$$q_j = q_n \quad \forall n \in \mathcal{N}, \; \forall j \in R^n$$

8. El tiempo de inicio de cada trabajo debe estar en la ventana de tiempo, esta restricción no es imprescindible (puesto que se tiene la 8), pero se añade para tratar de forzar que los trabajos comiencen en el horario más favorable del paciente

$$s_j \leq t(j) \leq e_j, \quad \forall j \in \mathcal{J}$$

9. El tiempo de inicio de cada trabajo debe ser el preferido por el paciente

$$t(j) = t_j, \quad \forall j \in \mathcal{J}$$

10. Los motivos de rechazo deben ser considerados

$$\mathcal{A}_n \cap \mathcal{A}_{cust(j)} = \emptyset \quad \forall n \in \mathcal{N}, \; \forall j \in R^n$$

Este problema se define como uno de minimización sin restricciones cuya función objetivo proporciona diferentes valores dependiendo de si la solución es factible o no, es decir, todas las restricciones forman parte de la función objetivo. De entre las soluciones válidas se desea elegir aquella que minimice los costes y maximice la satisfacción de los clientes y de las enfermeras.

La función objetivo asigna a cada solución un valor real, de modo que, si dicho valor se encuentra en el intervalo $[0, 1]$ la solución considerada es factible, en cambio, si toma valores mayores que 1 la solución no lo es. La función se divide en tres partes que se describen a continuación.

$$ob(\sigma) = \sum_{i=1}^{4} v_i + \sum_{i=5}^{8} v_i \gamma_i \phi_i + \sum_{i=9}^{11} v_i \gamma_i \phi_i$$

El primer sumando de la función hace referencia a las restricciones fuertes y las variables que intervienen en él son las siguientes.

Variable v_1 Contiene el número de asignaciones de trabajos inválidas, ya sea porque un trabajo no esté asignado, sea asignado más de una vez o por insuficiencia de la cualificación de la enfermera (restricciones 1-3).

Variable v_2 Numero de veces que se incumplen las restricciones de traslado, es decir, el número de veces que una enfermera no puede llegar al servicio a tiempo (restricción 4).

Variable v_3 Número de veces que se incumplen las disponibilidades de las enfermeras, es decir, número de trabajos que se le asignan a las enfermeras fuera de su jornada laboral (restricción 5).

Variable v_4 Número de veces que se incumple la restricción de trabajos fijados (restricción 6).

El segundo sumando de la función hace referencia a las restricciones débiles, y las variables que intervienen son las siguientes.

Variable v_5 La distancia entre la cualificación de una enfermera n y la requerida por el trabajo i que realiza (restricción 7).

Variable v_6 Cuantifica la desviación respecto a las ventanas de inicio $[s_j, e_j]$ para todos los trabajos (restricción 8).

Variable v_7 Cuantifica la desviación respecto al tiempo de inicio preferido por el usuario t_j para todos los trabajos (restricción 9).

Variable v_8 Cuantifica los incumplimientos de las preferencias de enfermeras y clientes (restricción 10).

El tercer sumando hace referencia a otras restricciones, como pueden ser el tiempo de trabajo o el tiempo de desplazamiento, y las variables son las siguientes.

Variable v_9 Cuantifica el trabajo realizado fuera del máximo diario.

Variable v_{10} Cuantifica el tiempo de trabajo de todas las enfermeras hasta el máximo diario.

Variable v_{11} Cuantifica el tiempo total de desplazamiento de las enfermeras.

Las constantes γ_i con $i = 5, ..., 11$ son tales que $\sum_{i=5}^{11} \gamma_i = 1$ y las constantes ϕ_i, $i = 5, ..., 11$, son factores de normalización, cuyo objetivo consiste en fijar que el mayor valor que la penalización por incumplimiento de las restricciones débiles (es decir, los sumandos en los que intervienen dichos parámetros) pueda tomar sea la unidad.

Técnicas de resolución

Para resolver este problema se estudian diferentes metodologías, que se describen brevemente a continuación.

Variable Neighbourhood Search. La *Variable Neighbourhood Search* (VNS), Mladenović & Hansen (1997), trata de mejorar la solución dada mediante una búsqueda en los vecindarios (o entornos) de ésta. Un vecindario es un conjunto de soluciones que se pueden conseguir realizando un único cambio (movimiento) a la solución considerada.

Determinar que solución del vecindario se selecciona depende de la estrategia que se esté utilizando, en este caso se consideran las siguientes:

- *Random*: uno de los vecinos se selecciona al azar.
- *Next-improvement*: se selecciona el primer vecino que mejora la solución actual.
- *Best-Improvement*: se selecciona el mejor vecino de todos los posibles.
- *Best-Of-Improvement*: se selecciona el mejor vecino de un subconjunto de ellos.

Los movimientos utilizados son:

- *Cambiar trabajo*: este movimiento cambia un trabajo de una ruta a otra distinta, teniendo en cuenta que, el trabajo se coloca en la mejor posición según la función objetivo.
- *Reposicionar trabajo*: un trabajo (asignado a una enfermera) se mueve a otra posición de la ruta sin reordenar los demás trabajos.
- *Intercambiar enfermeras*: dos enfermaras se intercambian las rutas entre ellas.

Simulated Annealing. El *Simulated Annealing* (SA), Kirkpatrick *et al.* (1983), es un método de búsqueda local que permite seleccionar soluciones que son peores que la solución dada, con cierta probabilidad.

En concreto, el método empleado para resolver el MHS se basa en procedimientos de búsqueda local simples que se crean combinando cada movimiento citado anteriormente con dos de las estrategias de mejora: *next-improvement* y *random*. Se usa la mejora aleatoria para seleccionar peores soluciones con el fin de escapar de un óptimo local, y se usa la *next-improvement* (en vez de best-improvement) ya que proporciona una mayor probabilidad de salir de un óptimo local, tras haber aceptado una peor solución en el mismo vecindario, y además emplea tiempos computacionales menores.

Memetic Algorithm. El *genetic algorithm* (GA), Holland (1975), considera una población inicial aleatoria, evalúa las soluciones iniciales y las evoluciona hasta cumplir la condición de finalidad (número de generaciones). El proceso de evolución comienza con un procedimiento de selección, donde un grupo de individuos se selecciona para tomar parte en el proceso de reproducción. Estos individuos se recombinan (combinación de las características de los padres) o se mutan (pequeñas alteraciones aleatorias), de modo que los nuevos individuos se evalúan y la generación original es reemplazada por los nuevos individuos según una estrategia de reemplazo (puede ser que individuos de la generación original se mantengan o no en la nueva generación).

El *memetic algorithm* (MA), Moscato *et al.* (1989), combina GA y una *búsqueda local*, que se utiliza antes de proceder con el reemplazo para tratar de mejorar aún más las soluciones obtenidas en el proceso de reproducción.

Scatter Search. La *scatter search* (SA), Glover (1998), consiste en generar soluciones no de forma aleatoria, sino teniendo en cuenta características de diferentes elementos de la solución. Para ello, se considera un conjunto de referencia, formado por soluciones buenas que se encuentran distribuidas por todo el espacio de soluciones. Dicho conjunto se genera según la función objetivo y la dispersión respecto a las demás soluciones (ya que se desean obtener soluciones buenas y diferentes entre sí). A partir de este conjunto se obtienen nuevas soluciones que, tras haber sido mejoradas, se añaden al conjunto de referencia y lo hacen evolucionar.

Comparación de los métodos

En el estudio del MHHS se comparan los métodos descritos anteriormente mediante la realización de experimentos, concluyendo que, el que mejor comportamiento presenta es el MA, seguido del VNS y el SS. Del mismo modo, también determina que el SA no proporciona soluciones tan buenas como los otos métodos considerados.

2.4.2. Home Care Crew Scheduling Problem

El *Home Care Crew Scheduling Problem* (HCCSP), Rasmussen *et al.* (2012), tiene como objetivo planificar las visitas del personal, de modo que se maximice el nivel del servicio y se minimicen los costes. Es decir, proporcionar las rutas que deben seguir las enfermeras tratando de dejar el menor número de servicios sin cumplir y minimizando los tiempos de desplazamiento. Es por esto que el HCCSP es una combinación del VRP y del NSP.

A continuación, se describe el modelo del problema, definiendo los conjuntos, parámetros, variables, restricciones y objetivos que lo determinan.

Conjuntos, parámetros y variables

Los conjuntos, parámetros y variables que intervienen en el problema se describen a continuación.

- $\mathcal{K} = \{1, ..., k\}$ representa el conjunto de enfermeras.
- $\mathcal{C} = \{1, ..., n-1\}$ es el conjunto de visitas a realizar (usuarios o trabajos); cada visita tiene una ventana de tiempo de inicio $[\alpha_i, \beta_i]$ que determina el tiempo más temprano y más tardío en que se puede empezar la visita i.
- $\mathcal{N}^k = \mathcal{C} \cup \{0^k, n^k\}$ determina el conjunto de visitas potenciales de k, donde 0^k y n^k son las visitas (ficticias) inicial y final, estas visitas ficticias tienen unas ventanas de tiempo $[\alpha_{0^k}, \beta_{0^k}] = [\alpha_{n^k}, \beta_{n^k}]$ que determinan la jornada laboral de k puesto que la primera visita no se puede realizar antes que α_{0^k} y la última visita no se puede realizar después de β_{n^k}.
- La distancia entre dos visitas i, j para la enfermera k es s_{ij}^k (depende de k por el medio de transporte que utilice). Se tiene que $s_{ij}^k = \infty$ si k no puede ir de i a j o si $i = j$. Estas distancias, s_{ij}^k, incluyen el tiempo de servicio de la visita i.
- El coste de que la enfermera k viaje de i a j queda determinado por c_{ij}^k.
- La variable binaria ρ_i^k es 1 si a k se le puede asignar la visita i y 0 en otro caso.
- El coste de que k haga la visita i es $\delta_i^k \in \mathbb{R}$. Si es negativo se quiere que k haga la visita i, mientras que si es positivo se prefiere que k no haga la visita i.
- La variable γ_i determina la prioridad de la visita i; cuanto mayor valor tome mayor será la prioridad.
- Las precedencias temporales (como por ejemplo la necesidad de sincronizar o solapar visitas, así como la máxima o mínima diferencia entre un par visitas) se modelan de la forma $\sigma_i + p_{ij} \leq \sigma_j$, donde σ_i es el tiempo de inicio de la visita i y p_{ij} es el margen requerido (es decir, el tiempo que debe pasar entre el inicio de la visita i y el de la visita j; por ejemplo, si las visitas deben sincronizarse este valor es 0, en cambio, para que haya solapamiento el margen puede tomar varios valores). $\mathcal{P}$ es el conjunto de parejas $(i, j) \in \mathcal{C} \times \mathcal{C}$ para las que existe restricción de precedencia temporales.
- x_{ij}^k es la variable binaria de ruta, que toma el valor 1 si k va de i a j ($i, j \in \mathcal{N}^k$) directamente y 0 en otro caso.
- $t_i^k \in \mathbb{R}^+$ es la variable que determina el tiempo en que k empieza la visita i. Si $t_i^k = 0$ entonces la enfermera k no se encarga de la visita i.
- y_i es la variable binaria de cobertura, que toma el valor 1 si ninguna enfermera realiza la visita i y 0 en otro caso.

Problema

Por último, a continuación, se presentan las restricciones del problema y la función objetivo a optimizar.

1. Cada visita se realiza una sola vez (o ninguna)

$$\sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{N}^k} x_{ij}^k + y_i = 1 \; \forall i \in \mathcal{C}$$

2. Las enfermeras sólo pueden realizar las visitas permitidas

$$\sum_{j \in \mathcal{N}^k} x_{ij}^k \leq \rho_i^k \ \forall i \in \mathcal{C}$$

3. Las enfermeras empiezan en la visita inicial

$$\sum_{j \in \mathcal{N}^k} x_{0^k j}^k = 1 \ \forall k \in \mathcal{K}$$

4. Las enfermeras terminan en la visita final

$$\sum_{i \in \mathcal{N}^k} x_{in^k}^k = 1 \ \forall k \in \mathcal{K}$$

5. Las rutas no se segmentan

$$\sum_{i \in \mathcal{N}^k} x_{ih}^k - \sum_{j \in \mathcal{N}^k} x_{hj}^k = 0 \ \forall k \in \mathcal{K} \ \forall h \in \mathcal{C}$$

6. Las ventanas de tiempo se deben respetar

$$\alpha_i \sum_{j \in \mathcal{N}^k} x_{ij}^k \leq t_i^k \leq \beta_i \sum_{j \in \mathcal{N}^k} x_{ij}^k \ \forall k \in \mathcal{K} \ \forall i \in \mathcal{C} \cup \{0^k\}$$

7. Las ventanas de tiempo de las enfermeras deben respetarse

$$\alpha_{n^k} \leq t_{n^k}^k \leq \beta_{n^k} \ \forall k \in \mathcal{K}$$

8. Las distancias de desplazamiento deben respetarse

$$t_i^k + s_{ij}^k x_{ij}^k \leq t_j^k + \beta_i (1 - x_{ij}^k) \ \forall k \in \mathcal{K} \ \forall i, j \in \mathcal{N}^k$$

9. Las restricciones de precedencia deben respetarse

$$\alpha_i y_i + \sum_{k \in \mathcal{K}} t_i^k + p_{ij} \leq \sum_{k \in \mathcal{K}} t_j^k + \beta_j y_j \ \forall (i,j) \in \mathcal{P}$$

   La suma $\sum_{k \in \mathcal{K}} t_i^k$ determina el tiempo de inicio de la visita i, puesto que el único elemento distinto de cero de la suma es instante de inicio de la visita, y las variables y_i, y_j garantizan el cumplimiento de estas restricciones en caso de que alguna de las visitas no se realice.

La función objetivo de este problema de minimización es la siguiente:

$$w_1 \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{N}^k} \sum_{j \in \mathcal{N}^k} c_{ij}^k x_{ij}^k + w_2 \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{C}} \sum_{j \in \mathcal{N}^k} \delta_i^k x_{ij}^k + w_3 \sum_{i \in \mathcal{C}} \gamma_i y_i$$

donde el primer sumando hace referencia a los costes de trasladarse de una visita a otra, el segundo sumando a las preferencias y el tercer sumando a las visitas sin asignar.

En general, la minimización de visitas no asignadas tiene prioridad con respecto a las preferencias y éstas, a su vez, tienen prioridad respecto a la minimización de los costes. Para ello se fijan los coeficientes w de la siguiente forma: $w_1 = 1$, $w_2 = \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{N}^k} \sum_{j \in \mathcal{N}^k} c_{ij}^k$ y $w_3 = w_2 |\mathcal{C}| \operatorname{máx}_{k \in \mathcal{K}, i \in \mathcal{C}} \delta_i^k$

Resolución

Ramussen et al. resuelven este problema empelando el método de *Branch and Price* (BP), Dantzig & Wolfe (1960), que es una combinación de los métodos *branch and bound* y generación de columnas. Este método consiste en aplicar un *branch and bound* en el que en cada vértice del árbol se pueden añadir columnas a la relajación del problema de programación lineal, es decir, al principio del método se excluye un conjunto de columnas de la relajación y estas pueden ir añadiéndose cuando sea necesario.

2.4.3. Otros problemas

Durante la realización de este trabajo también se han estudiado otros problemas como el *Home Health Care Problem*, Nickel *et al.* (2012), que consiste en abordar el problema de atención a domicilio en dos fases. O el *Care-at-Home Service Problem*, Koeleman *et al.* (2012), que estudia el caso en que se desee determinar qué nuevos usuarios deben añadirse al servicio de atención a domicilio considerado.

Estos problemas no se han añadido a este trabajo puesto que utilizan enfoques muy alejados al del problema de replanificación general que presenta Mayores.

2.5. Comparación con el problema en estudio

Para concluir este capítulo, es necesario especificar aquellos aspectos de los problemas estudiados que pueden utilizarse durante la modelización del problema de replanificación general en estudio.

En primer lugar, cabe destacar que el problema de Mayores tiene como base un VRPTW, puesto que las auxiliares deben desplazarse a las viviendas de los usuarios para realizar los servicios (realización de rutas, VRP) y que las visitas deben realizarse respetando los horarios disponibles de las mismas (ventanas de tiempo, TW).

La restricción de visitar a cada uno de los clientes es equivalente a exigir, en el problema de Mayores, que todos los servicios sean realizados por una única auxiliar. Del mismo modo, respetar las ventanas de los clientes es homólogo a respetar las de los servicios y, por último, no superar las capacidades máximas de los vehículos se corresponde con exigir que se respeten las jornadas de las auxiliares.

Pero, aun así, el problema de rutas presenta una característica que no se adapta al caso en estudio, y ésta es la necesidad de comenzar y finalizar las rutas en el almacén. En el problema de Mayores las jornadas de las auxiliares comienzan al realizar el primer servicio y finalizan al ejecutar el último, de modo que, los desplazamientos hasta el servicio inicial, y desde el servicio final, no se consideran tiempo trabajado. Por tanto, las rutas que recorren las auxiliares no comienzan y finalizan en el mismo punto.

Por otro lado, considerando los problemas de atención a domicilio (MHHS, HCCSP), debido a que son una combinación del VRP y del NSP, muchas de sus características se adaptan el problema de Mayores, como pueden ser que todos los trabajos deben realizarse dentro de sus ventanas horarias, que las planificaciones deben respetar los tiempos de desplazamiento entre visitas, o que las distancias de desplazamiento deben respetarse.

Ahora bien, algunas de las propiedades de estos problemas no se adecuan directamente al problema en estudio y será necesario modificarlas. Por ejemplo, las restricciones de cualificación y rechazo del MHS deben de cambiarse por unas en las que intervenga el nivel de afinidad entre la auxiliar y el usuario considerado.

No obstante, se tienen restricciones que no pueden ser aplicadas al problema de mayores, como, por ejemplo, las relativas a trabajos fijados del MHHS o las restricciones de precedencia del HCCSP.

Como conclusión, cabe destacar que, pesar de que estos problemas tengan una naturaleza similar al caso en estudio, el problema de Mayores presenta destacables novedades que deben modelarse de forma matemática para tener el problema correctamente definido. Estas nuevas características hacen

referencia a las jornadas trabajadas de las auxiliares, puesto que, todos los descansos forman parte de la jornada trabajada a excepción del mayor de ellos (cuya duración supera las 2 horas).

Capítulo 3

Modelado matemático del problema

A la vista de lo que se ha explicado con detalle en capítulos anteriores, el problema que presenta la empresa Mayores es muy complejo y muestra peculiaridades que no habían sido estudiadas en la literatura. Es por ello que se ha llevado a cabo la formulación del problema que se presenta en este trabajo como uno de programación lineal entera.

En este capítulo, se describirá la formulación del problema de planificación general de Mayores. En primer lugar se definen los conjuntos, variables y parámetros que intervienen en el problema, seguidamente se describen las restricciones que lo determinan y, por último, se establece la función objetivo a optimizar durante la resolución del mismo.

3.1. Variables, conjuntos y parámetros que intervienen en el problema

A continuación, se realizará la descripción de los conjuntos, parámetros y variables necesarios para la modelización del problema.

3.1.1. Conjuntos

Los conjuntos empleados en el modelado del problema de planificación general de Mayores son los siguientes:

Días El horizonte temporal considerado en este problema es una semana, que se denota como el conjunto de sus *días*, $Q = \{1, ..., 7\}$.

Auxiliares El conjunto total de *auxiliares* en el sistema es $N = \{1, ..., n\}$.

División del horizonte temporal Para poder tratar este problema como uno de programación entera es necesario discretizar el horizonte temporal, de modo que la semana se divide en *períodos* de 5 minutos de duración, quedando así definido el conjunto de los instantes de tiempo como $M = \{1, ..., m\}$, siendo $m \in \mathbb{N}$ el número máximo de períodos (es decir, los períodos de 5 minutos que tiene una semana).

Últimos períodos de cada día Para poder trabajar con los diferentes días de la semana será necesario definir el conjunto que contiene el *último período de cada día* de la forma $\bar{M} = \{m_0, m_1, ..., m_7\}$, donde $m_1, ..., m_7 \in M$ son los últimos períodos de los días de la semana, $m_0 = 0$ y $m_7 = m$ el último período del domingo (es decir, último período de la semana).

Servicios Los *servicios* a realizar se recogen en el conjunto $S = \{1, ..., s\}$. Para poder formular el problema correctamente es necesario definir dos servicios ficticios: el *servicio ficticio inicial*,

denotado 0, que representa el comienzo de la jornada de una auxiliar y el *servicio ficticio final*, denotado por $s+1 \in \mathbb{N}$, que representa el final de la jornada de una auxiliar.

De modo que, a partir de estos servicios se definen los siguientes conjuntos:

- El conjunto de todos los servicios más el ficticio inicial se denota de la forma $S^0 = S \cup \{0\}$.
- El conjunto de todos los servicios más el ficticio final se denota de la forma $S^1 = S \cup \{s+1\}$.
- El conjunto de todos los servicios más los ficticios inicial y final se denota de la forma $S^{01} = S \cup \{0\} \cup \{s+1\}$.

3.1.2. Parámetros

Los parámetros empleados en el modelado del problema de planificación general de Mayores son los siguientes:

Duración y ventanas de tiempo La *duración* y las *ventanas de tiempo* de los servicios se almacenan en una matriz $A = (a_{\alpha k})_{5\times s}$, donde cada columna representa un servicio $k \in S$ y cada fila almacena una característica de dicho servicio:

1. En la primera fila se tiene la duración de los servicios en minutos, $a_{1k} \in \mathbb{N}$.
2. En la segunda fila se tiene el inicio de la ventana de tiempo disponible para los servicios, $a_{2k} \in M$.
3. En la tercera fila se tiene el instante final de la ventana de tiempo disponible, $a_{3k} \in M$.
4. En la cuarta fila se tiene el inicio de la ventana de tiempo óptima, $a_{4k} \in M$.
5. En la quinta fila se tiene el final de la ventana de tiempo óptima, $a_{5k} \in M$.

Además es necesario que se cumplan las siguientes desigualdades:

- Las ventanas de tiempo disponible y óptima deben estar bien definidas, es decir, $a_{2k} < a_{3k}$ y $a_{4k} < a_{5k}$.
- El servicio puede realizarse dentro de la ventana óptima, es decir, $a_{5k} - a_{4k} \geq a_{1k} - 1$
- La ventana óptima está contenida en la ventana factible, es decir, $a_{5k} \leq a_{3k}$ *y* $a_{2k} \leq a_{4k}$

Afinidad El *nivel de afinidad* entre una auxiliar y el usuario que requiere el servicio se denota como $f_{ik} \in \{0,1,2,3,4,5\}\ \forall i \in N,\ \forall k \in S$.

Desplazamientos La matriz entera que recoge los *tiempos de desplazamiento* (en minutos) entre las tareas se denota como $T = (t_{kl})_{k\times l}$, donde t_{kl} es el tiempo entre las tareas $k \in S$ y $l \in S$, es decir, el tiempo de desplazamiento entre el usuario que requiere la tarea k y el usuario que requiere la tarea l. Los tiempos de desplazamiento en los que intervenga algún servicio ficticio son siempre nulos, puesto que estos servicios no existen y solamente se utilizan para la formulación del problema.

Jornada laboral El parámetro hs_i, con $i \in N$, denota las *horas semanales* que tiene contratada la auxiliar y, en caso de que sea necesario, el parámetro hd_{iq}, con $i \in N$ y $q \in Q$, denota las *horas diarias* de trabajo que están estipuladas en el contrato de la auxiliar.

Bolsa de horas El parámetro b_i, con $i \in N$, determina el número de horas acumuladas en la *bolsa de horas* de la auxiliar i; este valor es negativo si la auxiliar ha trabajado menos horas de las contratadas y es positivo en caso contrario.

3.1.3. Variables

Las variables consideradas en el modelado del problema de planificación general de Mayores son las siguientes:

Variable La variable binaria que será la base de la formulación del problema de programación se define de la siguiente forma:

$$x_{ijkl} = \begin{cases} 1 & \textit{si la auxiliar } i \textit{ empieza el servicio } k \textit{ en el período } j \textit{ y después} \\ & \textit{se traslada al servicio } l \\ 0 & \textit{en otro caso} \end{cases}$$

Con $i \in N$, $j \in M$, $k \in S^0$ y $l \in S^1$.

Descansos La variable entera d_{iq} determina el mayor descanso, cuya duración es igual o superior a las dos horas, que tiene la auxiliar $i \in N$ en el día $q \in Q$.

3.2. Formulación del problema de programación lineal

A continuación, se presenta el problema en estudio como uno de programación lineal entera. Para ello, en primer lugar, se explican las restricciones que definen el problema y, seguidamente, se detalla la función objetivo que se quiere optimizar.

3.2.1. Restricciones que determinan el problema

Las condiciones que debe cumplir una solución del problema, para que se adecúe a las demandas de la empresa Mayores, se enuncian mediante las restricciones expuestas a continuación.

Cada uno de los servicios debe ser llevado a cabo por una única auxiliar y, además, debe realizarse dentro de su ventana de tiempo disponible. Es decir, los servicios deben comenzar en algún instante de tiempo comprendido entre el inicio de la ventana factible $a_{2k} \in M$ y el instante $a_{3k} - a_{1k} \in M$, y esto solamente puede suceder para una auxiliar $i \in N$ y para uno de los posibles servicios siguientes $l \in S^1 \setminus \{k\}$.

$$\sum_{i \in N} \sum_{j \in [a_{2k}, a_{3k} - a_{1k}]} \sum_{l \in S^1 \setminus \{k\}} x_{ijkl} = 1 \quad \forall k \in S$$

Cada uno de los servicios sólo puede tener un servicio anterior. Es decir, se exige que solamente una auxiliar $i \in N$,pueda desplazarse desde un único servicio $k \in S^0$ hasta el servicio dado, $l \in S$, en un instante de tiempo, $j \in M$.

$$\sum_{i \in N} \sum_{j \in M} \sum_{k \in S^0 \setminus \{l\}} x_{ijkl} = 1 \quad \forall l \in S$$

Cada auxiliar debe comenzar su jornada diaria en el servicio ficticio inicial. De modo que, para cada día $q \in Q$ se necesita que la auxiliar $i \in N$ se desplace del servicio ficticio inicial 0 a algún otro servicio, $l \in S^1$, de los que pueden ser realizados en dicho día.

$$\sum_{j \in [m_{q-1}+1, m_q]} \sum_{\substack{l \in S^1 / \\ a_{2l} \in [m_{q-1}+1, m_q]}} x_{ij0l} = 1 \quad \forall i \in N,\ \forall q \in Q$$

Cada auxiliar debe finalizar su ruta diaria en el servicio ficticio final, es decir, para cada día $q \in Q$ se necesita que la auxiliar $i \in N$ se desplace desde algún servicio, $k \in S^0$, de los que se pueden realizar en dicho día, al servicio ficticio final $s+1$.

$$\sum_{j \in [m_{q-1}+1, m_q]} \sum_{\substack{k \in S^0 / \\ a_{2k} \in [m_{q-1}+1, m_q]}} x_{ijks+1} = 1 \quad \forall i \in N,\ \forall q \in Q$$

Los tiempos de inicio de dos trabajos consecutivos tienen que ser tales que sea posible trasladarse de uno a otro, es decir, deben respetarse las duraciones de los servicios y los tiempos de desplazamiento entre ellos. Es necesario diferenciar tres casos:

- Si los servicios son del mismo día ($k, l \in S$ tal que $a_{2k}, a_{2l} \in [m_{q-1}+1, m_q]$ para algún día $q \in Q$), el servicio l debe comenzar en un instante posterior al que se obtiene tras sumar al instante de inicio del servicio k, la duración de dicho servicio y el tiempo de desplazamiento entre ambos servicios, $v = j + a_{1k} + t_{kl}$.

$$x_{ijkl} \leq \sum_{v=j+a_{1k}+t_{kl}}^{m} \sum_{\substack{r \in S^1 / \\ r \neq k}} x_{ivlr} \quad \forall i \in N,\ \forall j \in M,\ \forall k \in S,\ \forall l \in S$$

$$x_{ijlk} = 0,\ i, j, k, l \ tal \ que \ j + a_{1k} + t_{kl} > m$$

- Si los servicios no son del mismo día ($k, l \in S$ tal que $a_{2k}, \in [m_{q-1}+1, m_q]$ y $a_{2l} \in [m_{p-1}+1, m_p]$ donde $q \neq p$, $q, p \in Q$), será imposible que una auxiliar realice el servicio k y seguidamente se desplace hasta el servicio l, por tanto, dichas variables deben fijarse a 0.

$$x_{ijlk} = 0$$

- Si la primera tarea es la ficticia 0, no hay desplazamiento, pero es necesario asegurarse de que se continúa correctamente. De modo que, si una auxiliar $i \in N$ realiza la tarea inicial 0 en un instante de tiempo $j \in [m_{q-1}+1, m_q]$, de un día $q \in Q$, y después se traslada a otro servicio $l \in S$, es necesario que dicha auxiliar comience la tarea l en dicho instante de tiempo j y que, seguidamente, se traslade a alguna otra tarea $r \in S^1$.

$$x_{ij0l} \leq \sum_{p \in S^1} x_{ijlp} \quad \forall i \in N,\ \forall j \in [m_{q-1}+1, m_q],\ \forall q \in Q,\ \forall l \in S$$

No asignar una auxiliar a un usuario cuando la afinidad entre ellos es nula (asignación vetada).

$$x_{ijkl} \leq f_{ik} \quad \forall j \in M,\ \forall l \in S^1,\ \forall i \in N,\ \forall k \in S$$

Cada auxiliar no puede trabajar más tiempo del que marca su jornada diaria. Es decir, la diferencia entre el instante en que la auxiliar $i \in N$ finaliza la jornada $q \in Q$ (instante en que se comienza el último servicio, $k \in S^0$, más la duración del mismo, a_{1k}) y el instante en que comienza la jornada (instante en que se comienza el primer servicio, $l \in S^1$), sin contar el mayor de los descansos de duración igual

o superior a las dos horas (d_{iq}) que realiza la auxiliar en ese día, no puede superar la jornada diaria máxima de dicha auxiliar (hd_{iq}).

$$\sum_{j\in[m_{q-1}+1,m_q]}\sum_{k\in S^0}(j+a_{1k})\times x_{ijks+1}-\sum_{j\in[m_{q-1}+1,m_q]}\sum_{l\in S^1}j\times x_{ij0l}-d_{iq}\leq hd_{iq}$$

$$\forall q\in Q,\ \forall i\in N$$

Cada auxiliar $i\in N$ no puede trabajar, semanalmente, más tiempo del que determina su jornada máxima semanal hs_i.

$$\sum_{q\in Q}\left(\sum_{j\in[m_{q-1}+1,m_q]}\sum_{k\in S^0}(j+a_{1k})\times x_{ijks+1}-\sum_{j\in[m_{q-1}+1,m_q]}\sum_{l\in S^1}j\times x_{ij0l}-d_{iq}\right)\leq hs_i$$

$$\forall i\in N$$

El cálculo del mayor de los descansos de cada jornada, μ_{iq}, se obtiene añadiendo al problema las cinco restricciones que se presentan a continuación.
En primer lugar, se tienen las dos siguientes restricciones, que garantizan que el mayor descanso que toma la auxiliar $i\in N$ en el día $q\in Q$ puede tener duración nula.

$$\mu_{iq}\geq 0,\ \forall i\in N,\ \forall q\in Q$$

$$\mu_{iq}\leq 0+C(1-y_{iq}),\ \forall i\in N,\ \forall q\in Q$$

siendo $C\in\mathbb{N}$ una constante grande.

Las dos restricciones que se presentan a continuación se utilizan para asegurar que en el cálculo del mayor descanso, que tiene la auxiliar $i\in N$ en su planificación para el día $q\in Q$, se tienen en cuenta todos los posibles descansos que se pueden generar al obtener su planificación.

$$\mu_{iq}+D(1-\sum_{j\in[m_{q-1}+1,m_q]}x_{ijkl})\geq\left(\sum_{j\in[m_{q-1}+1,m_q]}j\times x_{ijlp}-\sum_{j\in[m_{q-1}+1,m_q]}(j+t_{kl}+a_{1k})\times x_{ijkl}\right)$$

$$\forall k,l\in S,\ \forall p\in S^1,\ k\neq l\neq p,\ \forall i\in N,\ \forall q\in Q$$

$$\mu_{iq}+D(1-\sum_{j\in[m_{q-1}+1,m_q]}x_{ijkl})\leq\left(\sum_{j\in[m_{q-1}+1,m_q]}j\times x_{ijlp}-\sum_{j\in[m_{q-1}+1,m_q]}(j+t_{kl}+a_{1k})\times x_{ijkl}\right)$$
$$+C(1-y_{iqklp})$$

$$\forall k,l\in S,\ \forall p\in S^1,\ k\neq l\neq p,\ \forall i\in N,\ \forall q\in Q$$

siendo $D\in\mathbb{N}$ una constante grande.

Esta última restricción sirve para asegurar que el mayor descanso que presenta la planificación de la jornada $q\in Q$ de la auxiliar $i\in N$ toma, o bien el valor 0, o bien el mismo valor que el mayor de los posibles descansos que se generan en dicha jornada.

$$\sum_{k\in S}\sum_{l\in S}\sum_{p\in S^1}y_{iqklp}+y_{iq}=1$$

$$y_{iqklp},\ y_{iq}\in\{0,1\}\quad\forall k,l\in S\ \forall p\in S^1,\ \forall i\in N,\forall q\in Q$$

siendo k, l, p servicios distintos que se realizan en el día q ($a_{2k}, a_{2l}, a_{2p} \in [m_{q-1}+1, m_q]$) y, por último, y_{iqklp} e y_{iq} son variables binarias auxiliares que se utilizan para determinar que el mayor de los descansos debe de ser alguno de los calculados.

Los mayores descansos de cada día deben tener una duración igual o superior a las 2 horas para no formar parte de la jornada trabajada de las auxiliares. De modo que es necesario definir una variable binaria auxiliar, z_{iq}, y considerar las restricciones que se presentan a continuación.
Esta primera restricción garantiza que la variable z_{iq} tomará el valor 1 si μ_{iq} tiene una duración igual o superior a las dos horas, y 0 en caso contrario.

$$\frac{\frac{\mu_{iq}}{2} - 1}{E} < z_{iq} \leq \frac{\mu_{iq}}{2} \quad \forall i \in N,\ \forall q \in Q$$

Las siguientes cuatro restricciones sirven para garantizar que el descanso $d_{iq} \in \mathbb{E}$ será nulo si $z_{iq} = 0$ y que $d_{iq} = \mu_{iq}$ en caso contrario.

$$d_{iq} \leq E z_{iq} \quad \forall i \in N,\ \forall q \in Q$$

$$d_{iq} \leq \mu_{iq} \quad \forall i \in N,\ \forall q \in Q$$

$$d_{iq} \geq \mu_{iq} - (1 - z_{iq})E \quad \forall i \in N,\ \forall q \in Q$$

$$d_{iq} \geq 0 \quad \forall i \in N,\ \forall q \in Q$$

siendo $E \in \mathbb{N}$ una constante grande.

3.2.2. Función objetivo a optimizar

Los objetivos que se desean optimizar, ya explicados con detalle en capítulos anteriores, son los siguientes.

Los descansos que realizan las auxiliares durante su jornada diaria, a excepción del mayor de ellos si tiene duración igual o superior a las dos horas, producen pérdidas para la empresa y, por consiguiente, es necesario que a la hora de obtener la planificación óptima éstos sean lo menores posible. Del mismo modo, el tiempo que emplean las auxiliares desplazándose entre las viviendas de los usuarios a los que deben atender también produce pérdidas a Mayores y, por tanto, deben minimizarse. En base a esto, se requiere minimizar la jornada semanal total de trabajo de las auxiliares.

$$\sum_{i \in N} \sum_{q \in Q} \left(\sum_{j \in [m_{q-1}+1, m_q]} \sum_{k \in S^0} (j + a_{1k}) \times x_{ijks+1} - \sum_{j \in [m_{q-1}+1, m_q]} \sum_{l \in S^1} j \times x_{ij0l} - d_{iq} \right)$$

La empresa considera muy importante que los usuarios estén satisfechos con las auxiliares que los atienden y, por tanto, es necesario que la planificación óptima proporcione las asignaciones de auxiliares a usuarios que mayor afinidad generen. Es decir, se requiere maximizar la afinidad de las asignaciones que se realizan.

$$\sum_{i \in N} \sum_{j \in M} \sum_{k \in S} \sum_{l \in S^1} f_{ik} x_{ijkl}$$

En capítulos anteriores, se explica que los usuarios, aparte de tener un horario disponible, también cuentan con un horario óptimo en que prefieren ser atendidos. Debido a que respetar estas ventanas de

tiempo es opcional, solamente se considera en la función objetivo de forma que, se trata de maximizar el número de veces que los servicios se realizan dentro del horario óptimo.

$$\sum_{i\in N}\sum_{k\in S}\sum_{j\in[a_{4k},a_{5k}-a_{1k}]}\sum_{l\in S^1} x_{ijkl}$$

En consecuencia, mediante una ponderación de los tres objetivos explicados anteriormente, y realizando un cambio de signo en aquellos que requieren ser maximizados, la función objetivo a minimizar es la siguiente:

$$\alpha\sum_{i\in N}\sum_{q\in Q}\left(\sum_{j\in[m_{q-1}+1,m_q]}\sum_{k\in S^0}(j+a_{1k})\times x_{ijks+1}-\sum_{j\in[m_{q-1}+1,m_q]}\sum_{l\in S^1} j\times x_{ij0l}-d_{iq}\right)$$

$$-\beta\sum_{i\in N}\sum_{j\in M}\sum_{k\in S}\sum_{k\in S} f_{ik}x_{ijkl}-\gamma\sum_{i\in N}\sum_{k\in S}\sum_{j\in[a_{4k},a_{5k}-a_{1k}+1]}\sum_{l\in S^1} x_{ijkl}$$

con α, β y γ constantes reales utilizadas para realizar la ponderación.

3.2.3. Conclusiones

El problema que se ha modelado, para casos reales, como los que afronta Mayores de forma diaria, presenta un gran número de restricciones y variables, siendo algunas de ellas enteras. Por lo cual, se hace inabarcable obtener soluciones exactas en las que se obtengan planificaciones aplicables a problemas reales. Debido a esto, en el siguiente capítulo se introduce un algoritmo, basado en técnicas metaheurísticas, para resolver problemas reales de forma inmediata.

Modelar el problema como uno de programación lineal es interesante, ya que, dicho modelo puede utilizarse para comparar, mediante la resolución de ejemplos pequeños, el comportamiento del algoritmo diseñado con respecto a las soluciones exactas del problema.

Capítulo 4

Aproximación metaheurística

El problema que presenta Mayores es de gran complejidad, tal y como se muestra en capítulos anteriores. En consecuencia, será necesario abordarlo empleando técnicas heurísticas y/o metaheurísticas, con el objetivo de obtener soluciones aceptables en tiempos computacionales pequeños.

En este capítulo, se describe el algoritmo diseñado para resolver el problema en estudio. Dicho algoritmo está basado en el método metaheurístico *simulated annealing* y tiene como objetivo proporcionar planificaciones que respeten las restricciones impuestas por la empresa.

Este algoritmo se ha implementado dentro de la aplicación informática diseñada para Mayores, de tal forma que, no se trata de resolver un problema desde cero (como el que se modela en el capítulo 3), sino que se se quieren resolver las incidencias descritas en 1.3.2, aplicando las menores modificaciones posibles a la planificación considerada.

Para comenzar, se presenta la definición del método *simulated annealing* y, seguidamente se describe la metodología empleada a la hora de diseñar el algoritmo. Tras ello, se presenta la explicación del algoritmo desarrollado, detallando las diferentes fases en las que se divide el método y sus características más importantes. Por último, se comentan brevemente otras versiones del método que se han considerado.

4.1. Simulated annealing

El *simulated annealing* (recocido simulado, templado simulado) es un método de óptimización metaheuristico inspirado en el proceso de templado de metales. Este proceso consiste en calentar el metal hasta que alcance determinada temperatura, mantener el material a dicha temperatura y por último, enfriarlo lentamente para que obtenga determinadas propiedades.

4.1.1. Definición

El método del simulated annealing, Kirkpatrick *et al.* (1983), es una variante del algoritmo de Metropolis, Metropolis *et al.* (1953), y se basa en realizar una búsqueda por entornos con un criterio probabilístico de aceptación de soluciones. El algoritmo puede dividirse en dos fases: la primera consiste en una generación de soluciones y la segunda se basa en la aceptación de las mismas. Esta fase de aceptación consta de una componente aleatoria que permite seleccionar soluciones "peores" que la solución actual; esto se realiza con el objetivo de escapar de óptimos locales y poder alcanzar el óptimo global.

4.1.2. Parámetros del algoritmo

Los siguientes elementos deben especificarse para cada problema particular al que quiera aplicarse el método del *simulated annealing*.

Estado de energía Con el objetivo de poder comparar soluciones, es necesario definir el estado de energía asociado a una solución $E(s)$, es decir, la función objetivo a minimizar.

Generador de vecino Esta función es la que se utiliza para obtener los vecinos de la solución que está siendo considerada en cada iteración. La función genera vecinos mediante la aplicación de ligeras modificaciones, denominadas movimientos, a la solución base.

Probabilidad de transición La probabilidad de seleccionar como nueva solución un vecino "peor" que aquella que está siendo considerada se obtiene a partir de la temperatura T y de la diferencia entre los estados de energía (función objetivo) de las soluciones consideradas (el vecino y la actual). La formulación descrita por Kirkpatrick *et al.* (1983) es

$$P(s, s', T) = e^{-(E(s')-E(s))/T}$$

siendo s y s' las soluciones consideradas y T la temperatura actual. A parte de esta, existen otras muchas propuestas de funciones para calcular dicha probabilidad.

Temperatura inicial Esta temperatura es la que se considera en la primera iteración del algoritmo.

Enfriamiento de la temperatura Para que la probabilidad de transición disminuya según avanzan las iteraciones es necesario definir la función que determina cómo disminuye la temperatura considerada. La formulación descrita por Van Laarhoven & Aarts (1987) proporciona la temperatura, para cada iteración k, a partir de la inicial T_0 de la forma siguiente:

$$\begin{aligned} T_1 &= T_0 \cdot \beta \\ T_2 &= T_1 \cdot \beta = T_0 \cdot \beta^2 \\ &\ldots \\ T_{k+1} &= T_k \cdot \beta = T_0 \cdot \beta^{k+1} \end{aligned}$$

donde $\beta \in [0{,}8, 0{,}99]$, Henderson *et al.* (2003), Johnson *et al.* (1991), es el parámetro de enfriamiento.

4.1.3. Pseudocódigo del algortítmo

El pseudocódigo del algoritmo *simulated annealing* se presenta a continuación.

Algoritmo 4.1: Simulated annealing

```
1  s ← solucionInicial()
2  T ← temperaturaInicial()
3  while no se satisfagan condiciones de finalización do
4  |  s' ← vecinoAleatorio(s)
5  |  f(s) ← funciónObjetivo(s)
6  |  f(s') ← funciónObjetivo(s')
7  |  if f(s') <f(s) then
8  |  |  s ← s'
9  |  else
10 |  |  x ← númeroAleatorio(0,1)
11 |  |  if x <probabilidadAceptación(s,s') then
12 |  |  |  s ← s'
13 |  |  end
14 |  end
15 end
16 return s
```

4.2. Metodología empleada durante el desarrollo del algoritmo

La metodología seguida, para obtener el algoritmo metaheurístico más adecuado, para resolver el problema que presenta Mayores, consta de la siguientes fases:

Comparación de métodos mediante experimentación. En esta fase se comparan los métodos propuestos (*simulated annealing* y búsqueda local) mediante experimentación. Las pruebas realizadas llevan a determinar que el algoritmo a utilizar es el *simulated annealing*, puesto que emplea menor tiempo computacional en proporcionar buenas soluciones.

Optimización del algoritmo mediante experimentación y refinamiento. Habiendo decidido el método más adecuado para resolver el problema de Mayores, se lleva a cabo la segunda fase del proceso con el objetivo de conseguir mejorar el algoritmo. Para ello, se aplican varios refinamientos al método siguiendo el diagrama de flujo que se muestra en la figura siguiente.

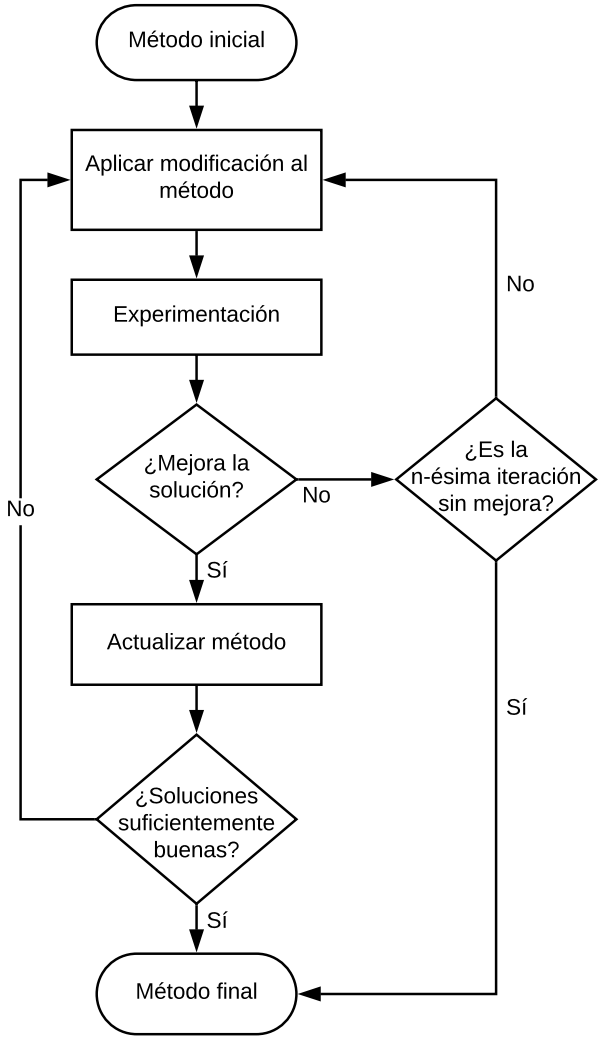


Figura 4.1: Diagrama de flujo de la metodología seguida

Partiendo del método seleccionado en la fase anterior, para cada problema a solucionar, se efectúa el refinamiento siguiendo la metodología siguiente.

Aplicar modificación al método. En esta fase de modificación se aplican cambios a la versión más reciente del método con el objetivo de obtener uno que proporcione mejores soluciones. Las nuevas variantes se pueden obtener, por ejemplo, cambiando las condiciones de parada del algoritmo, modificando las funciones utilizadas para la generación de vecindarios o alterando el tamaño de los mismos.

Experimentación. La fase de experimentación consiste en utilizar la versión modificada del método, obtenida en la etapa anterior, para resolver un conjunto de ejemplos. El objetivo de esta etapa es estudiar los resultados obtenidos durante la experimentación y comprobar si la nueva versión del algoritmo puede considerarse "mejor" que la anterior. Para ello se analizan, tanto las soluciones que proponen los algoritmos, como el tiempo que emplean en hacerlo. De modo que, en primer lugar, se considera "mejor" a aquel algoritmo que proporciones mejores soluciones y, en caso de que ambos métodos devuelvan soluciones equivalentes, se toma como "mejor" el algoritmo que proporcione las soluciones empleando menores tiempos de computación.

Actualizar método. La etapa de actualización solamente es llevada a cabo en aquellos casos en los cuales se puede observar, durante la experimentación, que las modificaciones aplicadas al método proporcionan mejoras respecto su versión previa. Por tanto, en caso de ser necesaria la actualización, definimos la nueva versión actualizada del método como la versión modificada considerada en la primera etapa.

4.3. Algoritmo definitivo

El algoritmo desarrollado para resolver el problema de planificación general de Mayores consiste en replanificar los servicios y optimizar las jornadas de las auxiliares con el objetivo de que los planes sean factibles, que las auxiliares no tengan huecos en sus jornadas y que todos los servicios estén correctamente planificados.

El algoritmo, detallado en el anexo A, que se ha diseñado puede descomponerse en las fases que se describen a continuación.

4.3.1. Fase inicial

Esta primera fase se basa en analizar los datos de entrada, de modo que se determine el conjunto de servicios a replanificar y, en caso de que dicho conjunto sea vacío, que se especifique si es necesario optimizar la planificación de las auxiliares con las que se esté trabajando.

Datos de entrada

Los datos de entrada con los que trabaja el algoritmo se definen a continuación.

- Coordinadora. Trabajadora que está realizando la planificación desde la aplicación informática.
- Usuarios seleccionados. Usuarios que la coordinadora ha seleccionado, es decir, usuarios que se consideran para la replanificación automática.
- Auxiliares seleccionadas. Lista de auxiliares que la coordinadora ha seleccionado, es decir, las auxiliares que se contemplan para atender a los usuarios seleccionados.

Auxiliares consideradas

El primer paso de esta fase inicial consiste en determinar el conjunto de auxiliares con las que debe trabajar el algoritmo. Debido a que es posible que la coordinadora no seleccione ninguna auxiliar se consideran los dos casos siguientes:

1. Si la coordinadora ha seleccionado auxiliares, es decir, si el conjunto de auxiliares seleccionadas no está vacío, el algoritmo trabaja con dicho conjunto.
2. Si la coordinadora no ha seleccionado auxiliares, es decir, si el conjunto de auxiliares seleccionadas es vacío, el algoritmo trabaja, en principio, con todas las auxiliares de las que dispone la coordinadora. Sin embargo, de todas estas auxiliares, se seleccionan unicamente aquellas cuya planificación actual respete su jornada contratada y que, además, no tengan la bolsa de horas en positivo (es decir, que no hayan trabajado de más en el pasado).

Planificación inicial

En caso de que la coordinadora haya seleccionado auxiliares, será necesario suprimir de su planificación todos aquellos servicios correspondientes a usuarios cuyos estados sean "baja" o "inactivo", puesto que las auxiliares no pueden atender a dichos usuarios. La eliminación de estos servicios puede dar lugar a huecos en las planificaciones de las auxiliares, de modo que será necesario reducir lo máximo posible estos descansos aplicando una optimización.

Selección de servicios a replanificar

En esta fase también se estudia el plan de intervención de los usuarios seleccionados, con el objetivo de determinar el conjunto de servicios que se deben replanificar (si es que los hay). Para ello se analizan los servicios correspondientes a cada uno de los usuarios y se comprueba si están correctamente planificados estudiando las siguientes características:

- El servicio no está planificado.
- El servicio no tiene la duración adecuada.
- El horario de realización del servicio no se encuentra dentro de su ventana disponible.
- El nivel de afinidad entre el usuario y la auxiliar que lo atiende es 0 o 1, es decir, el usuario no quiere seguir siendo atendido por dicha auxiliar.

En caso de que se tengan servicios que cumplan alguna de las características anteriores, el algoritmo se dirige a la fase de *Replanificación*, que se explicará posteriormente, en la cual se planifican dichos servicios.

Si, por el contrario, no se tienen servicios por replanificar, se realiza un análisis de la planificación de las auxiliares seleccionadas con el objetivo de comprobar si es necesario optimizarla. Para ello se estudia el tiempo de descanso de la planificación y, si ésta no es adecuada (es decir, si el descanso de la planificación toma valor no nulo), será necesario llevar a cabo una *Optimización*, cuyo funcionamiento se detalla más adelante, con el objetivo de mejorarla.

4.3.2. Fase de replanificación

Esta segunda fase del algoritmo se lleva a cabo cuando se tienen servicios por replanificar. En ella se determina la auxiliar que debe realizar cada uno de estos servicios y el horario en que deben llevarse a cabo. Para ello, se sigue el procedimiento que se describe a continuación.

Ordenar servicios

El primer paso que se lleva a cabo en esta fase de replanificación consiste en ordenar los servicios a replanificar, ya que recorrer los servicios de diferentes formas puede dar lugar a la obtención de resultados dispares.

Los servicios se ordenan por persona y franja. De tal forma que, primero se disponen las personas según las franjas en las que requieran servicios y, seguidamente, se estructuran sus servicios dependiendo de la franja en la que se deba realizar cada uno de ellos; ordenando los servicios de cada una de dichas

franjas según el día de las semana al que pertenecen. En ambos casos, el orden que se sigue para colocar las franjas es: mañana, mediodía, tarde, noche e indistinto.

Ordenar las auxiliares

A continuación, es necesario obtener una lista con las auxiliares disponibles ordenadas de mejor a peor. Esta estructuración se realiza para cada usuario y franja en la que presenta servicios por replanificar.

Las auxiliares se ordenan teniendo en cuenta los siguientes aspectos:

- Afinidad entre la auxiliar y el usuario que se esté considerando.
- Diferencia entre la jornada máxima permitida y la jornada trabajada.
- Desplazamiento medio entre el usuario considerado y los servicios que ya tiene asignados la auxiliar.

En esta lista de auxiliares ordenadas no se consideran aquellas que hayan alcanzado su jornada máxima permitida o estén vetadas por el usuario considerado.

Otro aspecto que también se tiene en cuenta, tras haber realizado la ordenación, es comprobar si alguna de las auxiliares disponibles está realizando servicios análogos (servios del mismo usuario y misma franja) al servicio considerado. En caso de ser así, la auxiliar pasaría a estar al principio de la lista.

Colocar los servicios

Habiendo ordenado las auxiliares, se recorren los servicios para añadirlos a la planificación en el mejor horario posible. Para cada servicio, se exploran las auxiliares ordenadas de modo que se trata de colocar dicho servicio en su planificación. El procedimiento utilizado es el siguiente:

1. Se selecciona la planificación de la auxiliar, ya sea la inicial (si la auxiliar no ha sido modificada) o la planificación con la que está trabajando el algoritmo (en casos en que el método ya haya modificado el plan de la auxiliar).

2. Se añade el servicio a la planificación de la auxiliar, seleccionando el horario en que se generen menores solapamientos y descansos (se utiliza un orden lexicográfico, de modo que, primero se comprueban el solapamiento y seguidamente el descanso. Es decir, se da prioridad a la elección de horarios en los cuales se generan solapamientos pequeños entre el servicio nuevo y los servicios que ya tenía planificados la auxiliar). Si al añadir el servicio, la planificación de la auxiliar está saturada, éste debe asignarse a la siguiente auxiliar de la lista o, en caso de que ya se hayan recorrido todas ellas, a una auxiliar ficticia. Entendiendo como saturada, una planificación que supera la jornada permitida de la auxiliar, o que presenta incompatibilidades entre sus servicios, es decir, solapamientos que está claro que no pueden ser eliminados.

3. Se optimiza la planificación, para así, eliminar los posibles solapamientos y/o descansos generados al añadir el nuevo servicio; para ello se emplea el método del *simulated annealing*.

Si la planificación obtenida tras aplicar la optimización no presenta solapamientos, se procede a planificar el servicio siguiente. En caso contrario, se repite el procedimiento utilizando la siguiente auxiliar de la lista ordenada. Si se ha probado con todas la auxiliares y ninguna de ellas puede realizar el servicio correctamente, éste se asigna a una auxiliar ficticia (esto se realiza con el objetivo de mostrar a la coordinadora que el servicio no puede ser planificado con las auxiliares de las que se dispone, y que será necesario ampliar contratos o seleccionar más auxiliares).

Igualar inicios

Por último, se tratan de igualar los horarios en los que cada auxiliar comienza su jornada en los diferentes días (ya que la optimización puede provocar que las horas de inicio sean dispares y, en ocasiones, esto puede solventarse).
Para ello se recorren las auxiliares y se agrupan sus planificaciones, de modo que se trabaja por un lado, con los fines de semana y, por otro, con los días semanales. En cada uno de estos grupos se comprueba si es posible desplazar los servicios de la auxiliar hasta conseguir que sus jornadas comiencen todos los días a la misma hora y, en caso de ser así, se aplica la modificación necesaria para obtener la igualdad en horarios de inicio.

4.3.3. Optimización

Como se explicaba en fases anteriores, en ocasiones es necesario realizar una optimización para eliminar descansos y/o solapamientos. Para ello, se emplea una versión del método *simulated annealing* adaptada al problema en estudio.

Simulated annealing

El algoritmo de optimización consiste en aplicar el recocido simulado, explicado en 4.1, con dos particularidades.

La primera consiste en obtener un vecindario, a partir de la solución actual, y considerar como solución candidata a la mejor del vecindario (en lugar de considerar una solución obtenida de forma aleatoria, como sucede en la versión original del método).

Por otro lado, a la hora de seleccionar el movimiento empleado para generar cada vecino, se realiza una elección aleatoria basada en la probabilidad de cada uno de ellos. Dicha probabilidad al principio del algoritmo es igual para todos los movimientos y, en cada iteración, se aumenta la correspondiente con el movimiento que haya generado una mayor proporción de mejores vecinos (considerando como mejores vecinos aquellas soluciones que tienen mejor valor de la función objetivo que la planificación que se esté considerando en esa iteración).

Función objetivo

La función objetivo que se utiliza en estas variantes del algoritmo es una función lexicográfica cuyos criterios son:

1. El tiempo total de solapamiento en la planificación.
2. Penalización por el número de servicios realizados por auxiliares ficticias.
3. El tiempo total empleado en descansos que tienen las auxiliares en su planificación, exceptuando aquellos que tienen duración mayor a las dos horas.
4. El tiempo total que los servicios son realizados fuera de sus ventanas de tiempo óptimas.

De modo que, para comparar dos soluciones, primero se cotejan los tiempos de solapamiento, de tal forma que se considera mejor aquella que proporcione menor tiempo de solapamiento. En caso de que ambas tomen el mismo valor, se analizan las penalizaciones por servicios asignados a auxiliares ficticias, de modo que, una solución será mejor que la otra si su penalización es menor. Si las soluciones están empatadas en cuanto a penalización, se comparan los tiempos de descanso, por lo cual se considera mejor solución aquella que tenga menor descanso. Si ambas toman el mismo valor se pasará a comparar el tiempo en que los servicios se realizan fuera de sus ventanas optimas, considerándose mejor solución la que menor tiempo emplee en realizar servicios fuera de las ventanas optimas. Por último, si las dos soluciones toman los mismos valores en todos los criterios, las planificaciones se consideran iguales.

Debido a que, en ocasiones, puede ser necesario eliminar las auxiliares ficticias que se tienen en la planificación (es decir, se desea asignar sus servicios a auxiliares reales) es necesario definir una nueva función objetivo lexicográfica. Esta función consta de:

1. Penalización por el número de servicios realizados por auxiliares ficticias.
2. El tiempo total de solapamiento en la planificación.
3. El tiempo total empleado en empleado en descansos que tienen las auxiliares en su planificación, exceptuando aquellos que tienen duración mayor a las dos horas.
4. El tiempo total que los servicios son realizados fuera de sus ventanas de tiempo óptimas.

De modo que, el algoritmo utilizará ambas funciones para combinar el objetivo de eliminar las auxiliares ficticias y el de eliminar los solapamientos; evitando así, que las soluciones traten de optimizar solamente uno de los dos objetivos principales.

Para pasar de una función objetivo a la otra, el algoritmo analiza las soluciones que va generando, de modo que, se intercambien las funciones en el momento en que las últimas planificaciones obtenidas no mejoren ni el valor de la penalización por ficticias ni el del solapamiento. Este cambio solamente se realiza cuando alguna de estas dos características toma valores no nulos, ya que, es en esos casos cuando se puede mejorar la planificación cambiando de función objetivo.

Cabe destacar que, estas funciones objetivo no se corresponden con la definida en el capítulo 3. Esto se debe, principalmente, a que el algoritmo ha sido diseñado para generar una solución inicial, que no tiene porqué ser factible e ir modificándola hasta que sea factible y, preferiblemente, óptima. Es por esto por lo que los dos primeros objetivos de la función objetivo del algoritmo no se encontraban en la función objetivo del problema de programación lineal.

Aun así, todos los elementos considerados en el objetivo del problema de programación lineal también se tienen en cuenta en este algoritmo, como se muestra a continuación.

Las afinidades y los tiempos de desplazamiento no se consideran en la función objetivo del algoritmo, pero si se tienen en cuenta, tanto en la fase de ordenación de las auxiliares (donde cuanto mayor sea la afinidad y menor sea el tiempo de desplazamiento, mejor se considera la auxiliar), como en los movimientos que se emplean durante la optimización (los servicios que se cambian de auxiliar solamente contemplan como posibles nuevas auxiliares aquellas que tienen alta afinidad con el usuario y poco desplazamiento con el mismo).

En cambio, los tiempos de descanso, aspecto que en la función objetivo del problema de programación lineal se considera al minimizar las jornadas de las auxiliares, y las ventanas óptimas sí que se consideran en la función objetivo del algoritmo; puesto que son el tercer y cuarto objetivo de la función lexicográfica definida anteriormente.

Condiciones de parada

Por último, es necesario mencionar que la optimización mediante *simulated annealing* finaliza cuando se satisface alguna de las siguientes condiciones:

- El número de iteraciones realizadas alcanza el número máximo de iteraciones permitido.
- El número de iteraciones realizadas, sin proporcionar mejores soluciones, alcanza el número máximo de iteraciones sin mejora permitido.
- La solución obtenida es óptima en cuanto a los tres primeros elementos considerados en el función objetivo.

Movimientos

A continuación, se describen los movimientos que se aplican a las planificaciones durante la implementación del *simulated annealing*. Estos movimientos pueden dividirse en dos clases: los movimientos básicos, que son los movimientos más sencillos y que suelen generar las mejores soluciones, y los movimientos combinados, que surgen de combinar o modificar movimientos básicos y que, en la práctica, no suelen utilizarse tanto como los otros pero que, en ocasiones, son imprescindibles para obtener mejores soluciones.

Para ilustrar los movimientos básicos se utiliza el ejemplo mostrado en la figura 4.2. En ella se tiene la planificación de dos auxiliares; la primera con tres servicios correctamente planificados (color azul) y, la segunda, con cuatro servicios iniciales (color azul) a la que, además, se le ha ha añadido un nuevo servicio (rojo). Este servicio se ha colocado en la posición que se muestra en la figura para poder ejemplificar los movimientos que se definirán a continuación, es decir, no se ha seguido ningún tipo procedimiento específico.

Entre cada par de servicios consecutivos se presenta el tiempo de desplazamiento como una linea discontinua de color rosa y, para cada servicio, se tiene la ventana horaria disponible en verde y la óptima en naranja. En esta planificación inicial, se puede comprobar que, el nuevo servicio se solapa con dos de los servicios (en concreto, el inicio del nuevo servicio se solapa con el 6 y el final del nuevo servicio se solapa con el 7) que tiene planificados la segunda auxiliar.

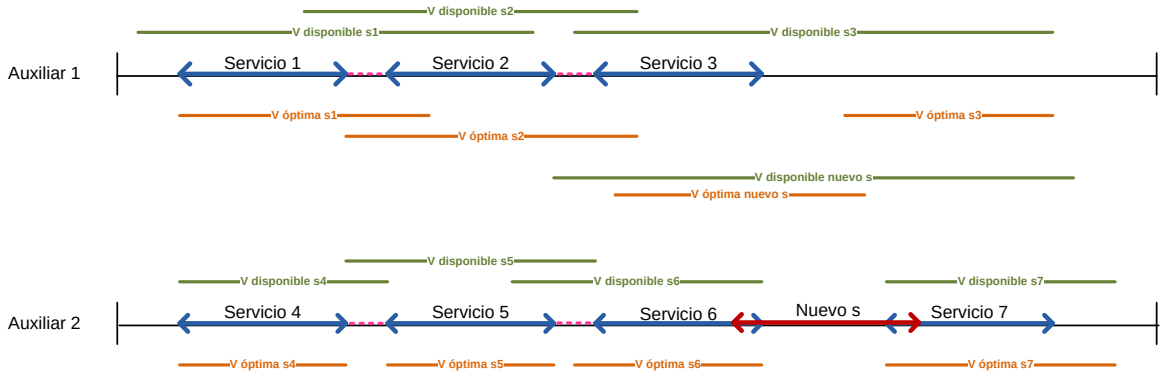


Figura 4.2: Planificación inicial

Cambiar un servicio de auxiliar Este movimiento consiste en, dado un servicio y una auxiliar (distinta a la que realiza dicho servicio en el instante considerado), reasignar el servicio a la nueva auxiliar.

La selección de la nueva auxiliar se lleva a cabo teniendo en cuenta el nivel de afinidad entre el usuario y dicha auxiliar, puesto que no se pueden realizar asignaciones que disminuyan la satisfacción de los clientes, y el tiempo de desplazamiento entre el usuario considerado y todos aquellos a los que visita la auxiliar, ya que estos tiempos se quieren minimizar.

En la figura 4.3 se presenta el ejemplo en el que a la planificación se le ha aplicado este movimiento, de modo que, el nuevo servicio (rojo) pasa a ser realizado por la primera auxiliar, reduciéndose así el tiempo de solapamiento de la planificación (ya que ahora el nuevo servicio solamente se solapa con el final del servicio 3).

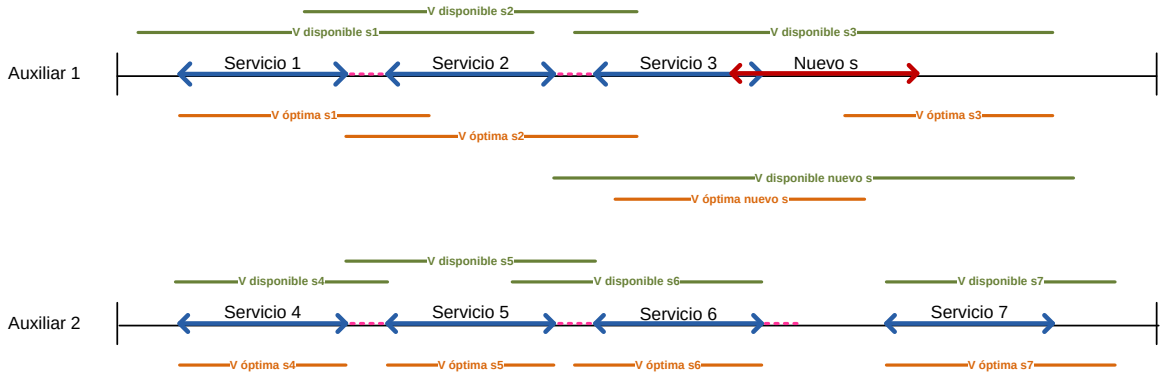


Figura 4.3: Planificación tras cambiar el servicio de auxiliar

Replanificar un servicio Este movimiento consiste en, dado un servicio y cierta cantidad de minutos, modificar el horario en que se realiza dicho servicio, añadiéndole los minutos considerados. En caso de que los minutos sean positivos, el servicio se retrasará, en cambio, si los minutos son negativos el horario de realización del servicio se adelantará.

Con el objetivo de reducir los tiempos de computación, en la mayoría de las realizaciones de este movimiento, los posibles minutos considerados se seleccionan de forma aleatoria de entre aquellos que proporcionarán soluciones factibles, es decir, soluciones en las cuales el servicio se mantenga dentro de su ventana disponible, y se respeten los horarios de los demás servicios de la auxiliar.

En la figura 4.4 se muestra el ejemplo tras haber replanificado (atrasado) el nuevo servicio, de manera que, se elimina el solapamiento que tenía la planificación, entre el servicio 3 y el nuevo servicio, y se genera un descanso en ella.

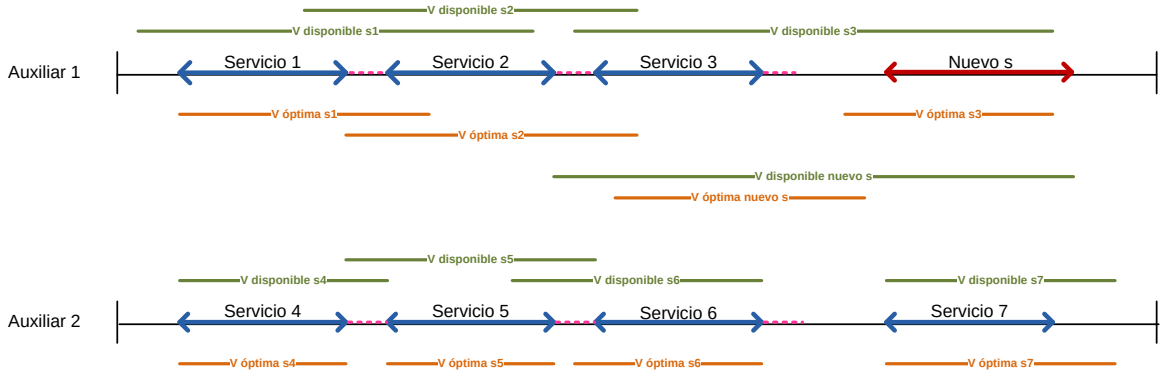


Figura 4.4: Planificación tras cambiar el horario del servicio

Desplazar varios servicios Este movimiento consiste en, dado un servicio, cierta cantidad de minutos y el sentido, desplazar los servicios anteriores o siguientes (dependiendo del sentido) en base a los minutos dados. Es decir, a todos los servicios a desplazar se les modifica su horario, añadiéndoles los minutos dados.

Con el objetivo de reducir los tiempos de computación, en la mayoría de las realizaciones de este movimiento, los minutos y el sentido se seleccionan de modo que las soluciones que proporcionen

sean factibles, es decir, que se respeten las ventanas de los servicios que se desplazan y los horarios de los demás servicios.

En la figura 4.5 se tiene el ejemplo después de aplicar este movimiento, es decir, tras retrasar los tres primeros servicios (servicios 1, 2 y 3) de la auxiliar 1. Al hacer esto se elimina el descanso que presentaba la planificación de dicha auxiliar (entre el servicio 3 y el nuevo servicio).

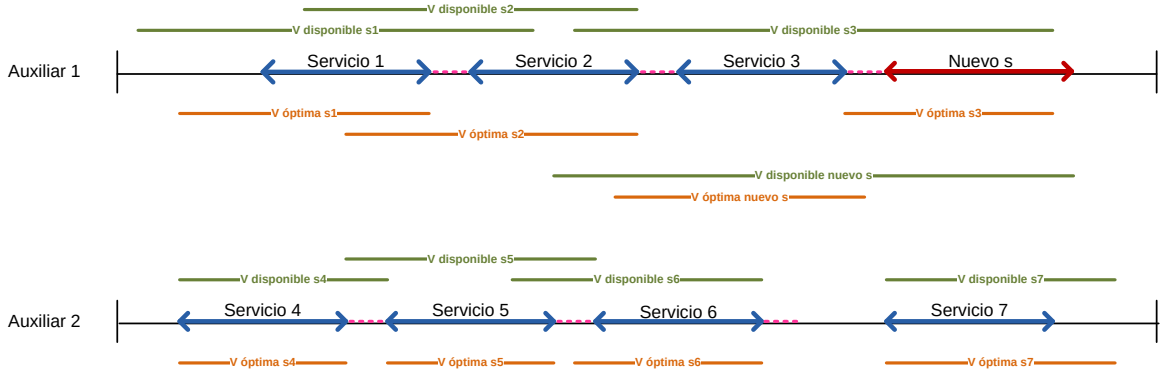


Figura 4.5: Planificación tras desplazar un bloque de servicios

Intercambiar servicios de una auxiliar Este movimiento consiste en, dados dos servicios que son realizados por una misma auxiliar en un cierto día, intercambiar los horarios de inicio de los servicios (y por consiguiente adaptar los horarios de fin según la duración de los mismos). En este movimiento también se tiene la opción de, tras haber realizado el intercambio, aplicar una replanificación de los servicios (esto se debe a que se comprobó, mediante experimentación, que las soluciones proporcionadas eran mejores añadiendo esta modificación).

Con el objetivo de reducir tiempos de computación, en la mayoría de las realizaciones de este movimiento, solamente se seleccionan servicios y minutos (para la replanificación de los mismos) que proporcionan soluciones factibles, es decir, soluciones en las que los servicios están dentro de sus ventanas disponibles y se respeten los horarios de los demás servicios de la auxiliar.

Por último, en la figura 4.6, se presenta la planificación obtenida después de intercambiar el horario en que se realizan el nuevo servicio y el servicio 3. Al hacer esto ambos servicios pasan a realizarse dentro de su horario óptimo, lo cual mejora la planificación de la auxiliar.

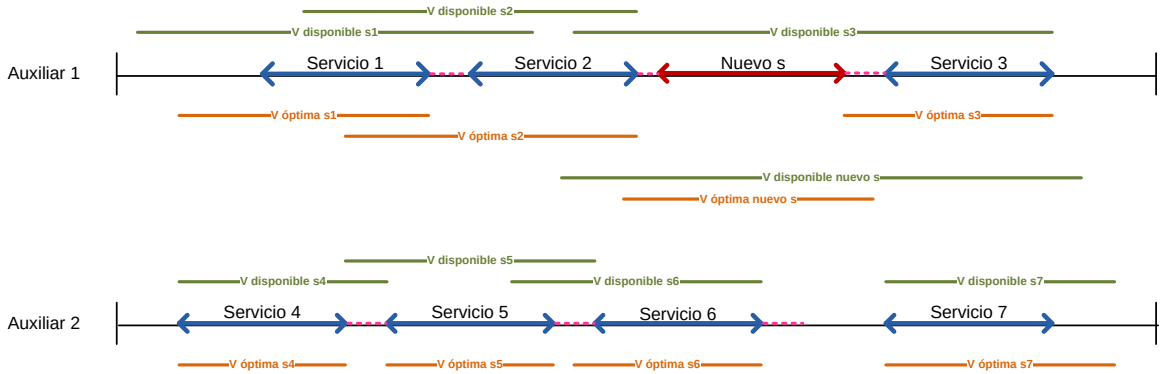


Figura 4.6: Planificación tras intercambiar el horario en que se realizan dos servicios

Los siguientes movimientos surgen como combinación o modificación de alguno de los movimientos que se acaban de describir.

Intercambiar servicios entre auxiliares Este movimiento consiste en, dados dos servicios que son realizados por diferentes auxiliares, y que corresponden al mismo día, intercambiar las auxiliares que los realizan.

Desplazar varios servicios con doble sentido Este movimiento consiste en, dado un servicio, aplicar dos veces el movimiento *desplazar varios servicios*, una para los servicios siguientes y otra para los anteriores (incluyendo en uno de los dos casos al servicio que se está considerando como pivote). En ambos casos, en la mayoría de las realizaciones de este movimiento, se seleccionan los minutos de modo que las soluciones obtenidas sean factibles, es decir, se respeten las ventanas disponibles de los servicios que se desplazan y los horarios de los demás servicios.

Intercambiar varios servicios de auxiliar Este movimiento consiste en, partiendo de dos servicios asignados a diferentes auxiliares:

1. Para cada uno de los servicios considerados, obtener el conjunto de servicios que cumplen las siguientes características:
   a) Los servicios corresponden al mismo usuario que el del servicio considerado.
   b) Los servicios los realiza la misma auxiliar a la que está asignado el servicio considerado.
   c) Los servicios pertenecen a la misma franja que el servicio considerado.
   d) Los servicios pertenecen al mismo tipo de día que el servicio considerado, es decir, día de semana o fin de semana.
2. Se intercambian las auxiliares que realizan estos conjuntos de servicios.

Intercambiar y desplazar Este movimiento consiste en aplicar el movimiento *intercambiar servicios de una auxiliar* y, seguidamente, aplicar a la planificación el movimiento *desplazar varios servicios*.

4.3.4. Otras variantes

Se acaba de describir la versión definitiva del algoritmo, pero durante el desarrollo del mismo se tuvieron otras en cuenta, con el fin de obtener la que mejor comportamiento presentaba. Las variantes consideradas fueron las siguientes.

Planificar todos los servicios
En esta variante se añaden a la planificación todos los servicios a la vez, asignándolos a la mejor auxiliar disponible y colocándolos en la posición que menores solapamientos y descansos provoque. Habiendo planificado todos los servicios se aplica una optimización, basada en el método del *simulated annealing*, para eliminar los posibles descansos y/o solapamientos que se tengan en la planificación.

Búsqueda local
Las dos versiones del algoritmo ya explicadas (planificación de los servicios uno a uno y planificación de todos los servicios a la vez) se implementaron también empleando el método de *búsqueda local* para realizar la optimización de las planificaciones. Este método se basa en, partiendo de una solución inicial, busca en su vecindario una mejor (en términos de la función objetivo) que ella; si la encuentra actualiza la solución considerada y continúa hasta que no se pueda mejorar más la solución actual.

Los experimentos realizados indicaron que el método que mejores resultados proporcionaba y, además, que menor tiempo de computación empleaba para ello, es el que planifica los servicios uno por uno y emplea el método del *simulated annealing* para realizar la optimización, es decir, el algoritmo definido con detalle en este capítulo.

Capítulo 5

Estudio de los métodos de resolución

En este capítulo, se estudia el comportamiento de la aproximación heurística desarrollada para resolver el problema de Mayores; para ello se compararán las soluciones que proporciona el algoritmo con las soluciones exactas de una batería de ejemplos.

El problema de programación lineal (definido en el capítulo 3) se ha resuelto de forma exacta empleando el solver de problemas de programación lineal entera mixta `Gurobi`. Es necesario recalcar que, debido a la alta complejidad del modelo, solamente se han podido resolver de forma exacta problemas de pequeño tamaño.

5.1. Características y parámetros

A continuación se determina la naturaleza de los ejemplos que se han utilizado para estudiar los métodos, así como los parámetros considerados para aplicar la resolución exacta y la aproximada.

5.1.1. Ejemplos estudiados

Con el objetivo de comparar las soluciones exactas del problema con aquellas que proporciona el algoritmo diseñado, se ha considerado un conjunto de ejemplos que consta de las siguientes características.

Tipo de ejemplo Los ejemplos considerados son casos "reales", es decir, casos sintéticos que presentan el mismo comportamiento que aquellos a los que se enfrentan las coordinadoras de la empresa en su día a día. En estos ejemplos, se tiene un conjunto de servicios, con sus respectivos horarios óptimos y disponibles, los cuales deben ser asignados a una auxiliar.

Estos ejemplos no llegan a ser idénticos a los casos reales puesto que, debido a lo complejo que es el problema de programación lineal, solamente se consideran casos de uno o dos días de la semana. En los casos reales, las coordinadoras se enfrentan a problemas, en los cuales, deben realizar planificaciones para los siete días de la semana.

En estos ejemplos no se trata de resolver ninguna de las incidencias descritas en 1.3.1, sino que se trata de obtener planificaciones desde cero. Es decir, partiendo de auxiliares vacías se quieren planificar todos los servicios, determinados en cada uno de los ejemplos, de la mejor forma posible.

Servicios El número de servicios que se quiere planificar, para cada día, varía según el ejemplo considerado (entre 4 y 8 servicios). Es necesario tener en cuenta que la duración de los servicios no siempre es la misma, sino que el servicio más breve tiene una duración de 30 minutos y el más duradero de 195 minutos.

Auxiliares En cada uno de los ejemplos se considera únicamente una auxiliar disponible cuya jornada laboral máxima permitida es de 12 horas, salvo en el ejemplo 11 que se consideran dos auxiliarles.

Afinidad La afinidad considerada entre los usuarios que requieren los servicios y la auxiliar disponible es igual para todos ellos, a excepción del ejemplo 11.

Días Debido a que el tamaño de los ejemplos considerados es reducido, la mayoría de ellos están formados por servicios correspondientes a un único día. A pesar de ello, también se han resuelto ejemplos en los que se tienen dos grupos de servicios, donde cada uno de ellos pertenece a un día diferente.

Desplazamientos Los tiempos de desplazamiento entre los diferentes usuarios considerados se corresponden con los tiempos empleados en desplazarse entre las viviendas de los mismos, tomando valores entre 5 y 20 minutos.

Los ejemplos considerados para realizar la comparación entre el algoritmo diseñado y la solución exacta son los siguientes:

Ejemplo 1
Este ejemplo está formado por 6 servicios, los cuales deben ser realizados por una auxiliar en un único día. Los datos relativos a los servicios se presentan en la siguiente tabla.

servicio	*duración*	*inicio Disp*	*fin Disp*	*inicio Opt*	*fin Opt*	*día*
1	50	6:30	17:30	11:00	13:30	1
2	60	7:00	13:00	9:00	11:00	1
3	60	7:00	14:30	9:00	12:00	1
4	60	7:00	13:00	7:00	13:00	1
5	60	9:00	15:00	12:00	15:00	1
6	60	16:00	21:00	16:00	21:00	1

Tabla 5.1: Datos de los servicios considerados en el ejemplo 1

El tiempo de desplazamiento entre todos los servicios considerados en este ejemplo es de 5 minutos.

Ejemplo 2
Este ejemplo está formado por 5 servicios, que deben ser realizados por una auxiliar en un único día. Los datos relativos a los servicios se presentan en la siguiente tabla.

servicio	*duración*	*inicio Disp*	*fin Disp*	*inicio Opt*	*fin Opt*	*día*
1	80	8:00	14:30	9:00	12:00	1
2	60	8:00	15:00	9:00	12:00	1
3	60	8:00	16:00	9:00	12:00	1
4	45	18:00	23:00	18:00	23:00	1
5	60	19:00	23:00	19:00	23:00	1

Tabla 5.2: Datos de los servicios considerados en el ejemplo 2

El tiempo de desplazamiento entre todos los servicios considerados en este ejemplo es de 5 minutos.

Ejemplo 3
Este ejemplo está formado por 8 servicios, los cuales deben ser realizados por una auxiliar en un único día. Los datos relativos a los servicios se presentan en la siguiente tabla.

servicio	*duración*	*inicio Disp*	*fin Disp*	*inicio Opt*	*fin Opt*	*día*
1	30	7:00	11:00	7:00	11:00	1
2	60	7:00	13:00	8:00	11:00	1
3	60	7:00	13:00	7:00	13:00	1
4	60	9:00	11:00	9:00	11:00	1
5	60	15:30	19:30	16:00	19:00	1
6	30	16:00	21:00	16:00	18:30	1
7	60	16:00	21:00	17:00	20:00	1
8	60	16:00	20:00	16:30	20:00	1

Tabla 5.3: Datos de los servicios considerados en el ejemplo 3

El tiempo de desplazamiento entre todos los servicios considerados en este ejemplo es de 5 minutos.

Ejemplo 4
Este ejemplo está formado por 8 servicios, los cuales deben ser realizados por una auxiliar en un único día. Los datos relativos a los servicios se presentan en la siguiente tabla.

servicio	*duración*	*inicio Disp*	*fin Disp*	*inicio Opt*	*fin Opt*	*día*
1	30	7:00	12:30	9:00	12:00	1
2	30	7:30	10:00	9:00	10:00	1
3	60	8:00	13:00	10:30	12:30	1
4	60	9:00	15:00	9:00	12:00	1
5	60	7:30	15:30	9:00	12:00	1
6	60	9:00	17:00	9:00	12:00	1
7	45	8:00	23:00	9:00	12:00	1
8	30	15:00	20:00	15:00	20:00	1

Tabla 5.4: Datos de los servicios considerados en el ejemplo 4

Los tiempos de desplazamiento entre los servicios considerados en este ejemplo son los siguientes.

	1	2	3	4	5	6	7	8
1	0	15	15	15	15	15	15	15
2	15	0	15	15	15	15	15	15
3	15	15	0	15	15	5	5	15
4	15	15	15	0	15	5	10	15
5	15	15	15	15	0	10	10	15
6	15	15	5	5	10	0	15	15
7	15	15	5	10	10	15	0	15
8	15	15	15	15	15	15	15	0

Tabla 5.5: Tiempos de desplazamiento del ejemplo 4

Ejemplo 5

Este ejemplo está formado por 6 servicios, los cuales deben ser realizados por una auxiliar en un único día. Los datos relativos a los servicios se presentan en la siguiente tabla.

servicio	*duración*	*inicio Disp*	*fin Disp*	*inicio Opt*	*fin Opt*	*día*
1	60	6:30	12:00	9:00	12:00	1
2	60	7:00	12:30	9:00	12:00	1
3	60	7:30	12:00	9:00	12:00	1
4	60	10:00	15:00	10:00	13:00	1
5	35	12:00	15:30	13:00	15:30	1
6	30	15:00	23:00	15:00	20:00	1

Tabla 5.6: Datos de los servicios considerados en el ejemplo 5

El tiempo de desplazamiento entre todos los servicios considerados en este ejemplo es de 5 minutos.

Ejemplo 6

Este ejemplo está formado por 13 servicios, los cuales deben ser realizados por una auxiliar en dos días diferentes. Los datos relativos a los servicios se presentan en la siguiente tabla.

servicio	*duración*	*inicio Disp*	*fin Disp*	*inicio Opt*	*fin Opt*	*día*
1	70	7:00	14:00	9:00	12:00	1
2	60	8:30	12:30	9:00	12:00	1
3	165	7:30	15:30	9:00	12:00	1
4	80	15:00	20:00	15:00	20:00	1
5	45	18:00	23:00	18:00	23:00	1
6	50	15:00	23:00	15:00	20:00	1
7	30	7:00	13:00	7:00	13:00	2
8	60	8:30	12:30	9:00	12:00	2
9	120	7:30	16:30	9:00	12:00	2
10	70	7:00	23:00	7:00	13:00	2
11	80	15:00	20:00	15:00	20:30	2
12	45	18:00	23:00	18:00	20:00	2
13	50	15:00	23:00	15:00	20:00	2

Tabla 5.7: Datos de los servicios considerados en el ejemplo 6

El tiempo de desplazamiento entre todos los servicios considerados en este ejemplo es de 20 minutos.

Ejemplo 7

Este ejemplo está formado por 7 servicios, los cuales deben ser realizados por una auxiliar en un único día. Los datos relativos a los servicios se presentan en la siguiente tabla.

servicio	*duración*	*inicio Disp*	*fin Disp*	*inicio Opt*	*fin Opt*	*día*
1	60	7:30	18:00	9:00	12:00	1
2	60	8:00	11:00	9:00	11:00	1
3	30	9:00	12:00	9:00	12:00	1
4	60	8:00	13:00	9:00	12:00	1
5	60	9:00	18:00	9:00	12:00	1
6	60	7:00	17:00	7:00	13:00	1
7	60	15:00	22:00	15:00	20:00	1

Tabla 5.8: Datos de los servicios considerados en el ejemplo 7

Los tiempos de desplazamiento entre los servicios considerados en este ejemplo son los siguientes.

	1	2	3	4	5	6	7
1	0	5	5	5	5	5	5
2	5	0	0	5	5	5	5
3	5	0	0	5	5	5	5
4	5	5	5	0	5	5	5
5	5	5	5	5	0	5	5
6	5	5	5	5	5	0	5
7	5	5	5	5	5	5	0

Tabla 5.9: Tiempos de desplazamiento del ejemplo 7

Ejemplo 8
Este ejemplo está formado por 4 servicios, los cuales deben ser realizados por una auxiliar en un único día. Los datos relativos a los servicios se presentan en la siguiente tabla.

servicio	*duración*	*inicio Disp*	*fin Disp*	*inicio Opt*	*fin Opt*	*día*
1	195	6:30	13:00	9:00	12:00	1
2	60	9:30	18:30	11:00	13:00	1
3	30	9:30	17:30	12:00	15:00	1
4	1200	15:00	23:00	15:00	20:00	1

Tabla 5.10: Datos de los servicios considerados en el ejemplo 8

Los tiempos de desplazamiento entre los servicios considerados en este ejemplo son los siguientes.

	1	2	3	4
1	0	25	25	25
2	25	0	10	25
3	25	10	0	25
4	25	25	25	0

Tabla 5.11: Tiempos de desplazamiento del ejemplo 8

Ejemplo 9

Este ejemplo está formado por 5 servicios, que deben ser realizados por una auxiliar en un mismo día. Los datos relativos a los servicios se presentan en la siguiente tabla.

servicio	*duración*	*inicio Disp*	*fin Disp*	*inicio Opt*	*fin Opt*	*día*
1	60	7:30	12:00	11:00	12:00	1
2	60	7:00	11:00	9:00	11:00	1
3	60	9:00	16:30	9:00	12:00	1
4	60	16:30	21:30	16:30	21:30	1
5	60	16:30	23:00	21:00	23:00	1

Tabla 5.12: Datos de los servicios considerados en el ejemplo 9

Los tiempos de desplazamiento entre los servicios considerados en este ejemplo son los siguientes.

	1	2	3	4	5
1	0	15	20	20	20
2	15	0	20	20	20
3	20	20	0	5	20
4	20	20	5	0	20
5	20	20	20	20	0

Tabla 5.13: Tiempos de desplazamiento del ejemplo 9

Ejemplo 10

Este ejemplo está formado por 7 servicios, los cuales deben ser realizados por una auxiliar en un único día. Los datos relativos a los servicios se presentan en la siguiente tabla.

servicio	*duración*	*inicio Disp*	*fin Disp*	*inicio Opt*	*fin Opt*	*día*
1	60	7:00	16:00	7:00	13:00	1
2	60	7:00	14:30	9:00	11:00	1
3	60	7:00	16:00	9:00	12:00	1
4	60	7:30	17:30	9:00	12:00	1
5	60	15:00	20:00	15:00	20:00	1
6	45	15:00	20:00	15:00	20:00	1
7	45	15:00	20:00	15:00	20:00	1

Tabla 5.14: Datos de los servicios considerados en el ejemplo 10

Los tiempos de desplazamiento entre los servicios considerados en este ejemplo son los siguientes.

	1	2	3	4	5	6	7
1	0	15	15	15	15	15	15
2	15	0	5	15	15	15	15
3	15	5	0	10	15	15	15
4	15	15	10	0	15	15	15
5	15	15	15	15	0	5	15
6	15	15	15	15	5	0	0
7	15	15	15	15	15	0	0

Tabla 5.15: Tiempos de desplazamiento del ejemplo 10

Ejemplo 11
Este ejemplo está formado por 6 servicios, los cuales deben ser realizados por dos auxiliares en un único día. Los datos relativos a los servicios se presentan en la siguiente tabla.

servicio	*duración*	*inicio Disp*	*fin Disp*	*inicio Opt*	*fin Opt*	*día*	*afin Aux1*	*afin Aux2*
1	60	7:00	13:00	7:00	13:00	1	2	5
2	60	7:00	13:00	8:00	11:00	1	2	5
3	60	9:00	11:00	9:00	12:00	1	4	4
4	60	15:30	19:30	16:00	19:00	1	5	2
5	30	16:00	21:00	16:00	18:30	1	5	2
6	60	16:00	21:00	17:00	20:00	1	4	4

Tabla 5.16: Datos de los servicios considerados en el ejemplo 11

Cabe destacar que en este ejemplo, a diferencia de los anteriores, se tienen dos columnas más, *afin Aux1* y *afin Aux2*; en ellas se especifica el nivel de afinidad que presenta cada uno de los servicios con las auxiliares disponibles.
El tiempo de desplazamiento entre todos los servicios considerados en este ejemplo es de 5 minutos.

5.1.2. Parámetros solución exacta

Como ya se veía en el capítulo 3, para resolver el problema de planificación general de Mayores de forma exacta se utiliza una función objetivo ponderada, de modo que, para aplicar el método exacto a los ejemplos considerados será necesario determinar los coeficientes de ponderación de la misma.

El algoritmo heurístico no ha sido diseñado exactamente para resolver el problema de programación lineal entera definido en el capítulo 3, sino que su principal objetivo es encontrar una planificación que resuelva las incidencias consideradas modificando lo menos posible la solución inicial, tal y como requería Mayores. Debido a esto, el objetivo de optimizar las ventanas óptimas de los servicios, es decir, el cuarto elemento considerado en la función objetivo del algoritmo, se tiene en cuenta en éste de forma inapreciable, puesto que optimizar dicho aspecto daría lugar a aplicar excesivos cambios a la solución inicial. Por tanto, para poder comparar el funcionamiento del algoritmo propuesto con el algoritmo exacto, no se considerará, en la función objetivo del problema de programación lineal entera, la parte correspondiente a las ventanas óptimas.

De modo que, los coeficientes considerados para ponderar la función objetivo son $\alpha = \beta = 1$, para que la jornada de las auxiliares y la afinidad tengan el mismo peso, y $\gamma = 0$, para no considerar las ventanas óptimas.

5.1.3. Parámetros solución aproximada

En el capítulo 4 se determina que el algoritmo diseñado presenta varias características que pueden ser modificadas, dependiendo de los aspectos a los cuales se quiera dar más prioridad a la hora de obtener las soluciones. Los parámetros utilizados para resolver los ejemplos considerados son los siguientes.

Temperatura inicial La temperatura inicial considerada es $T_0 = 100$.

Probabilidad de transición La probabilidad de transición utilizada es la descrita por Kirkpatrick *et al.* (1983), que se define en 4.1.2.

Enfriamiento El enfriamiento de la temperatura se basa en la formulación de Laarhoven, con $\beta =$ 0,85, descrita en 4.1.2.

Iteraciones El número máximo de iteraciones que se considera es $n = 100$ y el número máximo de iteraciones sin mejora es $m = 10$.

Vecindario El tamaño del vecindario que se considera en cada una de las iteraciones del *simulated annealing* es $l = 100$.

5.2. Resolución de ejemplos

En esta sección, se analizan los resultados obtenidos al resolver los ejemplos considerados utilizando el algoritmo diseñado y las soluciones exactas de los mismos.

5.2.1. Comparación de soluciones

En la tabla 5.17, se presentan los resultados obtenidos en la resolución de la batería de ejemplos empleando el método exacto, basado en el problema de programación lineal descrito en el capítulo 3, y el algoritmo implementado en la herramienta informática descrito en el capítulo 4.

ejemplo	*servicios*	*auxiliares*	*días*	*tiempo (exacta)*	*f.o. (exacta)*	*tiempo (alg)*	*f.o. (alg)*
1	6	1	1	6.89	39	0.071	39
2	5	1	1	0.87	40	0.124	40
3	8	1	1	5.38	43	0.045	43
4	8	1	1	58.17	41	0.152	41
5	6	1	1	0.27	42	0.045	42
6	13	1	2	1671.49	164	0.066	164
7	7	1	1	11.02	48	0.039	48
8	4	1	1	0.09	79	0.027	79
9	5	1	1	0.09	44	0.047	44
10	7	1	1	85.11	53	0.083	53
11	6	2	1	98.89	42	0.134	42

Tabla 5.17: Solución exacta y solución aproximada

Elementos de la tabla

La primera columna muestra el indicador del ejemplo considerado; en la segunda, se presenta el número de servicios que forman el ejemplo; la tercera columna, determina el número de auxiliares de

las que se dispone para planificar los servicios considerados y, finalmente, la cuarta columna representa el número de días considerados en el ejemplo.

Por otro lado, las columnas de *tiempo* representan el tiempo de CPU en minutos, empleado en obtener las soluciones a los ejemplos; de modo que, *tiempo (exacta)* muestra el tempo empleado por `Gurobi` en proporcionar la solución exacta y *tiempo (alg)* presenta el tiempo que emplea el algoritmo heurístico, diseñado en este trabajo, en proporcionar la solución aproximada del ejemplo.

Del mismo modo, las columnas *f.o.* presentan el valor de la función objetivo considerada para las soluciones que proporcionan tanto el método exacto como el aproximado. Es decir, *f.o. (exacta)* es el valor de la función objetivo evaluada en la solución obtenida mediante `Gurobi`, mientras que *f.o. (alg)* es el valor de la función objetivo en la solución obtenida utilizando el algoritmo desarrollado.

Resultados obtenidos

En primer lugar, estudiando la columna de los tiempos de ejecución de `Gurobi`, se observa que, aunque todos los ejemplos considerados son muy pequeños en comparación con los casos reales a los que se enfrentan las auxiliares, pueden llegar a tenerse tiempos de computación muy elevados. En concreto, el ejemplo que más ha tardado es el 6, cuyo tiempo de ejecución de 1671 minutos, es decir, 27 horas.

Por otro lado, se tiene que los tiempos de computación del algoritmo diseñado toman valores muy reducidos, siendo el mayor de ellos el de resolución del ejemplo 4, que fue de 0.152 minutos. Lo cual lleva a concluir que este método puede ser utilizado por las coordinadoras, ya que proporciona soluciones de forma prácticamente automática.

Estudiando los valores de la función objetivo que toman las diferentes soluciones, se observa que, en todos los ejemplos, el algoritmo diseñado proporciona planificaciones equivalentes a las óptimas, puesto que, dichos valores coinciden para ambos métodos.

En definitiva, en base a los ejemplos considerados en este experimento, se puede concluir que el algoritmo heurístico es una herramienta muy útil, puesto que proporciona soluciones óptimas empleando tiempos computacionales muy reducidos.

Algunos ejemplos resueltos

A continuación se presentan las planificaciones obtenidas, tanto mediante el método exacto como por el aproximado, de tres de los ejemplos considerados.

servicio	*inicio Disp*	*fin Disp*	*inicio Opt*	*fin Opt*	*sol exacta*	*sol aprox*
1	6:30	17:30	11:00	13:30	15:05 - 15:55	7:05 - 7:55
2	7:00	13:00	9:00	11:00	10:20 - 11:20	8:00 - 9:00
3	7:00	14:30	9:00	12:00	9:15 - 10:15	10:10 - 11:10
4	7:00	13:00	7:00	13:00	8:10 - 9:10	9:05 - 10:05
5	9:00	15:00	12:00	15:00	14:00 - 15:00	11:15 - 12:15
6	16:00	21:00	16:00	21:00	16:00 - 17:00	16:00 - 17:00

Tabla 5.18: Planificación obtenida para el ejemplo 1

El primer ejemplo que se estudia es el 1; para ello, es necesario recalcar el hecho de que ambas soluciones son óptimas. En la tabla 5.18 se presentan los horarios en que se realizan los servicios que forman el ejemplo, según cada una de las soluciones obtenidas. Analizando dichos horarios se pueden determinar aquellos aspectos en los que difieren y se asemejan ambas soluciones.

El primer aspecto de las soluciones que se debe mencionar, es que, en ambas planificaciones solamente hay un descanso, cuya duración es superior a las 2 horas, y por tanto, no forma parte de la

jornada trabajada de la auxiliar. Seguidamente, se tiene que los desplazamientos entre servicios, son de 5 minutos para ambas soluciones, tal y como se determinaba en la definición del ejemplo 1.

La única diferencia que presentan estas planificaciones corresponde a los horarios en que se realizan cada uno de los servicios, en la tabla 5.18 se puede observar que solamente el servicio 6 se realiza a la misma hora en ambas soluciones. Esta diferencia no tiene efecto alguno en la función objetivo ya que no se están teniendo en cuenta las ventanas óptimas, como bien se explicaba anteriormente, y por ello ambas soluciones proporcionan el mismo valor en la función objetivo considerada.

Seguidamente, se considera el ejemplo 11, en el que también las soluciones exacta y aproximada toman el mismo valor en la función objetivo, lo cual quiere decir que la planificación proporcionada por el algoritmo es igual de buena que la solución exacta.

Parámetros iniciales					Solución exacta		Solución aprox	
servicio	*inicio Disp*	*fin Disp*	*inicio Opt*	*fin Opt*	*horario*	*aux*	*horario*	*aux*
1	7:00	13:00	7:00	13:00	11:10 - 12:10	2	8:05 - 9:05	2
2	7:00	13:00	8:00	11:00	10:05 - 11:05	2	9:10 - 10:10	2
3	9:00	11:00	9:00	11:00	9:00 - 10:00	2	9:00 - 10:00	1
4	15:30	19:30	16:00	19:00	18:20 - 19:20	1	16:35 - 17:35	1
5	16:00	21:00	16:00	18:30	20:30 - 21:00	1	16:00 - 16:30	1
6	16:00	21:00	17:00	20:00	19:25 - 20:25	1	17:00 - 18:00	2

Tabla 5.19: Planificación obtenida para el ejemplo 11

En la tabla 5.19 se presentan las planificaciones exacta y aproximada obtenidas para el ejemplo 11, es decir, para cada solución se tiene el horario en que se realiza cada servicio y la auxiliar a la que ha sido asignado.

En primer lugar, se puede observar que todos los servicios han sido asignados, en ambas planificaciones, a las auxiliares que mayor afinidad generan. Para los servicios 3 y 6 se tiene que las auxiliares a las que están asignados en la solución exacta no coinciden con las de la solución aproximada, esto se debe a que ambos servicios tienen el mismo nivel de afinidad con las dos auxiliares disponibles, de modo que las dos posibles asignaciones son óptimas.

Otro aspecto en el que difieren las planificaciones es que, la solución exacta no presenta descansos para ninguna de las auxiliares, en cambio, la solución aproximada tiene un descanso, de duración superior a las 2 horas, para ambas auxiliares.

Por último, mencionar que los desplazamientos entre los servicios son de 5 minutos, tal y como se especificaba en la definición del ejemplo 11.

Todo ello hace que ambas planificaciones, aún con las diferencias que presentan entre ellas, tomen el mismo valor en la función objetivo, es decir, ambas son óptimas.

El último ejemplo que se considera es el 7; en este caso, tal y como sucedía con los ejemplos anteriores, tanto la solución exacta como la aproximada toman el mismo valor en la función objetivo.

En la tabla 5.20, se presentan las planificaciones exacta y aproximada obtenidas para el ejemplo 7. En primer lugar, se observa que ambas planificaciones presentan solamente un descanso, de duración superior las 2 horas, de modo que no forma parte de la jornada de la auxiliar. Por otro lado, se puede observar que ambas soluciones optimizan los desplazamientos, puesto que, el servicio 2 y el 3 se realizan de forma consecutiva (ya que el desplazamiento entre ellos es nulo) y el resto de desplazamientos que se tienen en las planificaciones son de 5 minutos.

servicio	*inicio Disp*	*fin Disp*	*inicio Opt*	*fin Opt*	*sol exacta*	*sol aprox*
1	7:30	18:00	9:00	12:00	8:55 - 9:55	11:55 - 12:55
2	8:00	11:00	9:00	11:00	10:00 - 11:00	9:15 - 10:15
3	9:00	12:00	9:00	12:00	11:00 - 11:30	10:15 - 10:45
4	8:00	13:00	9:00	12:00	11:35 - 12:35	8:10 - 9:10
5	9:00	18:00	9:00	12:00	17:00 - 18:00	10:50 - 11:50
6	7:00	17:00	7:00	13:00	12:40 - 13:40	7:05 - 8:05
7	15:00	22:00	15:00	20:00	15:55 - 16:55	15:00 - 16:00

Tabla 5.20: Planificación obtenida para el ejemplo 7

Aunque las soluciones proporcionen el mismo valor en la función objetivo, éstas no son completamente idénticas, debido a que el horario en que se deben realizar los servicios varía en cada una de las planificaciones. Por ejemplo, en la solución exacta, el servicio 4 se realiza a las 11:25, mientras que, en la solución aproximada, éste se lleva a cabo a las 8:10. Estas diferencias no son significativas puesto que, todos los servicios se realizan dentro de sus ventanas disponibles y, tal y como se explicaba anteriormente, las ventanas óptimas no están siendo consideradas en la función objetivo.

Conclusión

En los ejemplos considerados se observa que las soluciones que proporciona el algoritmo heurístico son equivalentes a las soluciones óptimas, diferenciándose éstas solamente en el horario en que deben realizarse los servicios, es decir, son idénticas en cuanto a minimizar las jornadas de las auxiliares y a maximizar las afinidades. El hecho de que los horarios obtenidos no coincidan no es preocupante, puesto que se tiene gran variedad de posibles soluciones óptimas debido a que las ventanas disponibles de los servicios son muy amplias. Por otro lado, cabe recordar que, en esta batería de ejemplos, no se está considerando el hecho de optimizar las ventanas óptimas de los servicios, de modo que las posibles discrepancias que puedan proporcionar las soluciones en cuanto a ello son irrelevantes.

Un factor muy importante a tener en cuenta son los tiempos de computación invertidos en proporcionar las soluciones. Por un lado, se tiene que el método exacto puede llegar a emplear tiempos muy grandes en resolver algunos de los ejemplos considerados, lo que haría imposible hallar la solución exacta a los problemas reales y, por otro, el algoritmo diseñado siempre emplea tiempos muy reducidos en resolver los ejemplos.

De modo que, mediante la resolución de esta batería de ejemplos, se puede concluir que el algoritmo diseñado para resolver el problema de planificación general de Mayores proporciona soluciones exactas para casos pequeños y, además, lo hace empleando tiempos computacionales mucho menores que el método exacto. Por tanto, para la resolución de problemas reales más complejos, será muy útil utilizar el algoritmo heurístico porque el método exacto no se podría emplear.

Capítulo 6

Aplicación informática

El algoritmo diseñado para resolver el problema de Mayores, explicado en el capítulo 4, se integra en la aplicación informática desarrollada por el equipo del Laboratorio de Bases de Datos de la UDC para la empresa Mayores, de modo que las coordinadoras puedan utilizar este algoritmo para obtener planificaciones de forma automática.

En este capítulo se presenta el funcionamiento de la herramienta de planificación integrada en la aplicación informática desarrollada para Mayores, así como algunos casos de uso del planificador automático (que emplea el algoritmo para resolver las incidencias).

6.1. Datos disponibles

En la aplicación informática se recogen todos los datos de la empresa, pero aquellos que son requeridos por el algoritmo definido en el capítulo 4 son los datos pertenecientes a las auxiliares y a los usuarios.

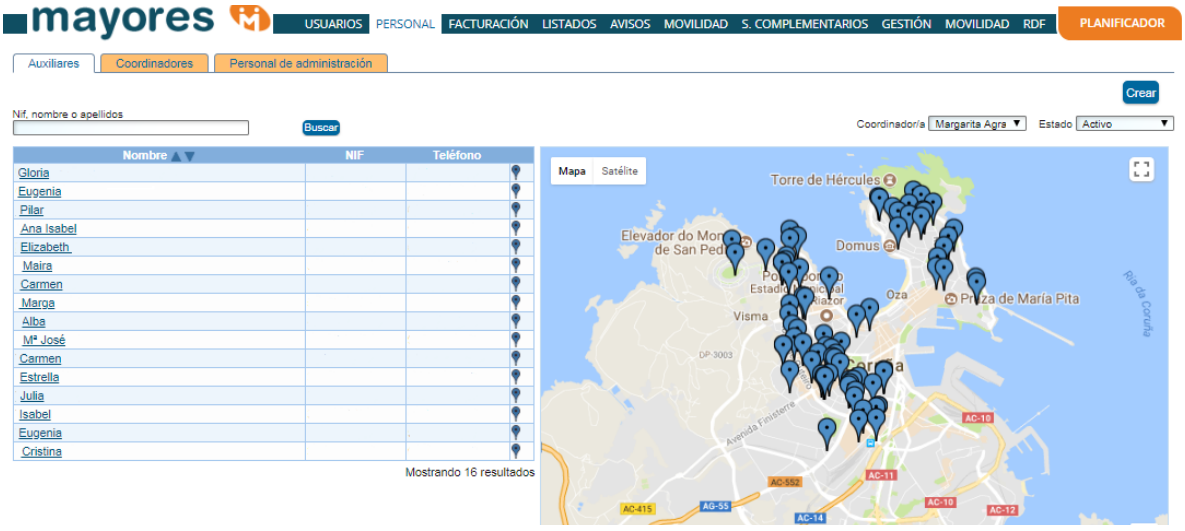


Figura 6.1: Lista de auxiliares de una coordinadora

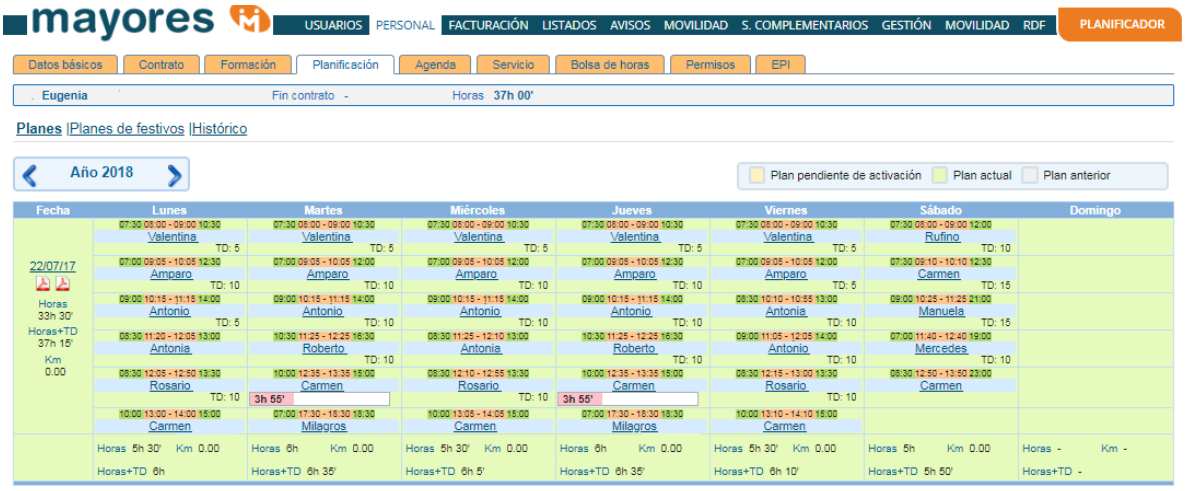


Figura 6.2: Planificación de una auxiliar

6.1.1. Auxiliares

Desde la aplicación se puede acceder a todas auxiliares de cada una de las coordinadoras disponibles. En la figura 6.1 se presenta la lista de auxiliares activas de las que dispone una coordinadora. Todos los datos, nombre completo, NIF, número de teléfono, dirección, etc. se recogen en la aplicación, pero en este trabajo, solamente se muestran los nombres de los usuarios y de las auxiliares.

Para cada auxiliar, es posible obtener la planificación que tiene asignada, es decir, los servicios que debe realizar, el horario en que debe llevarlos a cabo, los tiempos de desplazamiento que emplea en desplazarse entre ellos, y los tiempos de descanso que tiene durante su jornada. La planificación asignada a una auxiliar se presenta en la figura 6.2.

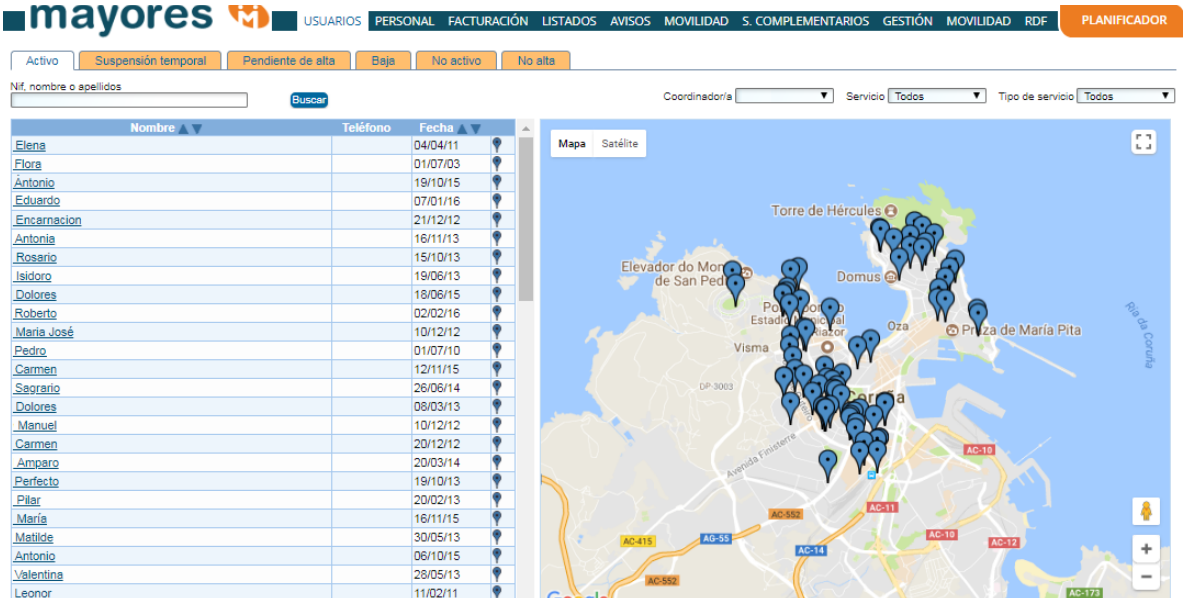


Figura 6.3: Lista de usuarios activos de una auxiliar

6.1.2. Usuarios

Al igual que sucedía con las auxiliares, desde la aplicación es posible acceder a la lista de todos los usuarios asignados a cada coordinadora. En la figura 6.3 se presentan los usuarios activos que están a cargo de la coordinadora seleccionada. Del mismo modo, también se tienen los usuarios cuyos estados son pendiente de alta, suspensión, baja, etc.

Habiendo seleccionado uno de los usuarios, es posible acceder a su plan de intervención. Como ejemplo, en la figura 6.4, se presentan los horarios disponibles y óptimos de los servicios que requiere un usuario. Se observa que, obviamente, las ventanas optimas (que se muestran en color verde) están contenidas en las disponibles (presentadas de color amarillo) para todos los servicios que requiere el usuario. En este caso las ventanas óptimas son estrictamente menores que las disponibles, pero esto no siempre es así, puesto que, en ocasiones, ambas ventanas de tiempo presentan la misma amplitud.

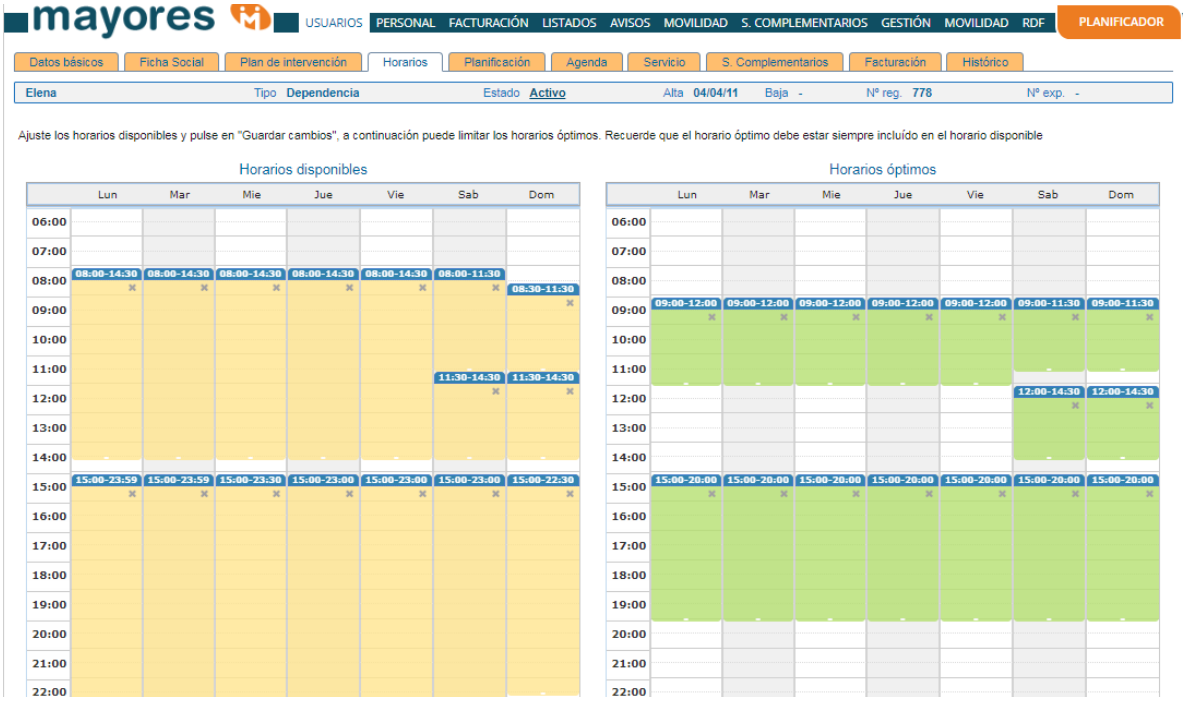


Figura 6.4: Horarios disponible y óptimo de los servicios de un usuario

Del mismo modo, es posible obtener el plan de cada usuario, es decir, el horario en que las auxiliares que lo atienen realizan cada uno de sus servicios y las tareas que deben llevar a cabo durante dichos servicios. Un ejemplo de esto se presenta en la figura 6.5, donde se puede ver la auxiliar, el horario y las tareas que se realizan para atender a los servicios que requiere un usuario. Este usuario requiere dos servicios, durante los días de semana, y tres, durante los fines de semana. Cada uno de estos servicios está asignado a una auxiliar, teniendo fijadas las tareas que ésta deberá llevar a cabo durante la realización de dichos servicios.

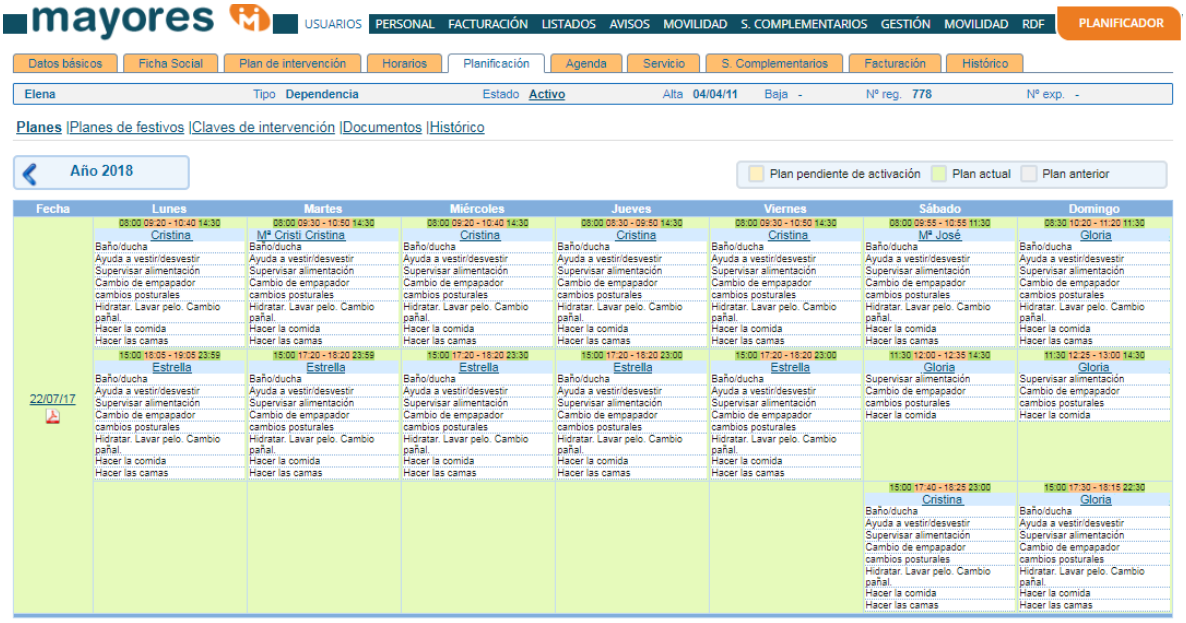


Figura 6.5: Planificación de un usuario

6.2. Funcionamiento del planificador

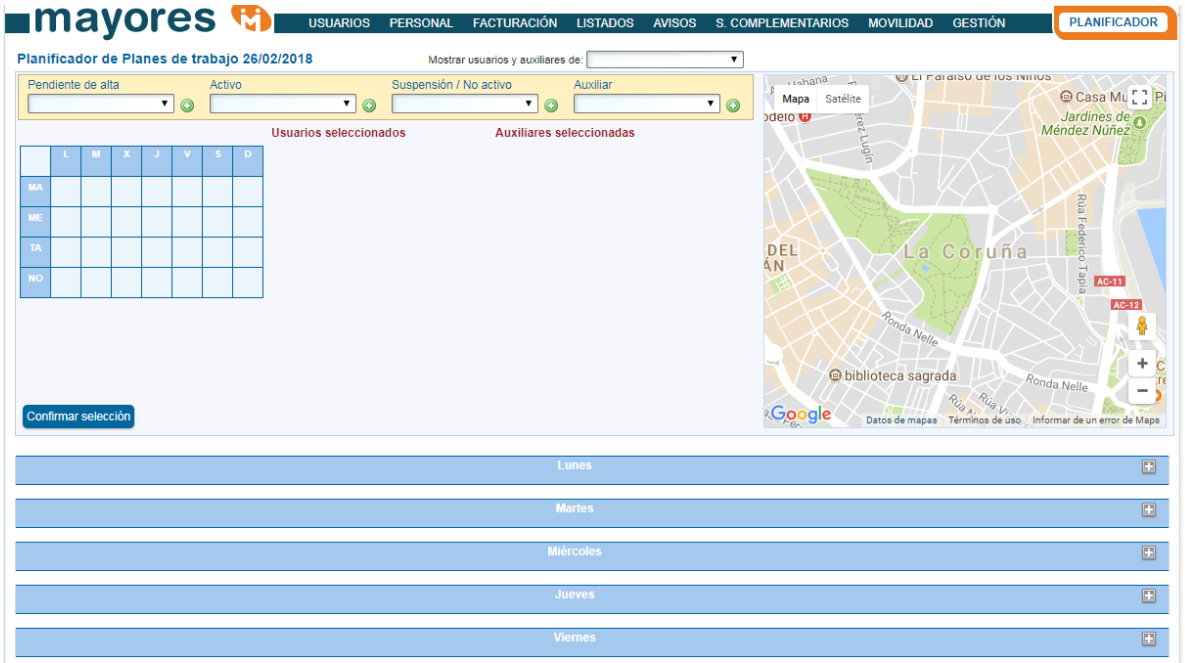


Figura 6.6: Planificador

El planificador es la herramienta utilizada para crear y modificar los planes de las auxiliares, en la figura 6.6 se muestra una imagen del planificador antes de haber comenzado a trabajar con él. Se puede observar que, en la parte de color amarillo, se tienen varios desplegables, en los cuales se encuentran las diferentes auxiliares, usuarios y coordinadoras con las que se puede trabajar. En la derecha del planificador se muestra un mapa, que se utiliza para presentar las localizaciones de las viviendas de los usuarios. Por último, en la parte baja de la pantalla, se presentan los días de la semana, en los cuales se mostrará la planificación de las auxiliares para dichos días.

6.2.1. Elementos para la selección

Estos elementos son aquellos que puede seleccionar la coordinadora de forma manual cuanto esté trabajando con una planificación.

Coordinadora

En este desplegable, mostrado en la figura 6.7, se tiene una lista con todas las coordinadoras de la empresa, de modo que es posible elegir con cuál de ellas se quiere trabajar, ya que, en ocasiones, una coordinadora está cubriendo a otra y debe realizar planificaciones para sus auxiliares y usuarios.

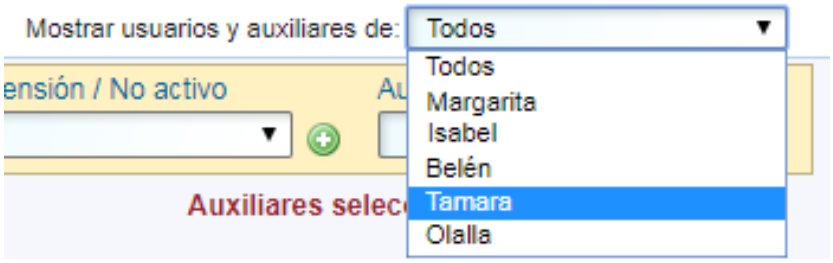


Figura 6.7: Selección de coordinadora

Pendiente de alta

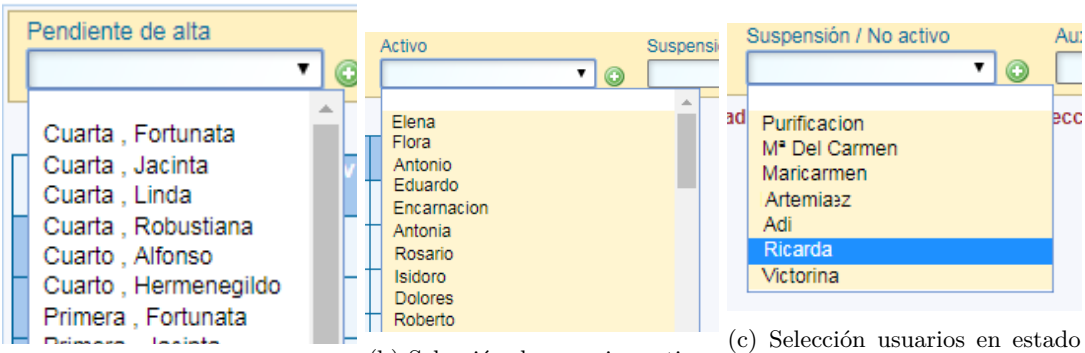
En la figura 6.8a, se muestra un desplegable en el cual se presentan todos los usuarios que están a cargo de la coordinadora selecionada, y cuyo estado es “pendiente de alta”, de modo que es posible seleccionar aquellos que se quieran añadir al sistema.

Activo

En el siguiente desplegable, figura 6.8b, se muestran todos los usuarios “activos” de la coordinadora seleccionada y que son elegibles para modificar su planificación. En este caso, al seleccionar un usuario, también se añade a la planificación la auxiliar (o auxiliares) que lo atiende, así como todos los usuarios a los que también atiende dicha auxiliar (o auxiliares).

Suspensión/No activo

En este desplegable, que se presenta en la figura 6.8c, se tienen los usuarios en estado de “suspensión” o “no activos” a cargo de la coordinadora; de modo que se pueden seleccionar para planificarlos en caso de que se quieran volver a activar. Notar que en, este caso, también es posible que se añada al planificador la auxiliar que lo atendía la última vez que el usuario estuvo activo, así como los usuarios a los que atiende actualmente dicha auxiliar.



(a) Selección de usuarios pendientes de alta

(b) Selección de usuarios activos

(c) Selección usuarios en estado de suspensión o no activos

Figura 6.8: Usuarios que se pueden seleccionar

Auxiliar

En este desplegable, figura 6.9, se presentan todas las auxiliares a cargo de la coordinadora seleccionada. De modo que, será posible escoger las auxiliares con las que se quiere trabajar. Es necesario tener en cuenta que, al seleccionar una auxiliar, también se añaden al planificador todos los usuarios a los que atiende.

La coordinadora puede seleccionar el conjunto de auxiliares con las que quiere trabajar (es decir, las auxiliares a las que quiere modificar la planificación, o aquellas que quiere tener en cuenta a la hora de planificar nuevos servicios), pero el algoritmo propuesto permite también que no se seleccione ninguna auxiliar, en cuyo caso se considerarán todas las auxiliares de las que dispone la coordinadora. En los ejemplos que se presentan a continuación se ilustran, tanto casos en los cuales se considera que la coordinadora ha seleccionado el conjunto de auxiliares con los que desea trabajar, como casos en los cuales no se selecciona nigua auxiliar, y es el sistema el encargado de determinar cuál es la mejor auxiliar (o auxiliares) para solventar la incidencia considerada.

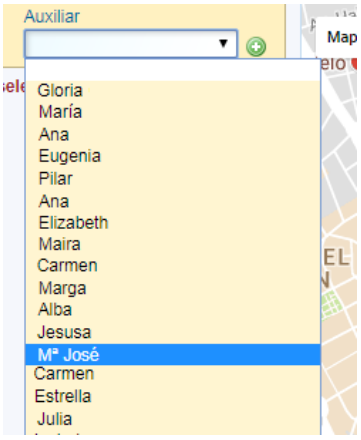


Figura 6.9: Selección de auxiliares

6.2.2. Elementos seleccionados

A continuación se resumen los elementos que han sido seleccionados por la coordinadora, así como los nuevos aspectos que presenta el planificador cuando se trabaja con los nuevos usuarios y auxiliares.

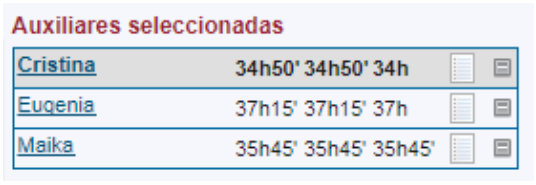
Usuarios seleccionados

En esta lista, presentada en figura 6.10a, se tienen todos los usuarios que han sido seleccionados, en negrita se tienen los usuarios elegidos manualmente y en letra normal aquellos que se han añadido de forma automática (por ejemplo, al seleccionar auxiliares).

Auxiliares seleccionadas

En esta lista, que se presenta en la figura 6.10b, se muestran todas las auxiliares que han sido seleccionadas; en negrita se muestran las auxiliares que se han seleccionado de forma manual y en letra normal aquellas que se han añadido de forma automática (por ejemplo: al seleccionar un usuario que está siendo atendido por una auxiliar). Para cada auxiliar se muestra el número de horas trabajadas semanalmente, así como el número de horas que se especifica en su contrato, para así poder comparar ambas cantidades.

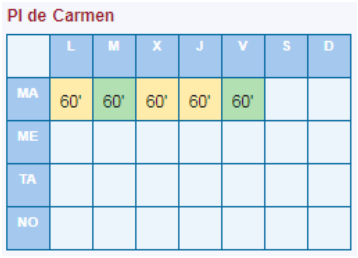
(a) Lista con los usuarios seleccionados



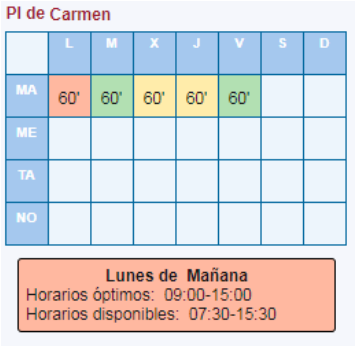
(b) Lista con las auxiliares seleccionadas

Figura 6.10: Elementos que han sido seleccionados

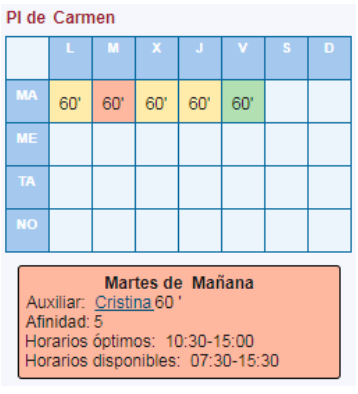
Plan de intervención



(a) Tramos



(b) Tramo sin asignar



(c) Tramo asignado

Figura 6.11: Tramos que requiere un usuario

En el cuadro mostrado en la figura 6.11a, se presenta el plan de intervención del usuario con el que se desee trabajar (para ello basta con seleccionar un usuario de la lista de los seleccionados). Si los

tramos se encuentran de color amarillo significa que no están planificados, es decir, no están asignados a una auxiliar y no tienen un horario de realización definido; en cambio, que los tramos estén en color verde significa que están planificados.

Al seleccionar alguno de los tramos, éstos se muestran de color salmón. En la figura 6.11b se muestran el horario óptimo y el disponible definidos para un tramo que no está planificado. Del mismo modo, en la figura 6.11c se presenta un tramo que sí está planificado, especificando su duración, la auxiliar a la que está asignado, y la afinidad entre la auxiliar y el usuario.

Mapa

En el mapa presentado en la figura 6.12 se muestra la ubicación de las viviendas de los usuarios seleccionados.

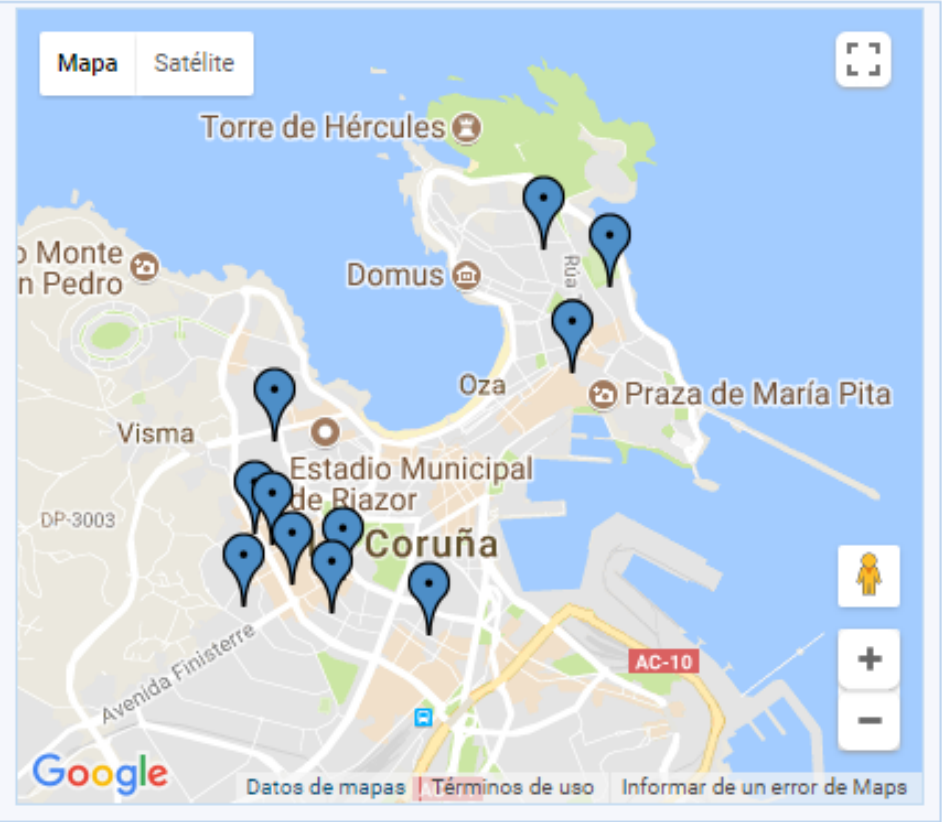


Figura 6.12: Localización de los usuarios seleccionados

6.2.3. Planificación

A continuación, en la figura 6.13, se presenta la planificación para un día de la semana de las diferentes auxiliares seleccionadas. Se muestran las auxiliares y los servicios que tienen planificados. Para cada uno de ellos se tiene el horario en que el servicio se va a realizar, así como su ventana disponible. Entre cada par de servicios consecutivos se muestra el tiempo de desplazamiento que se emplea en ir de la vivienda del primero hasta la del segundo y, en los casos en que se tenga un descanso en la planificación de la auxiliar, no se muestra el tiempo de desplazamiento, sino que se presenta el tiempo de descanso marcado en color salmón.

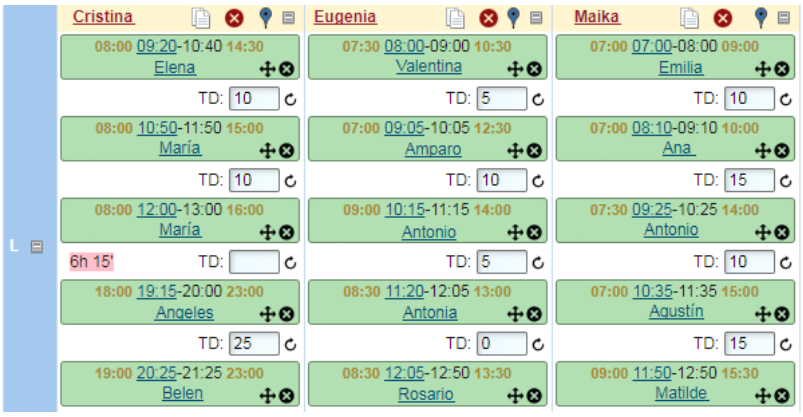


Figura 6.13: Planificación para un día de la semana

En la aplicación, figura 6.14, se muestran tres botones cuyo funcionamiento es el siguiente.

Replanificar. Sirve para obtener la replanificación automática.

Guardar cambios. Sirve para guardar todos los cambios que se hayan hecho en la planificación de las auxiliares.

Guardar y confirmar. Sirve para guardar los cambios y confirmar la planificación.

Figura 6.14: Botones del replanificador

Antes de realizar la replanificación automática, el sistema muestra un formulario, presentado en la figura 6.15, que indica las horas estipuladas en los contratos de las auxiliares con las que se está trabajando, así como unas casillas en las que se puede especificar las horas máximas que se permitirían para cada auxiliar, es decir, las horas hasta las cuales se podría ampliar el contrato de las auxiliares.

Figura 6.15: Formulario de ampliación de la jornada máxima permitida

Al definir un máximo permitido, el sistema puede proporcionar planificaciones cuya duración difiera de las horas contratadas de cada auxiliar. Es por esto por lo que, aunque el sistema no pueda modificar los contratos de las auxiliares, puede proponer las alteraciones que deberían llevarse a cabo.

6.3. Casos de uso

A continuación, se muestran algunos ejemplos en los cuales se utiliza el planificador automático, que emplea el algoritmo definido en el capítulo 4, para resolver las incidencias descritas en 1.3.2.

6.3.1. Alta de un nuevo usuario

Este ejemplo consiste en dar de alta a un nuevo usuario y planificar sus servicios.

Supóngase que se quiere dar de alta a un nuevo usuario, Robustiana. Para poder hacerlo correctamente es necesario tener planeados todos sus servicios, de modo que, cada uno de ellos debe asignarse a una auxiliar especificando el horario en que debe realizarse.

Planificación inicial

Tras seleccionar el usuario que se quiere dar de alta, se selecciona aquella auxiliar que quiere utilizar para atenderlo. Dando lugar a la planificación inicial mostrada en la figura 6.16.

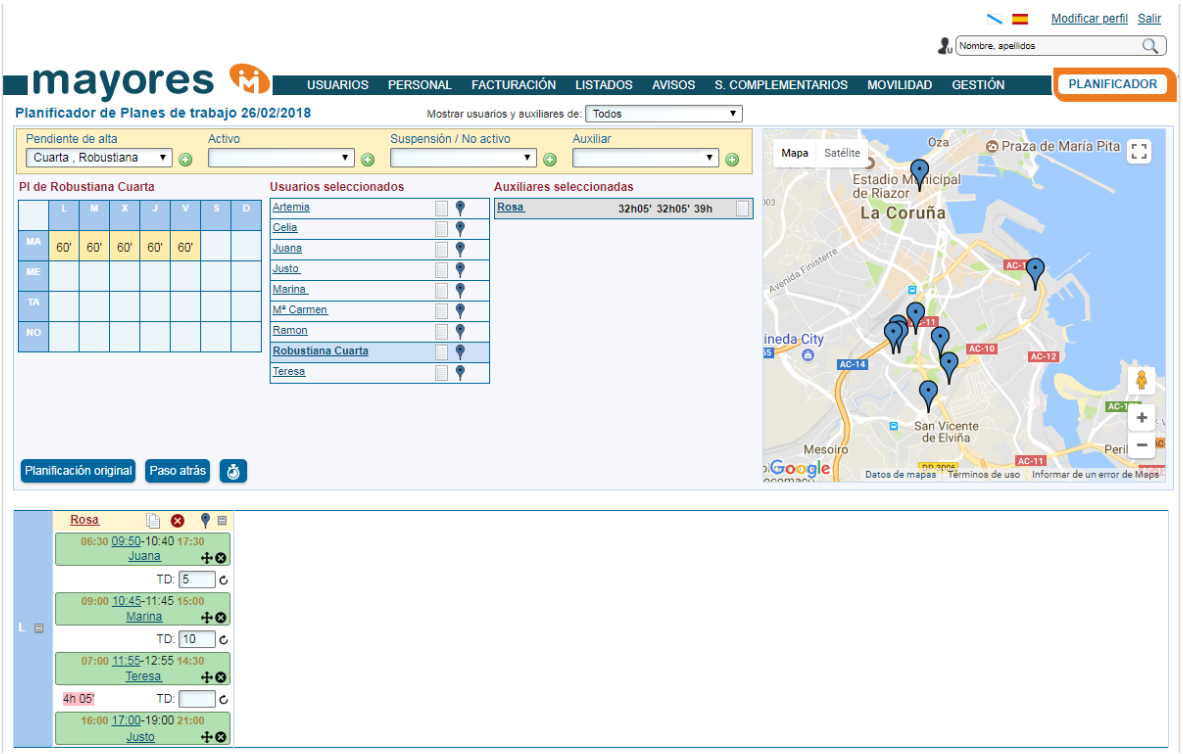


Figura 6.16: Planificación inicial para dar de alta a un usuario

En esta planificación solamente se muestra lunes por motivos de espacio, pero en la aplicación haciendo *scroll* hacia abajo se tiene la planificación de todos los días de la semana.

Se observa que la auxiliar solamente tiene planificados cuatro servicios para la jornada del lunes, y que el usuario a dar de alta no tiene ninguno de sus servicios planificados (ya que en el plan de intervención de Robustiana todos sus servicios se muestran de color amarillo).

Replanificación automática

Debido a que se quiere obtener la replanificación automática de los servicios del nuevo usuario, es necesario completar el formulario en el cual se pueden definir las horas máximas hasta las que se podría ampliar el contrato de la auxiliar. En este caso, figura 6.17, no se permitirá aumentar las horas de contrato de la auxiliar, puesto que la planificación inicial proporciona una holgura en la jornada semanal de casi 7 horas, como puede verse en la figura 6.16.

Figura 6.17: Formulario para ampliar jornadas máximas al dar de alta a un usuario

Planificación final

Tras generar la replanificación automática se obtiene la planificación presentada en la figura 6.18, en la cual los servicios del nuevo usuario han sido asignados a la auxiliar y se han determinado los horarios en que deben realizarse.

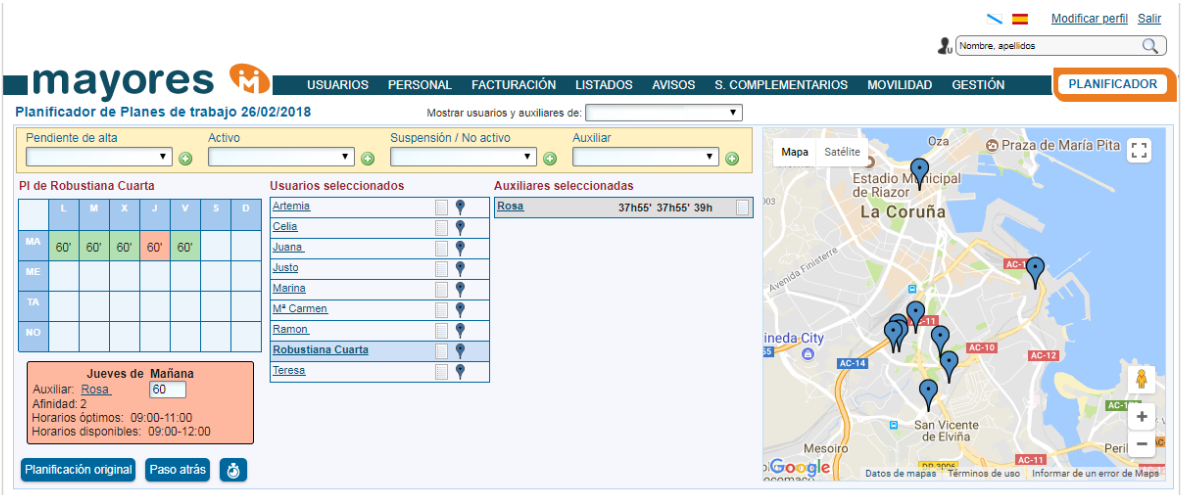
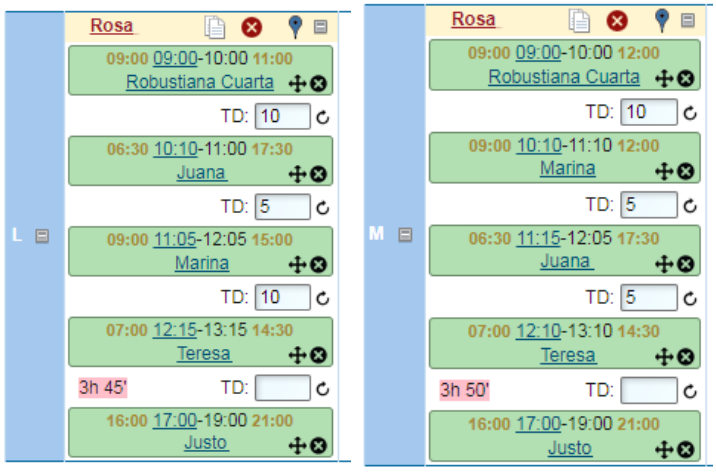


Figura 6.18: Planificación final para dar de alta un usuario

En el plan de intervención de Robustiana puede verse que los servicios aparecen de color verde; esto quiere decir que han sido asignados a la auxiliar y que se ha determinado el horario en que deben llevarse a cabo. Seleccionando el servicio del jueves se comprueba que ha sido asignado a Rosa Bravo y que el nivel de afinidad con esta auxiliar es 2; puesto que la auxiliar no había atendido nunca al usuario.

Por otro lado, se observa que la nueva planificación tiene una duración de 38 horas y 5 minutos, de modo que se respeta su contrato de 39 horas semanales.



(a) Planificación del lunes (b) Planificación del martes

Figura 6.19: Planificación final tras asignar los servicios del usuario que se da de alta

En la planificación del lunes, figura 6.19a, el servicio de Robustiana se añade al principio de la planificación, de tal forma que su horario coincide con el inicio de su ventana disponible. Observando la planificación inicial de la auxiliar, se puede comprobar que el servicio de Juana comenzaba a las 9:50 y ahora comienza a las 10:10; esto se debe a que es necesario retrasar los servicios para poder colocar el de Robustiana.

Del mismo modo, en la planificación del martes, que se presenta en la figura 6.19b, se observa que el servicio de Robustiana se añade al principio de la planificación, al igual que sucedía con la del lunes. La diferencia en este caso es que se intercambia el orden de los servicios de Juana y Marina. Este intercambio se debe a que es necesario retrasar los servicios para poder colocar correctamente el servicio de Robustiana, pero el de Marina no puede retrasarse todo lo necesario, en cambio, el servicio de Juana sí que puede retrasarse lo suficiente como para hacer hueco al nuevo servicio.

Por tanto, el algoritmo ha sido utilizado de forma satisfactoria para planificar los servicios del usuario que quiere darse de alta en el servicio de Mayores.

6.3.2. Baja de un usuario

En este ejemplo se considera el caso en que un usuario que estaba siendo visitado por alguna auxiliar se da de baja de la empresa, de modo que la auxiliar (o auxiliares) que lo atendía ya no debe realizar las visitas al usuario y, por tanto, es necesario eliminar esos servicios de su planificación. Al hacer esto es posible que se generen huecos en la jornada de la auxiliar; en estas ocasiones se optimiza la planificación que realiza la auxiliar para tratar de eliminar dichos huecos.

Planificación inicial

La planificación inicial de la auxiliar seleccionada se muestra en la figura 6.20. Observando el plan de intervención de Concepción es posible comprobar que los servicios que requiere están correctamente planificados, ya que se muestran de color verde. En concreto se puede ver ver que, seleccionando el servicio del miércoles, se muestran sus horarios óptimos y disponibles, así como la auxiliar a la que está asignado dicho servicio y el nivel de afinidad entre el usuario y la auxiliar (en este caso dicho nivel es 5 puesto que la auxiliar es la que está atendiendo actualmente, de forma satisfactoria, al usuario).

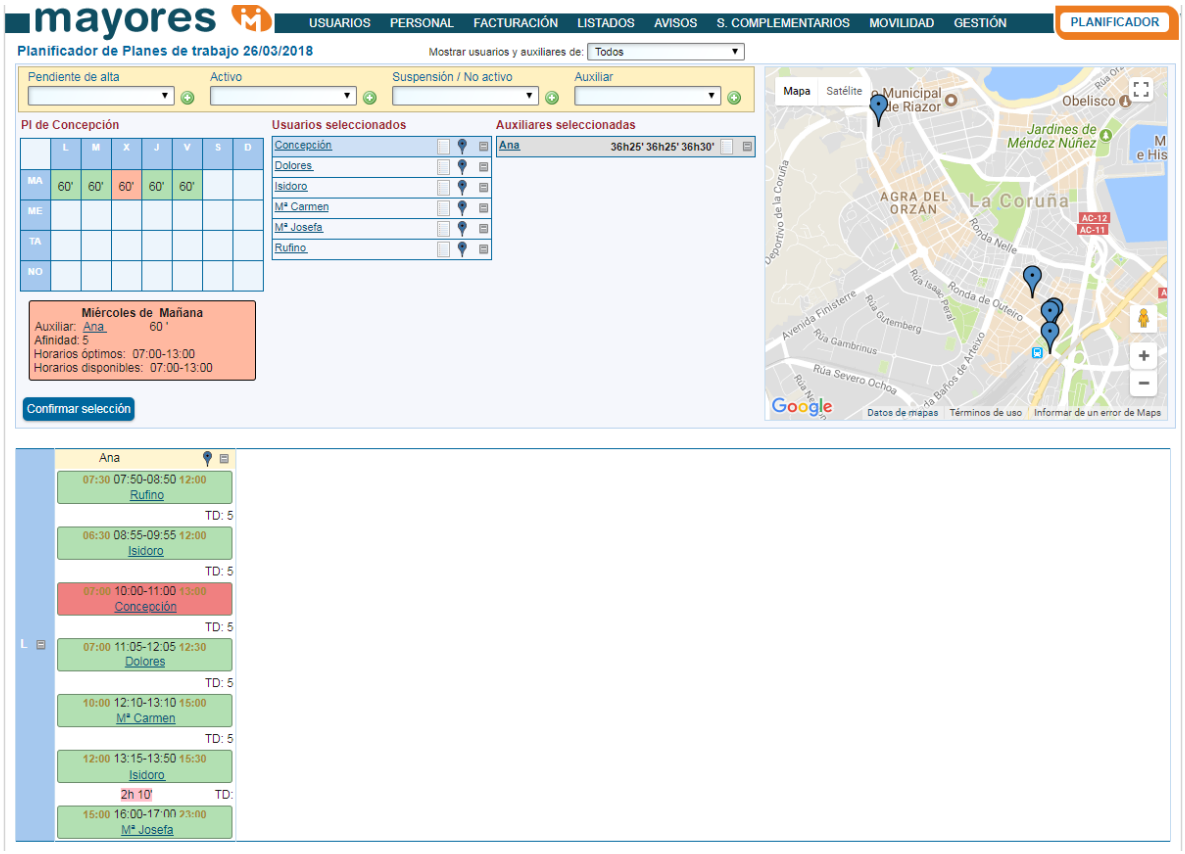


Figura 6.20: Planificación inicial para dar de baja un usuario

En esta planificación se puede observar que hay un servicio en color salmón; éste se corresponde con el del lunes de Concepción, el usuario que se desea dar de baja. Este servicio se eliminará provocando un hueco en la jornada de la auxiliar.

Replanificación automática

Figura 6.21: Formulario de ampliación de la jornada de las auxiliares para dar de baja un usuario

En este caso, como simplemente se quieren eliminar servicios, no se amplía el número de horas máximas permitidas, tal y como se muestra en la figura 6.21.

Planificación final

Tras generar la replanificación automática se obtiene una planificación, figura 6.22, en la cual se han eliminado todos los servicios asignados al usuario que se daba de baja; el sistema también ha conseguido eliminar los huecos que esto provocaba en la jornada de la auxiliar.

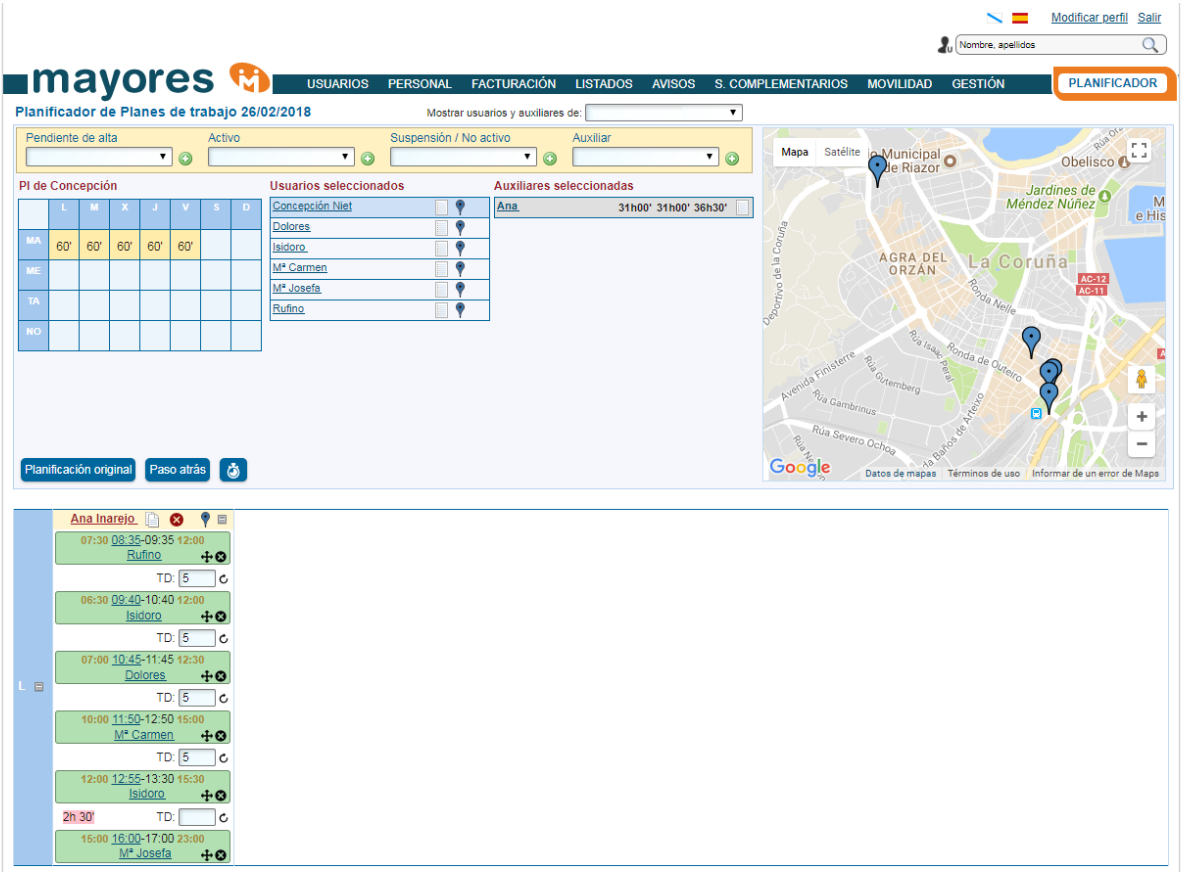


Figura 6.22: Planificación final para dar de baja un usuario

6.3.3. Alta de un usuario y baja de otro

Como se ha explicado en ejemplos anteriores, es posible dar de alta a usuarios y también darlos de baja. Por tanto, puede darse el caso de que se quiera aprovechar que un usuario ya no deba ser atendido para planificar nuevos servicios. Es decir, dar de baja a un usuario y dar de alta a otro diferente en la misma replanificación.

Planificación inicial

En el plan de intervención de Alfonso, usuario que se quiere dar de alta, que se presenta en la figura 6.23, se puede observar que este nuevo usuario requiere cinco servicios en jornada de mañana, con una duración de 60 minutos cada uno. También es posible comprobar que, en la planificación inicial de la auxiliar para el lunes, hay un servicio que se muestra en color salmón; éste es el servicio del usuario que se desea dar de baja, Concepción. Dicho servicio se debe eliminar de la planificación provocando un hueco en la jornada de la auxiliar.

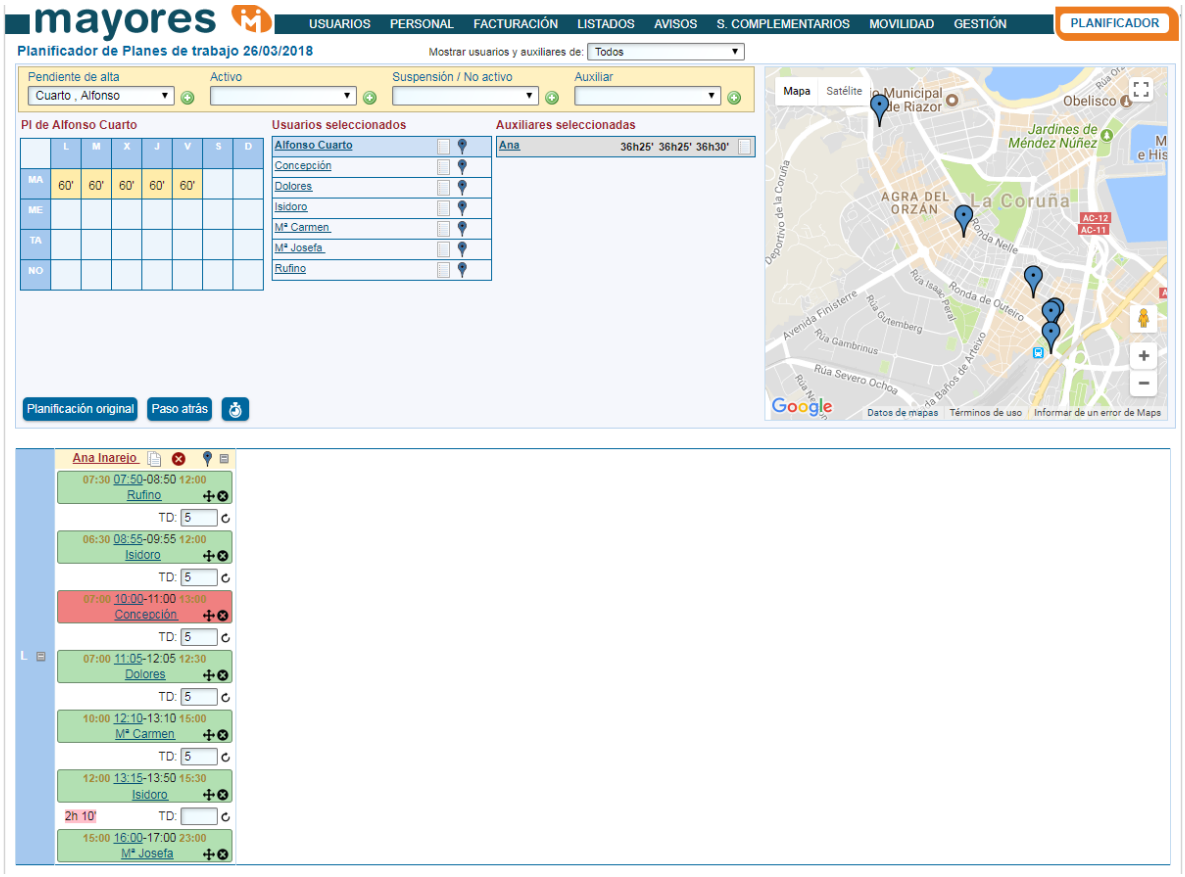


Figura 6.23: Planificación inicial en el caso en que se quiere dar de baja un usuario y de alta otro

En la lista de auxiliares seleccionadas, se puede ver que la jornada semanal contratada de la auxiliar es de 36 horas y 30 minutos, y la duración de la planificación inicial es igual a 36 horas y 25 minutos, de modo que se respeta el número de horas semanales del contrato de la auxiliar.

Replanificación automática

Figura 6.24: Formulario de ampliación de la jornada para dar de baja un usuario y dar de alta otro

En este caso se decide aumentar la jornada máxima permitida a 40 horas, ya que la planificación inicial estaba muy ajustada respecto a la jornada contratada de la auxiliar, como se presenta en la figura 6.24.

Planificación final

Tras generar la replanificación automática, figura 6.25, se obtiene una planificación en la cual los servicios del nuevo usuario han sido asignados a la auxiliar y se han eliminado los del usuario que se deseaba dar de baja.

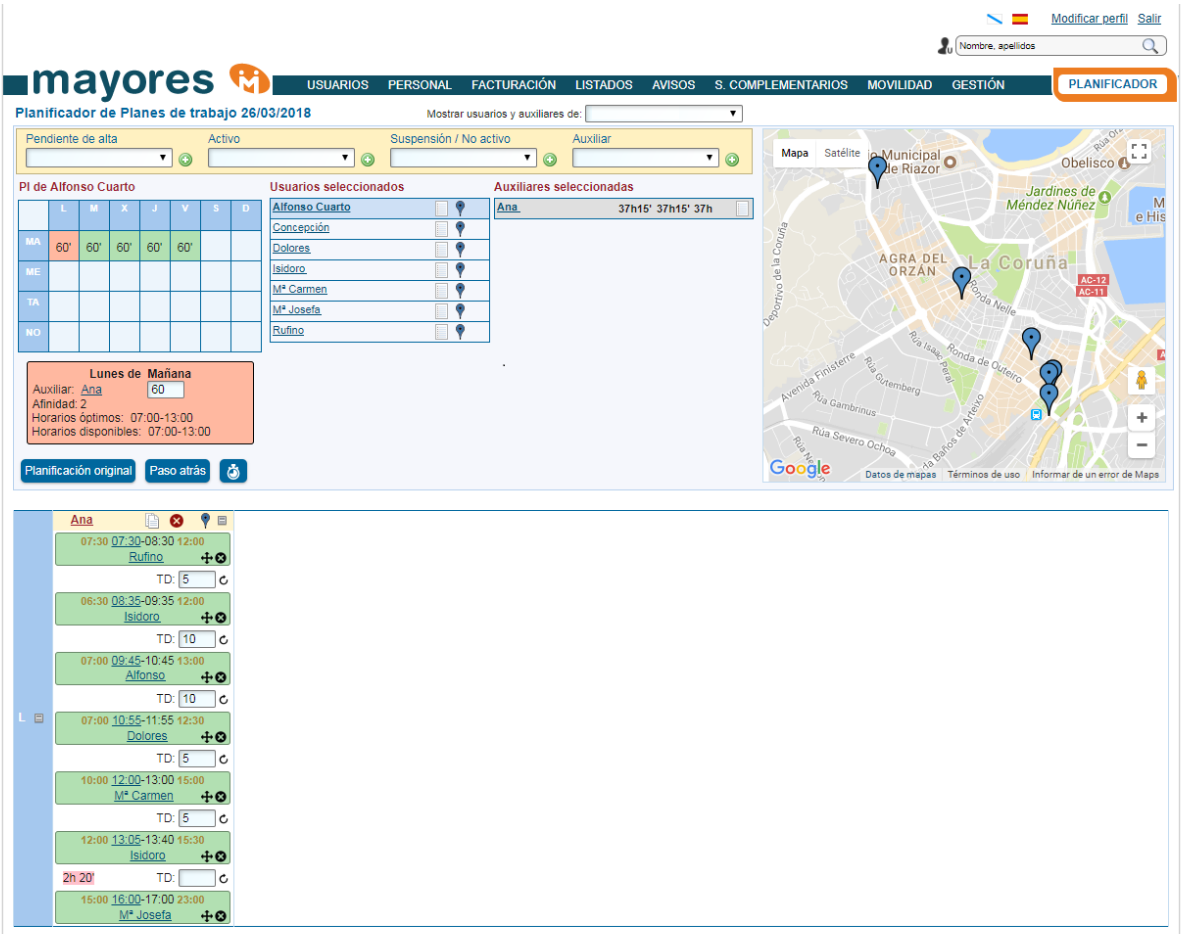


Figura 6.25: Planificación final para dar de alta un usuario y de baja otro

En el plan de intervención de Alfonso se comprueba que los servicios aparecen de color verde; esto quiere decir que han sido asignados a la auxiliar, y que se ha determinado el horario en que deben llevarse a cabo. Seleccionando el servicio del lunes se observa que ha sido asignado a Ana y que el nivel de afinidad con esta auxiliar es 2, puesto que la auxiliar no había atendido nunca a este usuario.

Por otro lado, es necesario mencionar que, tras eliminar los servicios del usuario que se da de baja y añadir los servicios del nuevo usuario, la auxiliar ahora tiene una jornada semanal planeada de 37 horas y 15 minutos, de modo que no respeta su contrato de 36 horas y 30, pero si respeta el máximo permitido de 40 horas.

En esta nueva planificación para el lunes se puede comprobar que los servicios del nuevo usuario (Alfonso) se han colocado en el hueco que se obtenía al eliminar los correspondientes al usuario que se quiere dar de baja (Concepción). Pero, aun utilizando dichos huecos, es necesario modificar los horarios en que se realizan los servicios anteriores y siguientes al nuevo servicio; esto se debe a que los tiempos de desplazamiento con el nuevo usuario no son iguales a los que se tenían con el usuario al que se de

baja. De modo que, esta diferencia en los desplazamientos es la que provoca que la jornada planificada de la auxiliar aumente con respecto a la que proporcionaba su planificación inicial.

Por tanto, el algoritmo diseñado ha servido para eliminar los servicios del usuario que se da de baja y planificar aquellos que requiere el usuario al que se da de alta.

6.3.4. Ampliar planificación

En este ejemplo se considera el caso en que un usuario que ya está dado de alta (es decir, que ya está siendo atendido por una o varias auxiliares) decide ampliar el número de servicios que requiere; de modo que será necesario planificar estos nuevos servicios.

Planificación inicial

La planificación inicial de la auxiliar, que atiende actualmente al usuario que desea ampliar el número de servicios que requiere, se muestra en la figura 6.26.

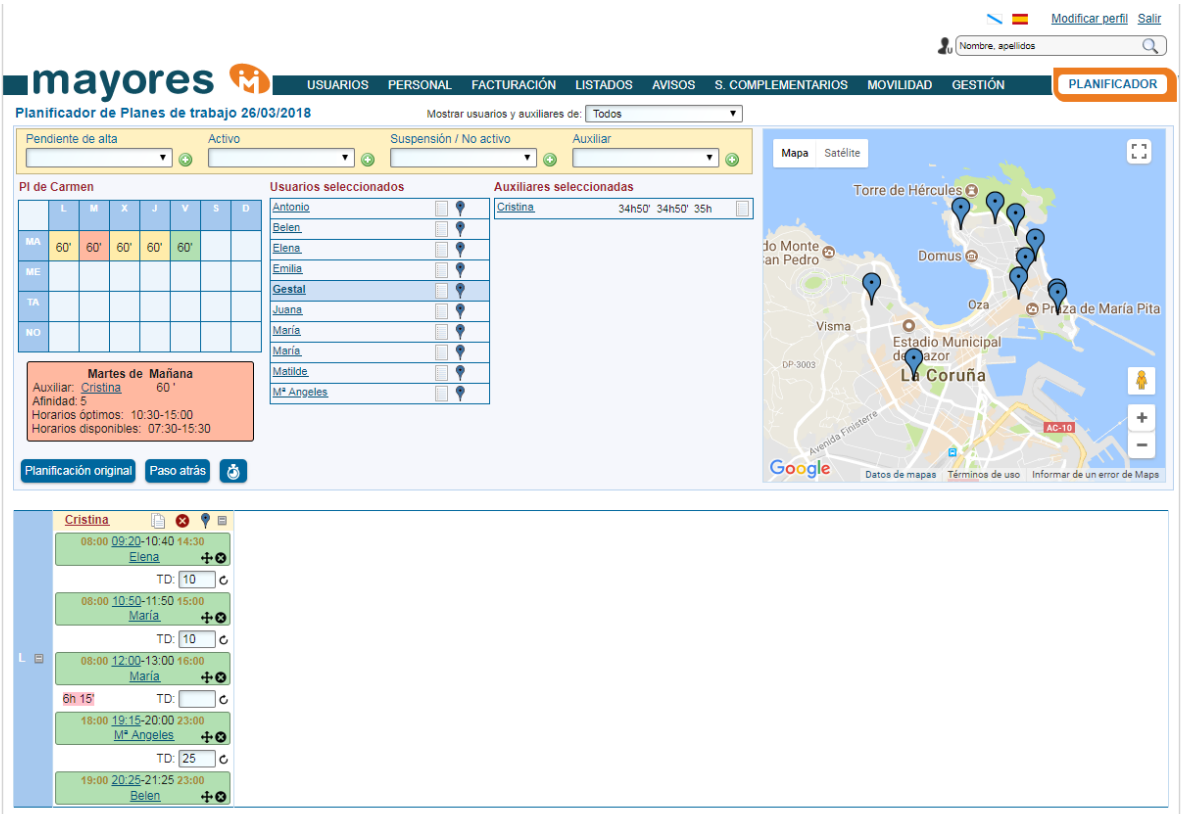


Figura 6.26: Planificación inicial para el caso en que se amplía el número de servicios de un usuario

Observando el plan de intervención de Carmen se comprueba que requiere cinco servicios, en tramo de mañana y de duración 60 minutos cada uno. De estos servicios se tienen dos ya planificados, los del martes y el viernes (color verde), y tres servicios sin planificar, los del lunes, miércoles y jueves (color amarillo). En concreto se puede ver que, seleccionando el servicio del martes, se muestran sus horarios óptimos y disponibles, así como la auxiliar a la que está asignado dicho servicio y el nivel de afinidad entre el Carmen y la auxiliar (en este caso el nivel es 5 puesto que la auxiliar es la que está atendiendo actualmente al usuario).

En la lista de auxiliares seleccionadas se tiene que la planificación actual de la auxiliar es de 34 horas y 50 minutos de trabajo, lo cual no supera su jornada contratada, de 35 horas.

Replanificación automática

En este caso se decide aumentar la jornada máxima permitida de la auxiliar a 40 horas, puesto que su planificación inicial está bastante ajustada al contrato de la auxiliar y se desean añadir varios servicios a ésta, como muestra la figura 6.27.

Figura 6.27: Formulario de ampliación de la jornada máxima para aumentar los servicios de un usuario

Planificación final

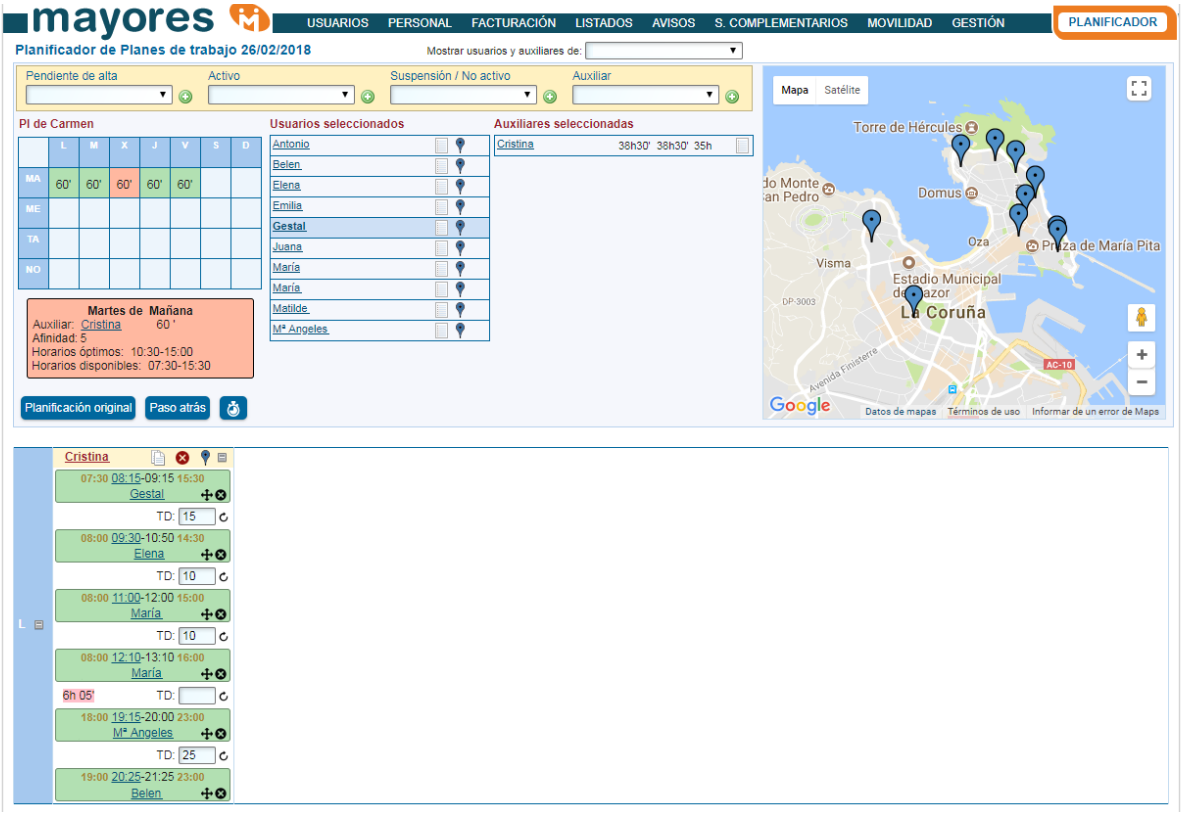


Figura 6.28: Planificación final para el caso en que se amplía el número de servicios de un usuario

En la figura 6.28 se puede observar el plan de intervención de Carmen, en el que se comprueba que todos sus servicios están planificados, es decir, los servicios de los días lunes, miércoles y jueves se han asignado a la auxiliar y se ha determinado el horario en que deben realizarse.

Seleccionando el servicio del miércoles se observa que ha sido asignado a Ana y que el nivel de afinidad con ella es 5; esto se debe a que la auxiliar es la que está atendiendo actualmente al usuario.

Por otro lado, en la lista de auxiliares seleccionadas, se puede observar que la nueva planificación de la auxiliar tiene una duración de 38 horas y 30 minutos, lo que supera su jornada semanal contratada de 34 horas, pero respeta el máximo permitido de 40 horas.

6.3.5. Baja de un usuario y ampliación de planificación

En los ejemplos anteriores se han explicado los casos en que se da de baja un usuario y aquellos en que se amplía el número de servicios que requiere un cliente, pero se puede aprovechar que hay que eliminar los servicios del usuario que se da de baja para planificar los nuevos servicios de otro que amplía su planificación. Éste es el caso que se ilustra a continuación.

Planificación inicial

En el plan de intervención del usuario María, figura 6.29, se puede observar que requiere 5 servicios, en franja mañana y con una duración de 60 minutos cada uno. De estos servicios, se tiene que los servicios del lunes y del viernes ya están correctamente planificados (se muestran de color verde), y que los del martes, miércoles y jueves no lo están (se muestran de color amarillo), siendo estos últimos los servicios que el sistema debe planificar. En concreto, se puede ver que, seleccionando el servicio del lunes, se presentan los horarios óptimos y disponibles del mismo, así como, la auxiliar a la que está asignado dicho servicio, y el nivel de afinidad entre el usuario y dicha auxiliar (en este caso el nivel es 5 puesto que la auxiliar es la que está atendiendo actualmente al usuario).

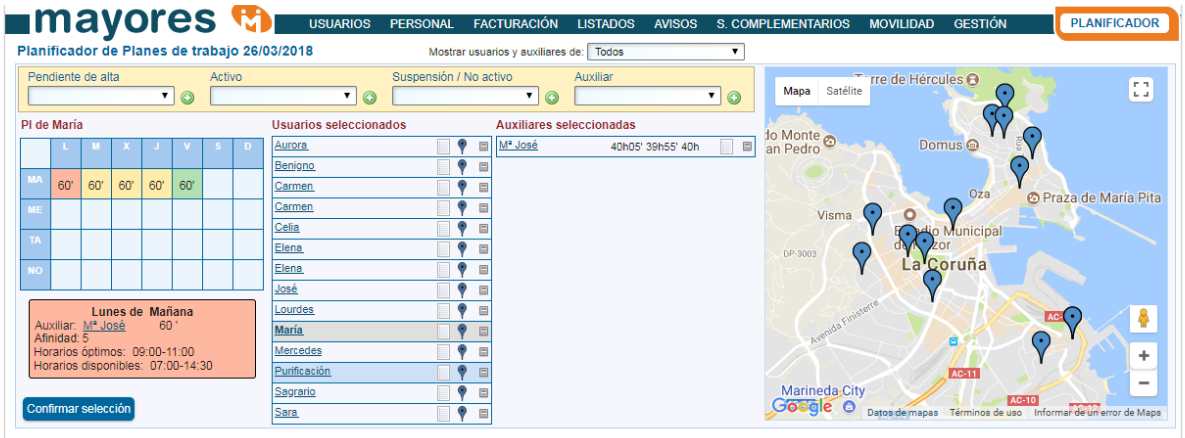


Figura 6.29: Plan de intervención inicial del usuario al que se le aumenta el número de servicios

En la lista de auxiliares seleccionadas se puede ver que Mª José tiene una planificación inicial de 39 horas y 55 minutos, lo cual respeta su jornada semanal contratada de 40 horas.

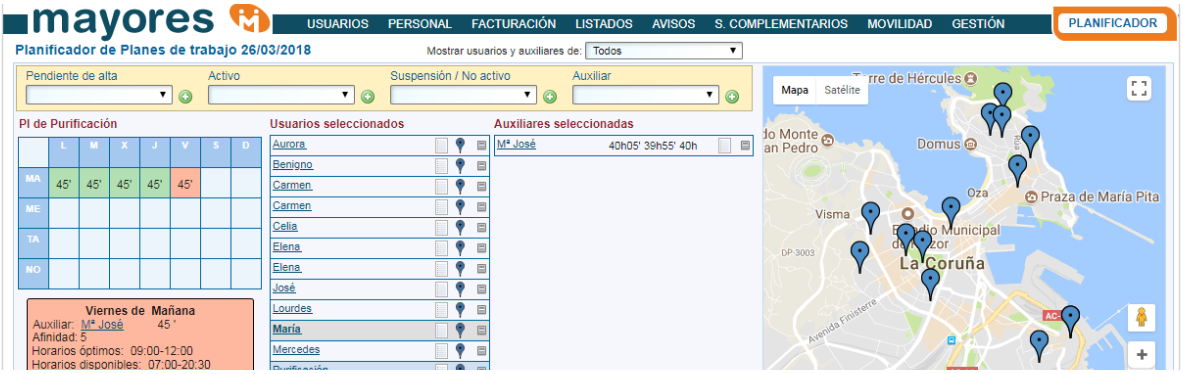


Figura 6.30: Plan de intervención inicial del usuario que se da de baja

Estudiando el plan de intervención de Purificación, presentado en la figura 6.30, se observa que tiene los cinco servicios, que se deben eliminar de la planificación de la auxiliar, correctamente planificados. En particular se tiene que, seleccionando el servicio del viernes, se muestran los horarios óptimos y disponibles, así como la auxiliar a la que está asignado dicho servicio y el nivel de afinidad entre el usuario y la auxiliar.

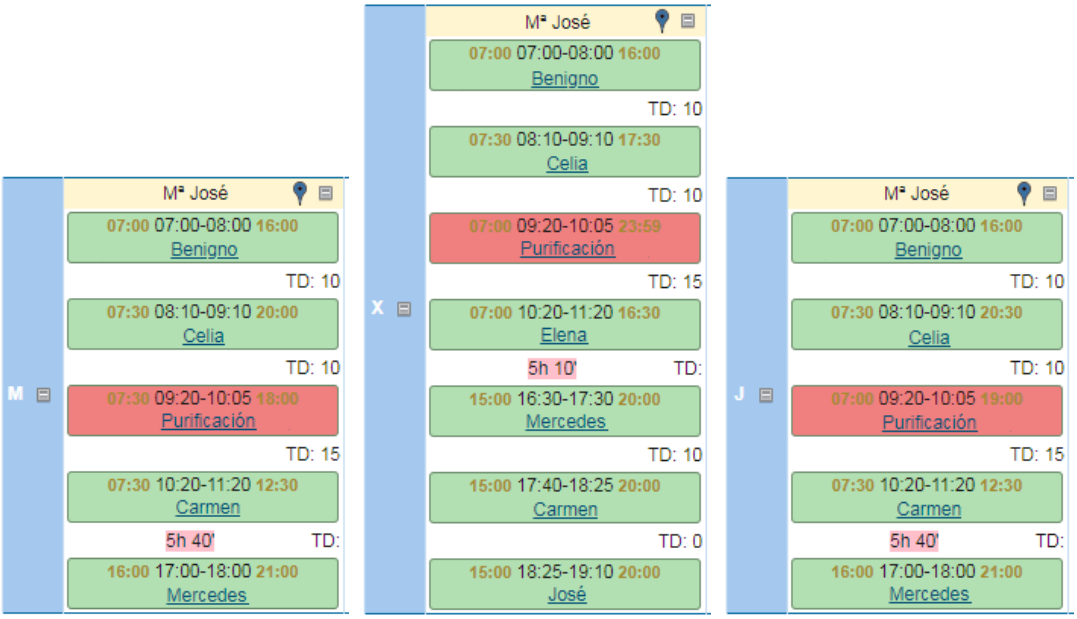


Figura 6.31: Planificación inicial para el martes, miercoles y jueves

La planificación inicial para los días martes, miércoles y jueves (solamente se muestran estos días porque son en los que se añadirán los nuevos servicios) se presenta en la figura 6.31. En ella se observa que, no hay servicios del nuevo usuario (María), y que se tienen servicios representados con un color salmón; dichos servicios se corresponden con los de un usuario al que se da de baja (Purificación). De modo que, para estos días, el sistema debe eliminar los servicios no activos y colocar los del nuevo usuario.

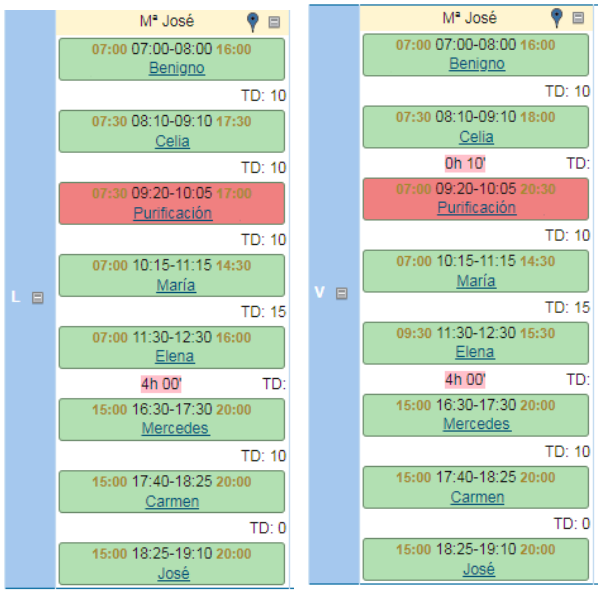


Figura 6.32: Planificación inicial para el lunes y viernes

La planificación del lunes y viernes se presenta en la figura 6.32, en ella se puede observar que se tienen planificados tanto los servicios del usuario al que se quiere ampliar la planificación (María), como los servicios correspondientes al usuario que se desea dar de baja, marcados en color salmón. Por tanto, para estos días solamente es necesario eliminar los servicios del usuario al que se desea a dar de baja (Purificación); al hacerlo se generarán huecos en la planificación que es necesario eliminar.

Replanificación automática

En este caso, como se van a eliminar más servicios de los que se van a añadir a la planificación, no se amplía el número de horas máximas permitidas de la auxiliar, como puede comprobarse en la figura 6.33.

Figura 6.33: Formulario de ampliación de la jornada máxima permitida

Planificación final

Tras generar la replanificación automática se obtiene una planificación, en la cual los nuevos servicios han sido asignados a la auxiliar, y se han eliminado los servicios del usuario no activo.

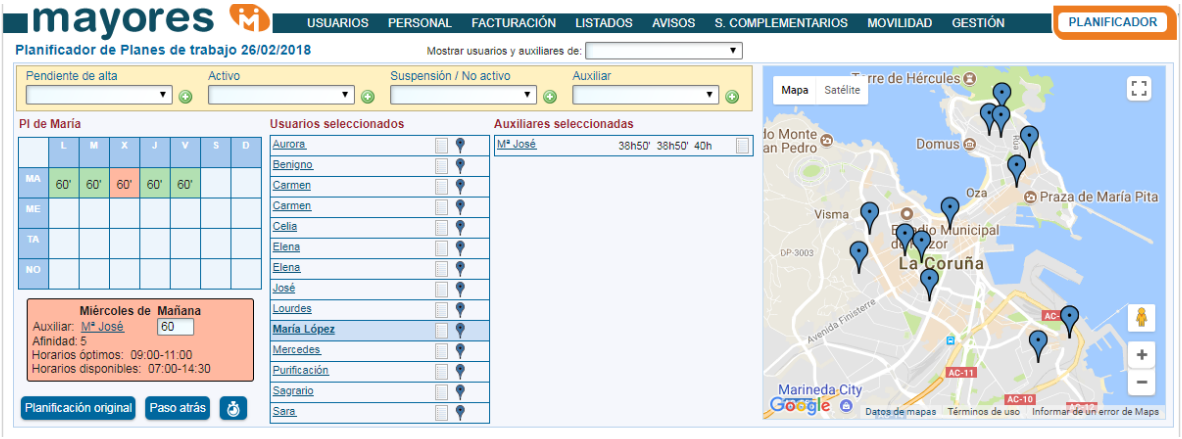


Figura 6.34: Plan de intervención final del usuario al que se le aumenta el número de servicios

En el plan de intervención de María, figura 6.34, se puede comprobar que ahora todos sus servicios están correctamente planificados, puesto que se encuentran de color verde, es decir, se han asignado a la auxiliar y se ha definido el horario en que deben realizarse. Seleccionando el servicio del miércoles se observa que ha sido asignado a Mª José y que el nivel de afinidad con esta auxiliar es 5, debido a que dicha auxiliar es la que está atendiendo al usuario en la actualidad.

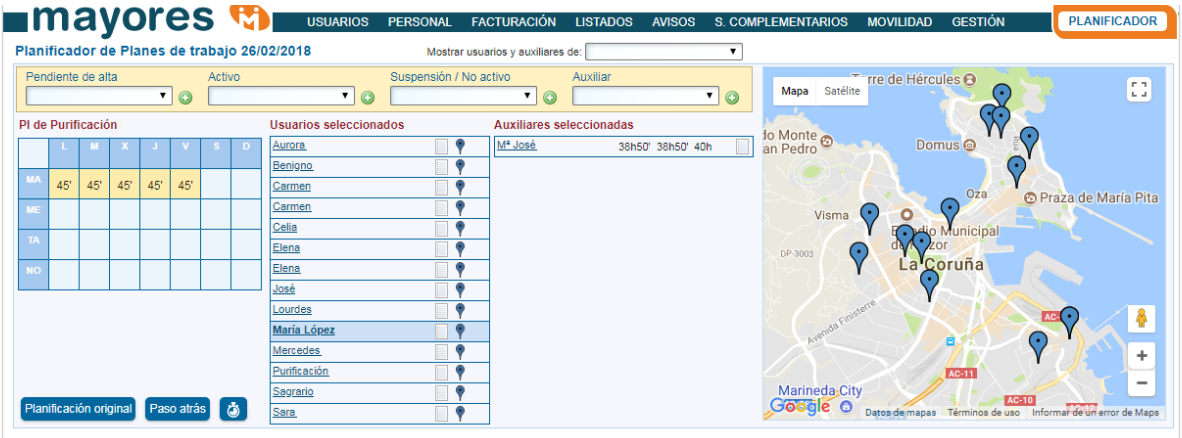


Figura 6.35: Plan de intervención final del usuario que se da de baja

Por otro lado, en el plan de intervención de Purificación, que se presenta en la figura 6.35, se observa que todos sus servicios ya no están planificados, puesto que aparecen marcados de color amarillo. De modo que dicho usuario ya no forma parte de la planificación de la auxiliar.

En la lista de auxiliares seleccionadas se muestra que la nueva planificación de Mª José tiene una duración de 38 horas y 50 minutos, lo cual respeta su jornada contratada de 40 horas semanales.

La planificación para los días martes, miércoles y jueves (días en los que había que eliminar los servicios de Purificación y añadir los nuevos servicios de María) se presenta en la figura 6.36. En esta nueva planificación se puede ver que los servicios de Purificación han sido correctamente eliminados. Del mismo modo también se puede observar que los servicios de María han sido añadidos a la planificación, en los tres casos el nuevo servicio ha sido colocado al principio del hueco de varias horas que tenía la auxiliar en su planificación inicial. Al eliminar los servicios de Purificación se generan descansos y por

ello es necesario que el sistema modifique el horario en que se realizan los servicios de Carmen y Elena con el fin de tener una planificación continuada.

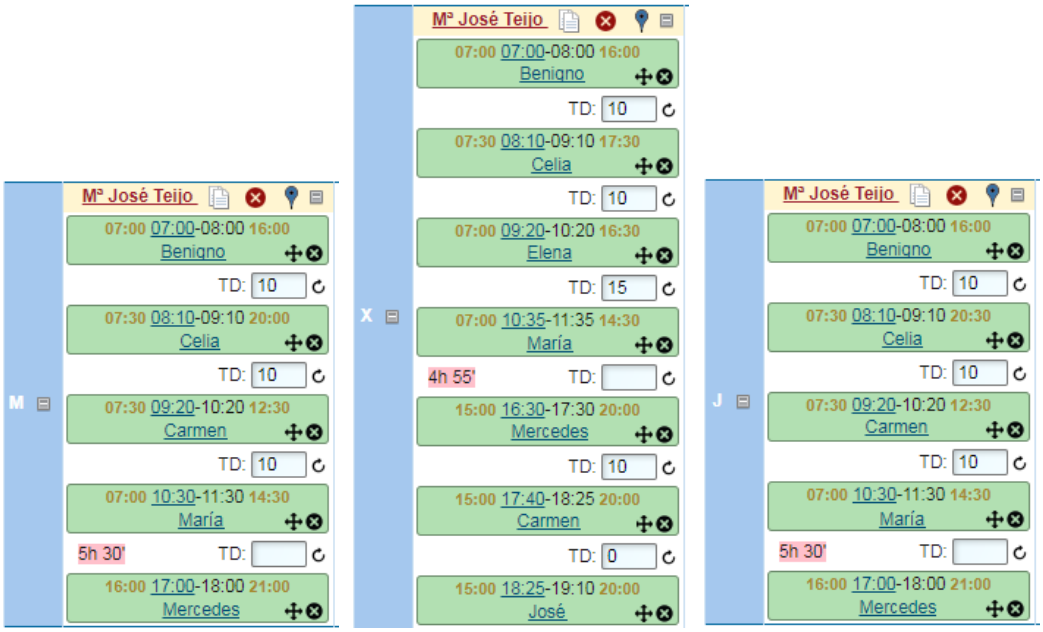


Figura 6.36: Planificación final para el lunes, miércoles y viernes

La planificación para los días lunes y viernes (días en que solamente se tienen que eliminar los servicios del usuario que se da de baja) se muestra en la figura 6.37.

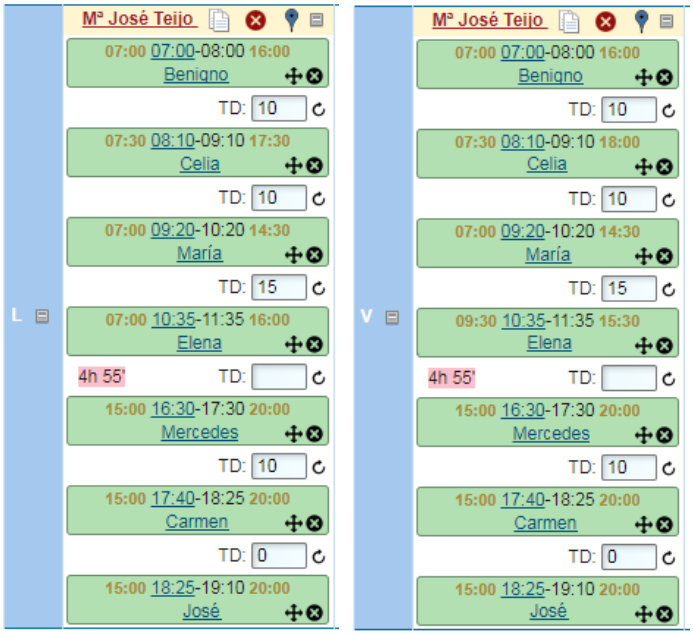


Figura 6.37: Planificación final para los días martes y jueves

Observando esta nueva planificación es posible comprobar que los servicios de Purificación han sido correctamente eliminados, puesto que ya no se encuentran planificados; al hacerlo se genera un hueco en la planificación de la auxiliar y, para subsanarlo, es necesario adelantar el horario en que se realizan los servicios de María y Elena. Notar que, el horario en que se realizan los servicios de la tarde (Mercedes, Carmen y José) no se modifica, esto se debe a que, entre medias, se tiene un descanso de cuatro horas en la jornada de la auxiliar; de modo que adelantar dichos servicios no proporcionaría ningún tipo de mejora (es más, se consideraría que la solución es peor puesto que mueve servicios de forma innecesaria).

De modo que el algoritmo diseñado ha servido para dar de baja un usuario, y ampliar el número de servicios de otro, de forma satisfactoria.

6.3.6. Alta de varios usuarios

En ejemplos anteriores se ha explicado el caso en el que se da de alta un nuevo usuario, pero puede ser que el objetivo sea dar de alta a más de un usuario a la vez, de modo que en una misma planificación se asignan los servicios de varios usuarios a las auxiliares consideradas.

En el ejemplo que se muestra a continuación se ilustra el caso en que se quiere dar de alta a dos usuarios, Hermenegildo y Jacinta, asignando sus servicios a la auxiliar Maribel.

Planificación inicial

Observando el plan de intervención de Hermenegildo, figura 6.38, se comprueba que requiere cinco servicios, en franja de tarde y con duración de 30 minutos cada uno de ellos. En concreto, estos servicios están marcados en color amarillo, lo cual indica que no están asignados a ninguna auxiliar.

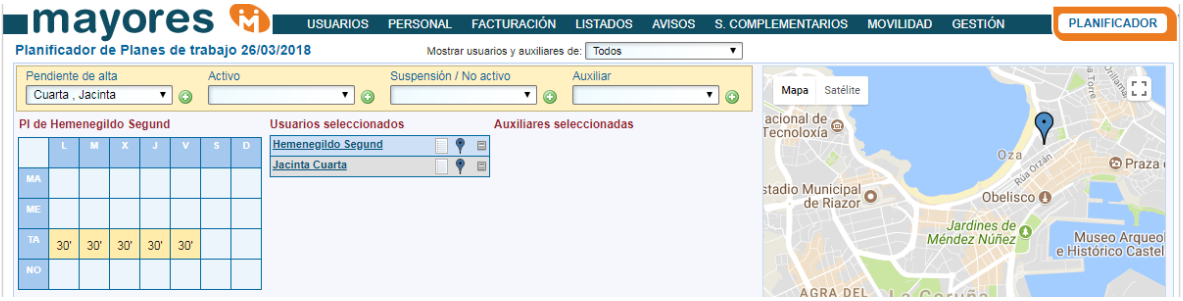


Figura 6.38: Plan de intervención del primero de los usuarios que se quiere dar de alta

En el plan de intervención de Jacinta, figura 6.39, se puede ver ver que requiere cinco servicios, en franja de tarde y con una duración de 60 minutos cada uno. Todos estos servicios se presentan en color amarillo, lo cual indica que no están asignados a ninguna auxiliar.

También es posible observar que la auxiliar seleccionada, Maribel, tiene una planificación inicial asignada de duración 22 horas y 45 minutos, lo cual respeta su jornada contratada de 26 horas semanales.

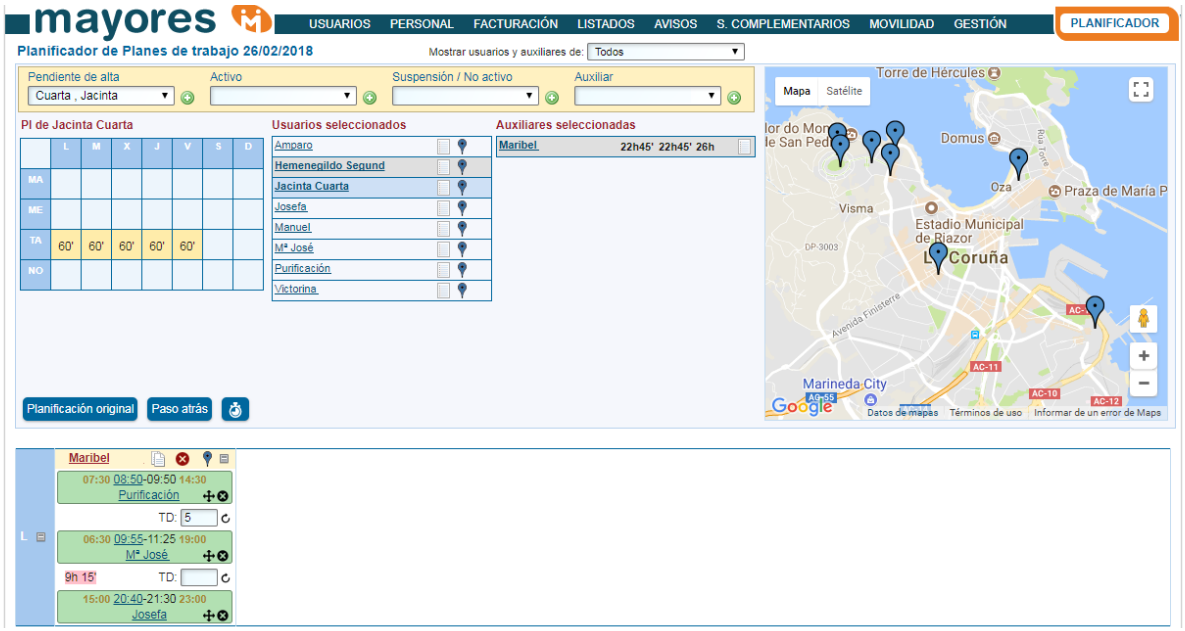


Figura 6.39: Plan de intervención del segundo de los usuarios que se quiere dar de alta

Replanificación automática

En este caso, debido a que se quieren asignar a la auxiliar todos los servicios de los dos usuarios que se desea dar de alta, se aumenta la jornada máxima permitida de la auxiliar a 35 horas semanales, tal y como se muestra en la figura 6.40.

Figura 6.40: Formulario de ampliación de la jornada

Planificación final

Tras generar la replanificación automática se obtiene una planificación en la cual los servicios de los nuevos usuarios han sido asignados a la auxiliar y se han determinado los horarios en los que deben realizarse dichos servicios.

Figura 6.41: Plan del intervención final del primero de los servicios

Observando el plan de intervención de Hermenegildo, figura 6.41, es posible comprobar que todos sus servicios han sido planificados, puesto que se muestran de color verde; esto quiere decir que han sido asignados a la auxiliar y que se ha determinado el horario en que deben llevarse a cabo. Seleccionando el servicio del martes se puede comprobar que ha sido asignado a Mª José y que el nivel de afinidad con esta auxiliar es 2, esto se debe a que la auxiliar no había atendido nunca al usuario.

La nueva planificación de la auxiliar tiene una jornada de 32 horas y 20 minutos, lo cual supera su jornada contratada de 26 horas semanales, pero respeta el máximo permitido de 35 horas a la semana.

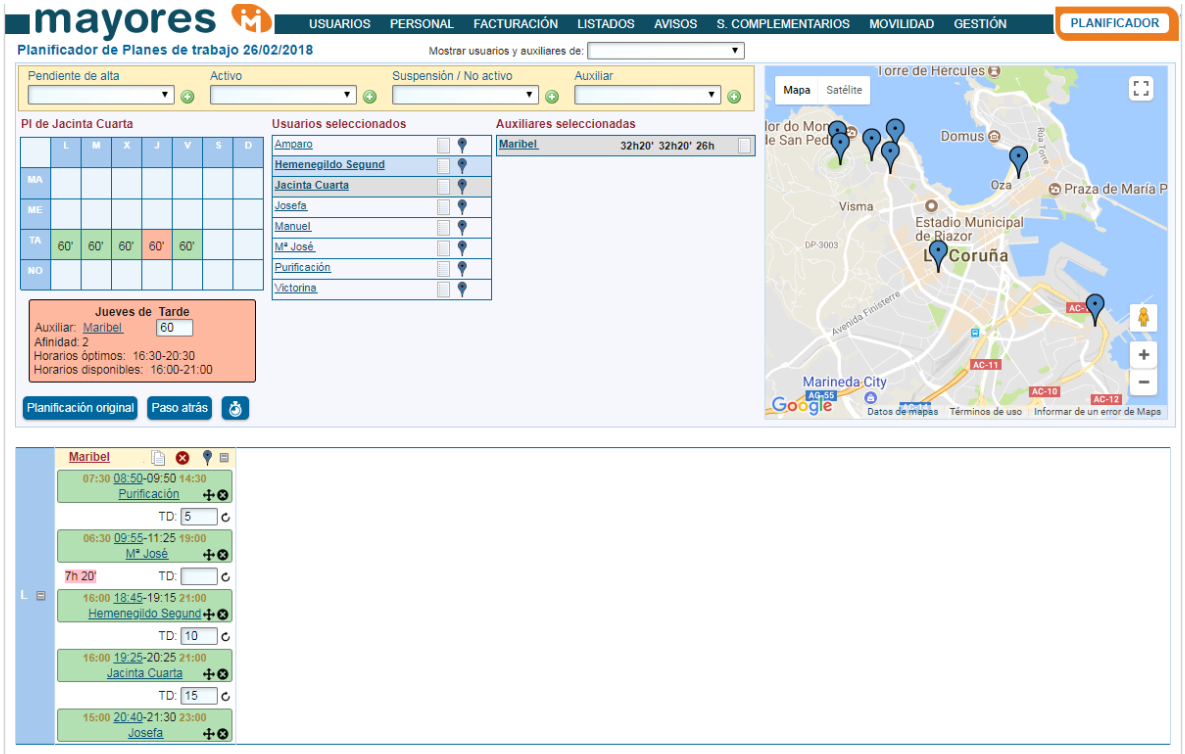


Figura 6.42: Planificación final para dar de alta dos usuarios

Del mismo modo, en el plan de intervención de Jacinta, presentado en la figura 6.42, se comprueba

que todos sus servicios han sido planificados, puesto que se muestran de color verde; esto quiere decir que han sido asignados a la auxiliar y que se ha determinado el horario en que deben llevarse a cabo. Seleccionando el servicio del jueves se observa que ha sido asignado a Rosa Bravo y que el nivel de afinidad con esta auxiliar es 2, esto se debe a que la auxiliar no había atendido nunca al usuario.

En la nueva planificación obtenida para el lunes se observa que se han añadido el servicio de Hermenegildo y el de Jacinta sin necesidad de modificar el horario en que se realizaban los servicios iniciales, puesto que estos nuevos servicios han sido colocados al final del hueco de varias horas que tenía la auxiliar en su planificación inicial, de modo que dicho hueco ha pasado de ser de 9 horas y 15 minutos a ser de 7 horas y 20 minutos.

Por tanto el algoritmo ha servido para planificar los servicios, de los dos usuarios que se quieren dar de alta, de forma satisfactoria.

6.3.7. Alta de un usuario, sin seleccionar auxiliares

Cuando el objetivo es dar de alta un nuevo usuario, en los casos estudiados hasta ahora, se seleccionaba la auxiliar que se deseaba utilizar para asignarle los nuevos servicios, pero también existe la posibilidad de que no se elija ninguna auxiliar, como ya se explicaba al comienzo del capítulo. Es decir, se permite que el sistema decida cuál es la mejor auxiliar (o auxiliares) para atender al nuevo usuario.

En este ejemplo se ilustra el caso en que, para dar de alta a un nuevo usuario, la coordinadora no selecciona el conjunto de auxiliares con el que desea trabajar.

Planificación inicial

En la parte izquierda del planificador, presentado en la figura 6.43, se muestra el plan de intervención de Linda, en éste se tienen los servicios que requiere el usuario; en este caso se tienen cinco servicios (uno para cada día de la semana) que están asignados a la franja de mañana y que tienen una duración de 60 minutos cada uno. Al seleccionar cualquiera de estos servicios se muestra un recuadro en el cual se detallan los horarios disponible y óptimo que tiene dicho servicio.

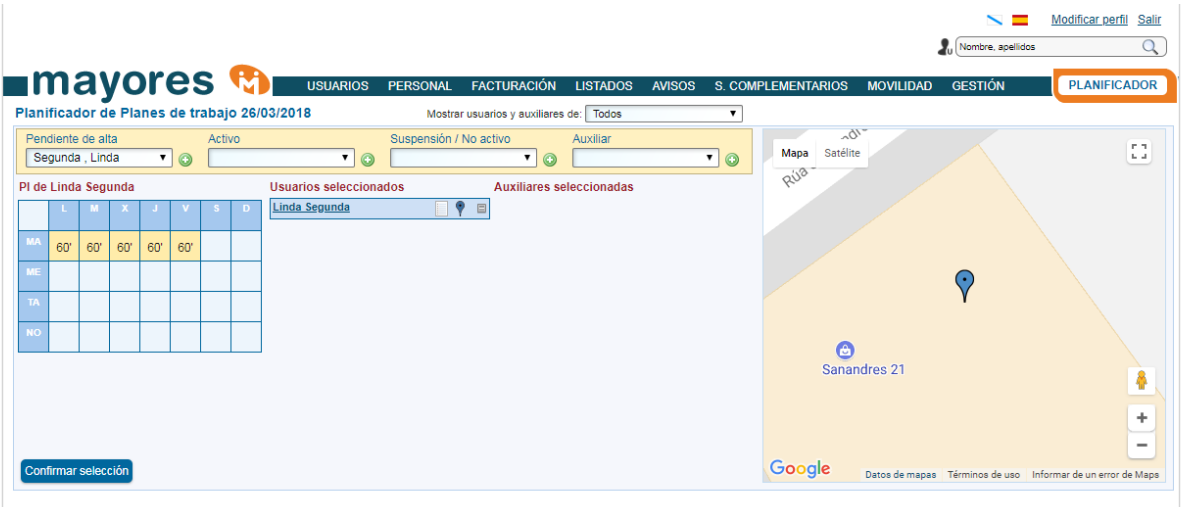


Figura 6.43: Plan de intervención del servicio a dar de alta

Replanificación automática

Como se comentaba anteriormente, al no seleccionar auxiliares se desconoce el contrato de la auxiliar a la que se le asignarán los servicios, por tanto, no es posible especificar un máximo permitido, tal y

como se muestra en la figura 6.44.

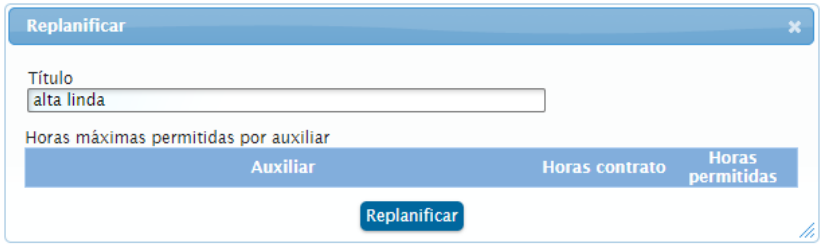


Figura 6.44: Formulario de ampliación de jornada en el caso en que no se hayan seleccionado auxiliares

Planificación final

Tras generar la replanificación automática se obtiene una planificación en la cual los servicios de los nuevos usuarios han sido asignados a una de las auxiliares de las que disponía la coordinadora. En este caso el sistema trabaja con todas las auxiliares de las que dispone la coordinadora, de modo que asigna los servicios a la mejor auxiliar (o auxiliares) disponible.

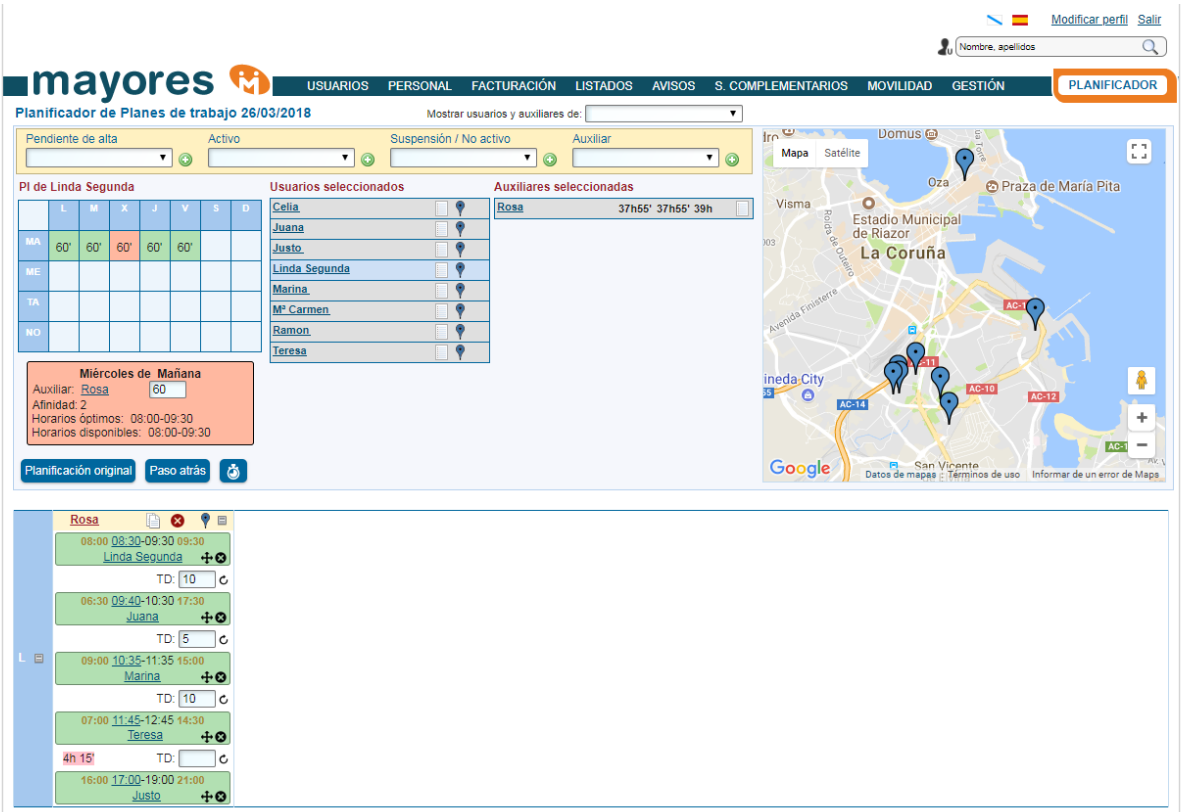


Figura 6.45: Planificación final para dar de alta un usuario sin seleccionar auxiliares

En la nueva planificación, presentada en la figura 6.45, se observa que se ha añadido al planificador la auxiliar Rosa, así como todos los usuarios a los que esta auxiliar atendía en su planificación inicial.

La jornada contratada de dicha auxiliar es de 39 horas, lo cual es respetado por la nueva planificación, que tiene una duración de 37 horas y 55 minutos. Cabe recalcar que, en este caso (no seleccionando ninguna auxiliar), no es posible que se generen planificaciones que superen la jornada contratada.

En el plan de intervención de Linda se observa que sus servicios aparecen de color verde, esto quiere decir que han sido asignados a la auxiliar y que se ha determinado el horario en que deben llevarse a cabo. Seleccionando el servicio del miércoles se comprueba que éste ha sido asignado a Rosa y que el nivel de afinidad con dicha auxiliar es 2, lo cual se debe a que la auxiliar no había atendido nunca al usuario. Se observa también que, en esta nueva planificación, se tienen los servicios del nuevo usuario colocados al inicio de la jornada de la auxiliar.

Por tanto el algoritmo puede planificar correctamente los servicios de un usuario que se da de alta en el caso en que la coordinadora no seleccione la auxiliar, o auxiliares, con la que quiere trabaja.

Conclusiones

Este trabajo, comienza con un estudio detallado del funcionamiento de la empresa mayores así como, de su situación actual. También se realiza un análisis del problema que se quiere resolver, determinando los elementos que se deben tener en cuenta, las restricciones que se deben satisfacer y los objetivos que se desea alcanzar.

En base al problema de Mayores, se realiza un estudio del estado del arte relacionado con éste. En concreto se presentan algunos problemas que, o bien son similares al que se quiere resolver, o simplemente pueden utilizarse como base para modelar el problema de Mayores.

El problema de planificación general de Mayores, se modela como uno de programación lineal entera. A pesar de ello, y debido a su alta complejidad, resulta necesario diseñar un algoritmo aproximado para poder resolver los problemas reales empleando tiempos computacionales reducidos.

El método de resolución diseñado, se basa en la técnica metaheurística del simulated annealing, siendo ésta implementada por nosotros e incorporada en la aplicación informática desarrollada para Mayores por el Laboratorio de Bases de Datos de la UDC. Este algoritmo tiene como objetivo principal el de proporcionar planificaciones donde se resuelven las incidencias consideradas de forma inmediata.

En este proyecto también se estudia cómo de eficaz resulta el algoritmo propuesto mediante la realización de una batería de ejemplos de pequeño tamaño. En ellos, se utiliza el algoritmo diseñado para obtener planificaciones desde cero; siendo las soluciones generadas por el algoritmo, comparadas con las soluciones que proporciona Gurobi al problema de planificación lineal modelado. En base a los resultados obtenidos, se concluye que la metodología diseñada tiene un comportamiento aceptable, puesto que proporciona planificaciones cercanas al óptimo en tiempos computacionales muy reducidos.

Para comprobar el buen funcionamiento del algoritmo desarrollado, se lleva a cabo la resolución de un conjunto de ejemplos, los cuales ilustran las posibles incidencias a las que se puede enfrentar una coordinadora. En todos los casos considerados, el método resuelve las incidencias de forma satisfactoria de manera inmediata, por lo que se puede concluir que el algoritmo se comporta adecuadamente.

Por último, se presenta el pseudocódigo del algoritmo planteado, donde se describen con detalle todos los pasos llevados a cabo en éste. Del mismo modo, también se incluye el código empleado para resolver los problemas de forma exacta empleando el solver Gurobi.

Cabe destacar que la empresa ha quedado satisfecha con el resultado de este trabajo, puesto que su principal objetivo era la obtención de planificaciones automáticas para facilitar la labor de las coordinadoras. El algoritmo propuesto solventa adecuadamente las incidencias teniendo en cuenta las restricciones impuestas por la empresa siendo las más importantes: la exigencia de no modificar las planificaciones previas de las auxiliares en exceso, junto con la obligación de respetar las preferencias de los usuarios a la hora de asignar sus servicios a las diferentes auxiliares de las que se dispone.

El algoritmo aún se encuentra en fase de incorporación a la aplicación informática que utilizará Mayores, puesto que alguno de los elementos que intervienen en él aun no se han añadido a la aplicación. Cuando ésta esté finalizada, las coordinadoras podrán hacer uso del planificador automático.

En el futuro, además de poder desarrollar otras técnicas para tratar de resolver el problema de planificación general que se presenta en este trabajo, también se podrá estudiar el problema de re-planificación puntual, explicado brevemente en el Capítulo 1. En él se deben obtener planificaciones automáticas, que resuelvan incidencias esporádicas que desaparecen con el paso del tiempo.

Apéndice A

Pseudocódigo del algoritmo

A continuación se presenta el pseudocódigo del algoritmo diseñado para implementar en la aplicación informática creada para Mayores.

A.1. Fase inicial

En esta primera fase se analizan los datos del tanteo, de modo que se obtienen los usuarios y auxiliares seleccionados, seguidamente se determina el conjunto de auxiliares que se van a utilizar durante la replanificación y los servicios que será necesario replanificar. En caso de que no se tengan servicios por replanificar se determina si es necesario optimizar la planificación; en caso de que así sea se aplica la optimización.

A continuación, se describen las variables de entrada de la función y se especifican aquellas que se utilizan internamente en ella. Por último, se presenta el pseudocódigo de la función, donde se muestra una explicación de los pasos que se están siguiendo.

Nombre y clase	Descripción
Variables de entrada	
tanteo, `Tanteo`	Tanteo que se está modificando
servicios, `List<ServicioTanteo>`	Lista con los servicios que ha sido seleccionados
personasSeleccionadas, `UsuarioTanteo`	Lista con las personas que requieren de los servicios
auxiliaresSeleccionadas, `AuxiliarTanteo`	Lista con las auxiliares que se han seleccionado
Variables que se utilizan en la función	
auxiliaresTanteo, `List<Auxiliar>`	Lista con auxiliares seleccionadas
personasNoActivas, `List<Persona>`	Lista con las personas no activas
personasBajaSuspension, `List<Persona>`	Lista con las personas en estado de baja o suspensión
mapPlanIntervencion, `Map<Long, PlanIntervencion>>`	Mapa que relaciona las personas con su plan de intervención

serviciosPlan, `ServicioTanteo`	Lista con los servicios de las personas que realizan auxiliares que están en el tanteo
serviciosNoTanteo, `ServicioTanteo`	Lista con los servicios de las personas que son realizados por auxiliares fuera del tanteo
auxiliaresDisponibles, `List<Auxiliar>`	Lista con todas las auxiliares disponibles
auxiliaresResumenHoras, `Map<Auxiliar, ResumenHoras>`	Mapa que relaciona auxiliares con sus resúmenes de horas
auxiliaresDisponiblesCoordinadora, `List<Auxiliar>`	Lista con todas las auxiliares disponibles de la coordinadora
auxiliaresBuenas, `List<Auxiliar>`	Lista de auxiliar que se utilizaran en la optimización
auxiliaresDiasDiponibles, `Map<Auxiliar, Lista<DiaSemana>>`	Mapa con las auxiliares y los días en que pueden realizar servicios
mapAuxiliaresServicios, `Map<Auxiliar, Lista<ServicioPlan>>`	Mapa que relaciona las auxiliaresBuenas con los servicios que realizan
auxiliaresNoRespetan, `List<Auxiliar>`	Lista con las auxiliares que superan su contrato
tramosNoActivos, `List<Tramo>`	Lista de tramos que nunca han sido planificados
tramosNoFactibles, `List<Tramo>`	Lista de tramos que ya estaban en el plan y que se deben replanificar
tramosAfinidadNula, `List<Tramo>`	Lista de tramos que antes los realizaba una auxiliar vetada por el usuario
tramosReplanificar, `List<Tramo>`	Lista con todos los tramos que se deben planificar
personasConServiciosNoActivos, `List<Persona>`	Lista de personas que corresponden a los *tramosNoActivos*
personasConServiciosNoFactibles, `List<Persona>`	Lista de personas que corresponden a los *tramosNoFactibles*
personasConServiciosAfinidadNula, `List<Persona>`	Lista de personas que corresponden a los *tramosAfinidadNula*
personasReplanificar, `List<Persona>`	Lista de personas que corresponden a los *tramosReplanificar*
personasTramos, `Map<Persona, List<Tramo>>`	Mapa que relaciona cada persona con los tramos a realizar
planOptimizado, `List<ServicioPlan>`	Lista de servicios ya optimizados
serviciosFinales, `List<ServicioTanteo>`	Lista de servicios ya optimizados

Algoritmo A.1: Fase inicial

```
1  Inicio
   Paso 0.- Almacenar las auxiliares y los usuarios seleccionados
2  auxiliaresTanteo ← auxiliaresSeleccionadas
3  personas ← personasSeleccionadas
4  mapaAuxiliaresHoras ← horas máximas permitidas de cada auxiliar
5  personasNoActivas
6  personasBajaSuspension
   Paso 1-. Almacenar el plan de intervención de cada usuario
7  mapPlanIntervencion
8  for persona en personas do
9  |  añadir a mapPlanIntervencion el id y el plan de intervención de persona
10 |  if estadoPersona = null o  estadoPersona != activo then
11 |  |  añadir  persona  a  personasNoActivas
12 |  |  if estadoPersona = baja o  estadoPersona = suspension then
13 |  |  |  añadirpersonaapersonasBajaSuspension
14 |  |  end
15 |  end
16 end
17 serviciosPlan
18 serviciosNoTanteo
   Paso 3.- Almacenar servicios ocultos de los usuarios (servicios que realizan
   auxiliares de fuera del tanteo)
19 añadir a  serviciosNoTanteo los servicios ocultos del tanteo
   Paso 4.- Almacenar los servicios de los usuarios activos
20 añadir a  serviciosPlan  los servicios de las personas activas
   Paso 5.- Seleccionar todas las auxiliares disponibles
21 auxilaresDisponibles ← todas las auxiliares disponibles de la empresa
22 auxiliaresNoActivas
23 auxiliaresDisponiblesCoordinadora
   Paso 6.- Seleccionar las auxiliares no activas y las auxiliares de la
   coordinadora
24 for auxiliar en auxiliaresDisponibles do
25 |  if auxiliar no está activa then
26 |  |  añadir la auxiliar a auxiliaresNoActivas
27 |  else
28 |  |  if auxiliar es de la coordinadora then
29 |  |  |  añadir la auxiliar a auxiliaresDisponiblesCoordinadora
30 |  |  end
31 |  end
32 end
33 eliminar de auxiliaresDisponibles las auxiliaresNoDisponibles
   Paso 7.- Seleccionar las auxiliares que se usarán en la planificación (las
   auxiliares seleccionadas o todas las auxiliares)
34 auxiliresBuenas
35 if auxiliaresTanteo ≠ ∅ then
36 |  auxiliaresBuenas ← auxiliaresTanteo
37 else
38 |  auxiliaresBuenas ← auxiliresDisponibleCoordinadora
39 end
```

```
   Paso 8.- En caso de que se trabaje con todas las auxiliares de la
   coordinadora, eliminar de la lista las auxiliares que tengan horas positivas
   en su bolsa de horas
40 auxiliaresNoDisponibles
41 if auxiliaresTanteo = ∅ then
42 |  for auxiliar en auxiliaresBuenas do
43 |  |  if bolsaHorasAuxiliar > 0 then
44 |  |  |  añadir la auxiliar a auxiliaresNoDisponibles
45 |  |  end
46 |  end
47 end
48 eliminar de auxiliaresBuenas las auxiliaresNoDisponibles
   Paso 9.- Obtener los días que está disponible cada una de las auxiliares
49 auxiliaresDiasDisponibles
50 for auxiliar en auxiliaresBuenas do
51 |  añadir a auxiliaresDiasDisponibles  la auxiliar y sus días disponibles
52 end
   Paso 10.- Obtener la planificación de las auxiliares y eliminar de la lista
   (en caso de que se trabaje con todas las auxiliares) aquellas que superan su
   contrato
53 mapAuxiliaresServicios
54 auxiliaresNoRespetan
55 for auxiliar en auxiliaresBuenas do
56 |  añadir a mapAuxiliaresServicios  la auxiliar y los servicios activos que realiza
57 |  if horasTrabajadasAuxiliar > horasContratadasAuxiliar then
58 |  |  añadir la auxiliar a auxiliaresNoRespetan
59 |  end
60 end
61 if auxiliaresTanteo = ∅ then
62 |  eliminar de auxiliaresBuenas las auxiliaresNoRespetan
63 end
   Paso 11.- Seleccionar los tramos no planificados que requieren los usuarios
64 tramosNoActivos
65 personasConServiciosNoActivos
66 for persona en personas do
67 |  if persona está activa y planIntervencionPersona = ∅ then
68 |  |  for tramo en tramosPersona do
69 |  |  |  if tramo no está planificado then
70 |  |  |  |  añadir el tramo a tramosNoActivo
71 |  |  |  |  if persona ∉ personasConServiciosNoActivos then
72 |  |  |  |  |  añadir la persona a personasConServiciosNoActivos
73 |  |  |  |  end
74 |  |  |  end
75 |  |  end
76 |  end
77 end
```

```
   Paso 12.- Seleccionar los tramos mal planificados de los usuarios
78 tramosNoFactibles
79 personasConServiciosNoFactibles
80 for persona en personas do
81 |  if persona está activa y planIntervencionPersona = ∅ then
82 |  |  for tramo en tramosPersona do
83 |  |  |  if el tramo no está bien planificado then
84 |  |  |  |  añadir el tramo a tramosNoFactibles
85 |  |  |  |  if persona ∉ personasConServiciosNoFactibles then
86 |  |  |  |  |  añadir la persona a personasConServiciosNoFactibles
87 |  |  |  |  end
88 |  |  |  end
89 |  |  end
90 |  end
91 end
   Paso 13.- Seleccionar los tramos de los usuarios asignados a auxiliares con
   las que tienen afinidad nula
92 tramosAfinidadNula
93 personasConServiciosAfinidadNula
94 for persona en personas do
95 |  if persona está activa y planIntervencionPersona = ∅ then
96 |  |  for tramo en tramosPersona do
97 |  |  |  if afinidad entre el tramo y su auxiliar es nula then
98 |  |  |  |  añadir el tramo a tramosAfinidadNula
99 |  |  |  |  if persona ∉ personasConServiciosAfinidadNula then
100|  |  |  |  |  añadir la persona a personasConServiciosAfinidadNula
101|  |  |  |  end
102|  |  |  end
103|  |  end
104|  end
105end
   Paso 14.- Seleccionar todos los tramos que será necesario planificar, los
   usuarios que requieren dichos tramos y crear un mapa que relacione las
   personas con sus tramos
106tramosReplanificar ← tramosNoActivos ∪ tramosNoFactibles ∪ tramosAfinidadNula
107personasReplanificar ← personasConServiciosNoActivos ∪ personasConServiciosNoFactibles ∪
    personasConServiciosAfinidadNula
108personasTramos
109for persona en personasReplanificar do
110|  añadir la persona y sus tramosReplanificar a personasTramos
111end
   Paso 15.- En caso de que no haya tramos que replanificar, comprobar si hay
   descansos y, si los hay, aplicar una optimización
112if  tramosReplanificar ≠ ∅ then
113|  if tiempo de descanso de la planificación = 0 then
114|  |  return servicioPlan
115|  else
116|  |  return SA(servicioPlan)
117|  end
118end
```

```
   Paso 16.- Pasar los servicios del tanteo al formato ServicioPlan y ordenarlos
119 serviciosPlan ← ServicioPlan(servicios)
120 serviciosPlanOrdenados ← ordenar(serviciosPlan)
   Paso 17.- Aplicar la optimización
121 parámetros
122 planOptimizado ← optimizaciónSA(parámetros, serviciosPlanOrdenados, auxiliaresBuenas,
      tramosReplanificar, personasReplanificar, auxiliaresDiasDisponibles)
   Paso 18.- Actualizar las auxiliares y los servicios según la planificación
   optimizada
123 auxiliaresSeleccionadas ← auxiliares(planOptimizado)
124 personasSeleccionadas ← personas(planOptimizado)
   Paso 19.- Añadir a la planificación optimizada los servicios ocultos y pasar
   los servicios al tipo Serviciotanteo
125 servicioFinales ← serviciosOultos(planOptimizado)
126 servicioFinales ← servicioFinales ∪ ServicioTanteo(planOptimizado)
   Paso 20.- Devolver la lista de servicios optimizada
127 return servicioFinales
```

A.2. Replanificación

En esta fase se realiza una replanificación de las planificaciones iniciales de las auxiliares, aquí se determina la auxiliar que debe realizar cada uno de los nuevos servicios, así como el horario en que deben realizarse.

En primer lugar, se describen las variables de entrada de la función y seguidamente se detallan aquellas que se utilizan internamente en ésta. Por último, se muestra el pseudocódigo de la función, detallando los pasos que se están siguiendo.

Nombre y clase	Descripción
Variables de entrada	
serviciosPlanOrdenados, `List<ServicioPlan>`	Lista de servicios que se tienen en el tanteo
auxiliaresBuenas, `List<Auxiliar>`	Lista de auxiliares disponibles
auxiliaresDiasDisponibles, `Map<Auxiliar, List<DiaSemana>>`	Mapa que relaciona las auxiliares con la lista de día en los que están disponibles
tramosReplanificar, `List<Tramo>`	Lista de tramos que se deben replanificar
personasReplanificar, `List<Persona>`	Lista de personas que tienen tramos por replanificar
personasTramos, `Map<Persona, List<Tramo>>`	Mapa que relaciona cada persona con los tramos a realizar
Variables que se utilizan en la función	
planificacionTanteo, `List<ServicioPlan>`	Lista con los servicios que realizan las auxiliares del tanteo

tramospersonaFranja, `Map<Persona, Map<FranjaTramo, List<Tramo>>>`	Mapa que relaciona cada persona con un mapa que almacena las franjas y los distintos tramos que requiere la persona en cada una de ellas
personasOrdenadas, `List<Persona>`	Lista de personas ordenada
franjasTramos, Map<FranjaTramo, `List<Tramo>>>`	Mapa que relaciona las franjas y los tramos
listaTramosFranja, `List<Tramo>`	Lista de tramos que se deben realizar en una franja
mapTramosAuxiliares, `Map<Tramo, List<Auxiliar>>`	Mapa que relaciona tramos con la lista de auxiliares ordenadas
mejoresAuxiliaresTramo, `List<Auxiliar>`	Lista con las auxiliares ordenadas de mejor a peor para realizar el tramo
auxiliarFicticia, `Auxiliar`	Auxiliar ficticia utilizada para planificar servicios que no pueden ser realizados por las auxiliares reales
asignarFicticia, `boolean`	Parámetro que determina si un tramo debe asignarse a una auxiliar ficticia o no
Servprueba, `ServicioPlan`	Servicio que se corresponde con el tramo ya planificado
serviciosAuxiliarSemana, `List<ServicioPlan>`	Lista de servicios que una auxiliar tiene planeados para toda la semana
serviciosAuxiliarDia, `List<ServicioPlan>`	Lista de servicios que una auxiliar tiene planeados para cierto día

Algoritmo A.2: Replanificación

```
1  Inicio
   Paso 1.- Seleccionar los servicios y las auxiliares que los llevan a cabo
2  planificacionTanteo ← serviciosPlanOrdenados
3  auxiliaresTotal ← auxiliares(planificacionTanteo)
   Paso 2.- Seleccionar los tramos por replanificar de cada persona asignados a
   cada franja
4  tramosPersonaFranja
5  for persona en personaTramos do
6  |  planificacionTanteo ← ∅
7  |  tramosPersona ← tramos(persona)
8  |  foreach franja do
9  |  |  tramos ← tramosFranja(tramosPersona )
10 |  |  añadir a tramosFranja la franja y tramos
11 |  end
12 |  añadir a tramosPersonaFranja  la persona y tramosFranja
13 end
   Paso 3.- Ordenar las personas
14 personasOrdenadas ← ordenar(personas(personasTramos))
```

```
   Paso 4.- Para cada persona recorrer sus franjas y tramos y, en caso de que sea
   necesario, optimizar
15 for persona en personasOrdenadas
16 |  franjasTramos ← franjaTramos(personas, tramosPersonaFranja)
17 |  for franja en franjasTramos
18 |  |  listaTramosFranja ← tramos(franja, franjasTramos)
   |  |  Paso a.- Obtener las auxiliares ordenadas para cada tramo
19 |  |  mapTramosAuxiliares ← ordenarAuxiliares(listaTramosFranja, auxiliaresBuenas)
   |  |  Paso b.- Asignar cada tramo a una auxiliar
20 |  |  for tramo en listaTramosFranja
21 |  |  |  planificacionTanteo ← false
22 |  |  |  mejoresAuxiliaresTramo ← auiliares(mapTramosAuxiliares)
   |  |  |  Paso. - Si el tramo es de semana, poner como primera auxiliar de la
   |  |  |  lista aquella auxiliar que ya realice servicios del mismo tipo que el
   |  |  |  tramo (es decir, tramos del mismo usuario y misma franja)
23 |  |  |  auxiliaresAtienden if dia(tramo) ∉ findeSemana then
24 |  |  |  |  for servicio en planificacionTanteo do
25 |  |  |  |  |  if usuario(tramo) = usuario(servicio) y franja(tramo) =
   |  |  |  |  |   franja(servicio) then
26 |  |  |  |  |  |  if auxiliar ∉ auxiliaresAtienden then
27 |  |  |  |  |  |  |  añadir la auxiliar a auxiliaresAtienden
28 |  |  |  |  |  |  end
29 |  |  |  |  |  end
30 |  |  |  |  end
31 |  |  |  |  poner auxiliaresAtienden al principio de mejoresAuxiliaresTramo
32 |  |  |  end
   |  |  |  Paso c.- Si no hay auxiliares para ese tramo asignarlo a la auxiliar
   |  |  |  ficticia
33 |  |  |  if mejoresAuxiliaresTramo = ∅ then
34 |  |  |  |  servicioprueba ← aniadirServicio(tramo, auxiliarFicticia)
35 |  |  |  |  añadir servicioprueba a planificacionTanteo
36 |  |  |  |  if auxiliarFicticia ∉ auxiliaresTotal then
37 |  |  |  |  |  añadir axuliliarFicticia a auxiliaresTotal
38 |  |  |  |  end
39 |  |  |  |  pasar al siguiente tramo
40 |  |  |  end
   |  |  |  Paso d.- Recorrer las auxiliares ordenadas
41 |  |  |  for auxiliar en mejoresAuxiliaresTramo do
42 |  |  |  |  if auxiliar ∈ auxiliaresFicticias then
43 |  |  |  |  |  asignarFicticia ← true
44 |  |  |  |  end
45 |  |  |  end
```

```
46
47
48
49
        Paso e.- Si la auxiliar no está entre las que realizan los
        servicios será necesario añadirla
50      if auxiliar ∉ auxiliaresTotal
51          serviciosAuxiliarSemana ← planificacionPrevista(auxiliar)
            Paso f.- Comprobar si es posible que la auxiliar realice el
            tramo
52          if horasTrabajadasSemana(auxiliar) >
             horasPermitidasSemana(auxiliar) then
53              if auxiliar es la ultima de mejoresAuxiliaresTramo then
54                  asignarFicticia ← true
55              else
56                  pasar a la siguiente auxiliar
57              end
58          end
59          if horasTrabajadasDia(auxiliar) > horasPermitidasDia(auxiliar) then
60              if auxiliar es la ultima de mejoresAuxiliaresTramo then
61                  asignarFicticia ← true
62              else
63                  pasar a la siguiente auxiliar
64              end
65          end
66          if auxiliarSaturada(auxiliar) = true then
67              if auxiliar es la ultima de mejoresAuxiliaresTramo then
68                  asignarFicticia ← true
69              else
70                  pasar a la siguiente auxiliar
71              end
72          end
73          if incompatibilidades(auxiliar, tramo) = true then
74              if auxiliar es la ultima de mejoresAuxiliaresTramo then
75                  asignarFicticia ← true
76              else
77                  pasar a la siguiente auxiliar
78              end
79          end
            Paso g.- Si no es posible realizar el tramo pasar a la siguiente
            auxiliar o asignar a la ficticia
80          servprueba serviciosNuevos ← planificacionTanteo
81          if asignarFicticia = true then
82              if auxiliar es la ultima de mejoresAuxiliaresTramo then
83                  servicioprueba ← aniadirServicio(tramo, auxiliarFicticia)
84                  if auxiliarFicticia ∉ auxiliaresTotal then
85                      añadir axuliliarFicticia a auxiliaresTotal
86                  end
87                  añadir servprueba a serviciosNuevos
88              else
89                  pasar a la siguiente auxiliar
90              end
91          end
```

```
92
93
94
95
96
        Paso h.- Si se puede realizar el tramo colocarlo en la mejor
        posición dentro de la planificación de la auxiliar
97      if asignarFicticia = false then
98          serviciosAuxiliarSemana ← serviciosPlaneados(auxiliar)
99          serviciosAuxiliarDia ← serviciosPlaneados(auxiliar, dia)
100         servicioprueba ← aniadirServicio(tramo, auxiliar,
              serviciosAuxiliarDia)
101         añadir servprueba a serviciosNuevos
102         añadir serviciosAuxiliarSemana a planificacionTanteo y a
              serviciosNuevos
103     end
        Paso i.- Optimizar la planificación a la que se ha añadido el
        nuevo servicio
104     serviciosTotalesOrdenados ← ordenar(serviciosNuevos)
105     if asignarFicticia = true then
106         planSinSolapamientos ← serviciosTotalesOrdenados
107     else
108         planSinSolapamientos ← SA(serviciosTotalesOrdenados, auxiliaresTotal)
109     end
        Paso j.- Si se han eliminado los solapamientos actualizar la
        planificación y pasar al siguiente tramo
110     solapamiento ← solapamiento(planSinSolapamientos)
111     if solapamiento = 0 then
112         planificacionTanteo ← planSinSolapamientos
113         pasar al siguiente tramo
114     else
            Paso j.- Si no se han eliminado todos los solapamientos, o se
            supera el máximo de horas permitidas para la auxiliar,
            eliminar los servicios que realizaba la auxiliar y pasar a la
            siguiente auxiliar o a la auxiliar ficticia en caso de que se
            estuviera tratando con la última de la lista
115         eliminar auxiliar de auxiliaresTotal y de auxiliaresTanteo
116         serviciosAuxiliar ← serviciosPlanificados(planSinSolapamientos,
              auxiliar)
117         texteliminar serviciosAuxiliar de planSinSolapamientos
118         if auxiliar es la ultima de mejoresAuxiliaresTramo then
119             servicioprueba ← aniadirServicio(tramo, auxiliarFicticia)
120         else
121             pasar a la siguiente auxiliar
122         end
123     end
124 end
```

```
125
126
127
128
        Paso k.- Si la auxiliar ya está entre las que realizan los
        servicios es necesario trabajar con los servicios de la lista
        asignados a ella
129     if auxiliar ∈ auxiliaresTotal then
130         serviciosAuxiliarSemana(planificacionTanteo, auxiliar)
            Paso l.- Comprobar si es posible que la auxiliar realice el
            tramo
131         if horasTrabajadasSemana(auxiliar) >
              horasPermitidasSemana(auxiliar) then
132             if auxiliar es la ultima de mejoresAuxiliaresTramo then
133                 asignarFicticia ← true
134             else
135                 pasar a la siguiente auxiliar
136             end
137         end
138         if horasTrabajadasDia(auxiliar) > horasPermitidasDia(auxiliar) then
139             if auxiliar es la ultima de mejoresAuxiliaresTramo then
140                 asignarFicticia ← true
141             else
142                 pasar al la siguiente auxiliar
143             end
144         end
145         if auxiliarSaturada(auxiliar) = true then
146             if auxiliar es la ultima de mejoresAuxiliaresTramo then
147                 asignarFicticia ← true
148             else
149                 pasar al la siguiente auxiliar
150             end
151         end
152         if incompatibilidades(auxiliar, tramo) = true then
153             if auxiliar es la ultima de mejoresAuxiliaresTramo then
154                 asignarFicticia ← true
155             else
156                 pasar a la siguiente auxiliar
157             end
158         end
159     end
```

```
160
161
162
163
164
        Paso m.- Si no es posible realizar el tramo pasar a la siguiente
        auxiliar o asignar a la ficticia
165     servprueba
166     serviciosNuevos ← planificacionTanteo
167     if asignarFicticia = true then
168         if auxiliar es la ultima de mejoresAuxiliaresTramo then
169             servicioprueba ← aniadirServicio(tramo, auxiliarFicticia)
170             if auxiliarFicticia ∉ auxiliaresTotal then
171                 añadir axuliliarFicticia a auxiliaresTotal
172             end
173             añadir servprueba a serviciosNuevos
174         else
175             pasar a la siguiente auxiliar
176         end
177     end
        Paso n.- Si se puede realizar el tramo colocarlo en la mejor
        posición dentro de la planificación de la auxiliar
178     if asignarFicticia = false then
179         serviciosAuxiliarDia ← serviciosAuxiliarDia(planificaciontanteo,
            auxiliar, dia)
180         servicioprueba ← aniadirServicio(tramo, auxiliar,
            serviciosAuxiliarDia)
181         añadir servprueba a serviciosNuevos
182     end
        Paso ñ.- Optimizar la planificación a la que se ha añadido el
        nuevo servicio
183     serviciosTotalesOrdenados ← ordenar(serviciosNuevos)
184     if asignarFicticia = true then
185         planSinSolapamientos ← serviciosTotalesOrdenados
186     else
187         planSinSolapamientos ← SA(serviciosTotalesOrdenados, auxiliaresTotal)
188     end
        Paso j.- Si se han eliminado los solapamientos actualizar la
        planificación y pasar al siguiente tramo
189     solapamiento ← solapamiento(planSinSolapamientos)
190     if solapamiento = 0 then
191         planificacionTanteo ← planSinSolapamientos
192         pasar al siguiente tramo
193     end
        Paso j.- Si no se han eliminado todos los solapamientos, o se
        supera el máximo de horas permitidas para la auxiliar, eliminar
        los servicios que realizaba la auxiliar y pasar a la siguiente
        auxiliar o a la auxiliar ficticia en caso de que se estuviera
        tratando con la última de la lista
```

```
194
195 |
196 |   |
197 |   |   |
198 |   |   |   |
199 |   |   |   |   if solapamiento ≠ 0 then
200 |   |   |   |   |   eliminar auxiliar de auxiliaresTotal y de auxiliaresTanteo
201 |   |   |   |   |   serviciosAuxiliar ← serviciosPlanificados(planSinSolapamientos,
    |   |   |   |   |    auxiliar)
202 |   |   |   |   |   texteliminar serviciosAuxiliar de planSinSolapamientos
203 |   |   |   |   |   if auxiliar es la ultima de mejoresAuxiliaresTramo then
204 |   |   |   |   |   |   servicioprueba ← aniadirServicio(tramo, auxiliarFicticia)
205 |   |   |   |   |   else
206 |   |   |   |   |   |   pasar a la siguiente auxiliar
207 |   |   |   |   |   end
208 |   |   |   |   end
209 |   |   |   end
210 |   |   end
211 |   end
212 |  end
213 end
    Paso 5.- Si la planificación tiene descansos y/o solapamientos optimizarla
214 descanso ← descansos(planificacionTanteo)
215 solapamiento ← solapamientos(planificacionTanteo)
216 if descanso ≠ 0 o solapameintos ≠ 0 then
217 |  serviciosModificables ← serviciosDescansoSolapamiento(planificacionTanteo)
218 |  auxiliaresModificables ← auxiliares(serviciosModificables)
219 |  planFinal ← planificacionTanteo
220 |  for auxiliares en auxiliaresModificables do
221 |  |  dias ← diasServicios(auxiliar, serviciosModificables)
222 |  |  for dia en dias do
223 |  |  |  planFnal ← SA(planFinal,serviciosModificables)
224 |  |  end
225 |  end
226 |  planificacionTanteo ← planFinal
227 end
    Paso 6.- Si la planificación tiene servicios asignados a auxiliares ficticias,
    optimizarla
228 auxiliaresPlanificacion ← auxiliares(planificacionTanteo)
229 if auxiliaresPlanificacion cotiene alguna auxiliarFicticia then
230 |  planFnal ← SA(planificacionTanteo)
231 |  planificacionTanteo ← planFinal
232 end
    Paso 7.- Si es necesario, para cada auxiliar, se tratan de igualar las horas
    de inicio de los diferentes días en los que trabaja
233 if igualarInicios = true then
234 |  planificacionTanteo ← igualarIniciosPlanificacion(planificacionTanteo)
235 end
    Paso 8.- Devolver la planificación optimizada
236 return planificacionTanteo
```

A.3. Optimización

Esta fase consiste en optimizar las planificaciones de las auxiliares con las que se está trabajando, es decir, aplicarles el *simulated annealing* para mejorarlas. Para ello se parte de la planificación inicial y se le aplican movimientos para generar un vecindario, de modo que la solución se va actualizando con la mejor solución de dicho vecindario.

En primer lugar, se detallan las variables de entrada de la función y seguidamente se describen aquellas que se utilizan internamente en ella. Por último, se muestra el pseudocódigo de la función, donde se muestra una explicación de los pasos que se están siguiendo.

Nombre y clase	Descripción
Variables de entrada	
solucionInicial, `List<ServicioPlan>`	Lista de servicios que es necesario optimizar
serviciosModificables, `List<ServicioPlan>`	Lista de servicios que se pueden modificar
auxiliaries, `List<Auxiliar>`	Lista de auxiliares
auxiliareFicticias, `List<Auxiliar>`	Lista de auxiliares ficticias
tramosServicios, `Map<ServicioPlan, Tramo>`	Mapa que relaciona el ServicioPlan con su Tramo
nIterMax, `int`	Número de iteraciones máximo
L, `int`	Tamaño del vecindario
soloServicioNuevo, `boolean`	Parámetro que determina si solamente se quieren considerar los servicios que realiza la auxiliar en el día en que realiza el servicio nuevo
soloReplanificables, `boolean`	Parámetro que determina si solamente se quieren considerar los servicios replanificables
eliminarFicticias, `boolean`	Parámetro que determina si se quieren eliminar las auxiliares ficticias
cambiarFuncionObjetivo, `boolean`	Parámetro que determina si se quiere cambiar la función objetivo durante la ejecución del algoritmo
Variables que se utilizan en la función	
Candidata, `List<ServicioPlan>`	Solución candidata a ser la mejor del vecindario
candidataOrdenada, `List<ServicioPlan>`	Solución candidata ordenada por auxiliar, día y hora
auxiliaresCandidata, `List<Auxiliar>`	Lista de auxiliares que realizan los servicios de la solución candidata
solucionActual, `List<ServicioPlan>`	Solución actual que se quiere mejorar
solucionActualOrdenada, `List<ServicioPlan>`	Solución actual ordenada
auxiliaresSolucionActual, `List<Auxiliar>`	Auxiliares que realizan los servicios de la solución actual
diferencia, `double`	Diferencia entre dos soluciones
nIteraciones, `int`	Número de iteraciones
Movimiento, `String`	Movimiento utilizado para generar vecinos

movReplanificar, `int`	Parámetro que determina la probabilidad de elegir el movimiento *Replanificar*
movDesplazar, `int`	Parámetro que determina la probabilidad de elegir el movimiento *Desplazar*
movIntercambiarDentro, `int`	Parámetro que determina la probabilidad de elegir el movimiento *IntercambiarDentro*
movIntercambiarFranja, `int`	Parámetro que determina la probabilidad de elegir el movimiento *IntercambiarFranja*
movIntercambiarFuera, `int`	Parámetro que determina la probabilidad de elegir el movimiento *IntercambiarFuera*
movPasarFuera, `int`	Parámetro que determina la probabilidad de elegir el movimiento *PasarFuera*
movDesplDos, `int`	Parámetro que determina la probabilidad de elegir el movimiento *DesplazarDos*
movInterDespl, `int`	Parámetro que determina la probabilidad de elegir el movimiento *IntercambirDesplazar*
serviciosInicialesUtiles, `List<ServicioPlan>`	Servicios iniciales que se consideran durante la optimizacion
maximoMinutos, `int`	Minutos máximos que se pueden mover los servicios
mejorCandidata, `List<ServicioPlan>`	Lista de servicios que es la mejor entre la candidata y la *solucionActual*
mejorSolucoinVecindario, `List<ServicioPlan>`	Lista de servicios que es la mejor solución que proporciona el vecindario
auxiliaresMejorSolucionVecindario, `List<Auxiliar>`	Lista de auxiliares que realizan los servicios de la *mejorSolucionVecindario*
mejoraReplanificar, `int`	Contabilizador del número de veces que el vecino generado por el movimiento *Replanificar* es mejor que la *solucionActual*
mejoraDesplazar, `int`	Contabilizador del número de veces que el vecino generado por el movimiento *Desplazar* es mejor que la *solucionActual*
mejoraIntercambiar, `int`	Contabilizador del número de veces que el vecino generado por el movimiento *Intercambiar* es mejor que la *solucionActual*
mejoraIntercambiarFuera, `int`	Contabilizador del número de veces que el vecino generado por el movimiento *IntercambiarFuera* es mejor que la *solucionActual*
mejoraIntercambiarFranja, `int`	Contabilizador del número de veces que el vecino generado por el movimiento *IntercambiarFranja* es mejor que la solucionActual
mejoraPasar, `int`	Contabilizador del número de veces que el vecino generado por el movimiento *PasarFuera* es mejor que la *solucionActual*
vecinosReplanificar, `int`	Contabilizador del número de vecinos generados por el movimiento *Replanificar*
vecinosDesplazar, `int`	Contabilizador del número de vecinos generados por el movimiento *Desplazar*

vecinosIntercambiar, `int`	Contabilizador del número de vecinos generados por el movimiento *Intercambiar*
vecinosIntercambiarFuera, `int`	Contabilizador del número de vecinos generados por el movimiento *IntercambiarFuera*
vecinosntercambiarFranja, `int`	Contabilizador del número de vecinos generados por el movimiento *IntercambiarFranja*
vecinosPasar, `int`	Contabilizador del número de vecinos generados por el movimiento
Optimo, `List<ServicioPlan>`	Mejor solución obtenida

Algoritmo A.3: Optimización

```
1  Inicio
   Paso 1.- Iniciar la solución candidata y la solución actual
2  candidata ← solucionInicial
3  candidataOrdenada ← ordenar(candidata)
4  auxiliaresCandidata ← auxiliares(candidata)
5  solucionActual ← solucionInicial
6  auxiliaresSolucionActual ← auxiliares(solucionActual)
7  optimo ← solucionInicial
   Paso 2.- Definir los parámetros para actualizar el movimiento
8  movReplanificar ← 15
9  movDesplazar ← 15
10 movIntercambiarDentro ← 15
11 movIntercambiarFranja ← 15
12 movIntercambiarFuera ← 15
13 movPasarFuera ← 15
14 movDesplDos ← 15
15 movInterDespl ← 15
16 sumaTotal ← movReplanificar + movDesplazar + movIntercambiarDentro +
    movIntercambiarFranja + movIntercambiarFuera + movPasarFuera + movDesplDos +
    movInterDespl
   Paso 3.- Obtener los servicios útiles (servicios que se quieren considerar
   para calcular los descansos, solapamientos y ficticias durante la
   optimización)
17 if soloDiasServicioNuevo = true o  soloServicioNuevo = true o  soloReplanificables = true
    then
18 |  serviciosInicialesUtiles ← servicios Utiles(solucionInicial, soloServicioNuevo,
   |    soloDiasServicioNuevo, soloReplanificables)
19 else
20 |  serviciosInicialesUtiles ← solucionInicial
21 end
   Paso 4.- Si no hay descansos, solapamientos o auxiliares ficticias devolver la
   solución inicial
22 solapamientos ← solapamiento(serviciosInicialesUtiles)
23 descansos ← descanso(serviciosInicialesUtiles)
24 penalizacionFicticias ← penalizacionAuxilairesFicticias(serviciosInicialesUtiles)
25 if solapamientos = 0 y descansos = 0 y penalizacionFicticias = 0 then
26 |  return solucionInicial
27 end
```

```
   Paso 5.- Determinar el máximo de minutos que se pueden aplicar a los
   movimientos
28 descansoUtil ← descanso(serviciosInicialesUtiles)
29 solpamientoUtil ← solapamiento(serviciosInicialesUtiles)
30 maximoMinutos ← máximo(descansoUtil, solpamientoUtil)
   Paso 6.- Realizar la optimización
31 mejorCandidata
32 mejorSolucionVecindario
33 auxiliaresMejorSolucionVecindario
34 solucionActualOrdenada ← ordenar(solucionActual)
35 solucionActualOrdenada ← true
36 contadorFinalizador = 0
37 while
38 |  n
39 Iteraciones < nIterMax y continuar = true) Paso a.- Iniciar el contador de los
   vecinos que se generan de cada movimiento y el de los vecinos de cada
   movimiento que mejoran la solución actual
40 mejoraReplanificar, mejoraDesplazar, mejoraIntercambiar, mejoraIntercambiarFuera,
    mejoraIntercambiarFranja, mejoraPasar, vecinosReplanificar, vecinosDesplazar,
    vecinosIntercambiar, vecinosIntercambiarFuera, vecinosIntercambiarFranja, vecinosPasar = 0
41 solapameintoSolucionActual ← solapamiento(solucionActual)
42 if solapameintoSolucionActual ≠ 0 then
43 |  haySolapamientoActual ← true
44 end
   Paso b.- Obtener el vecindario
45 for i ← 0 to L
   |  Paso c.- obtener el movimiento que se va a utilizar para generar el vecino
46 |  movimiento ← generarVecino(movimientos, movReplanificar, movDesplazar,
   |   movIntercambiarDentro, movIntercambiarFranja, movIntercambiarFuera, movPasarFuera,
   |   movDesplDos, movInterDespl)
   |  Paso d.- Obtener los servicios modificables, es decir, servicios que pueden
   |  ser modificados durante la generación de vecinos
47 |  if soloServicioNuevo = true then
48 |  |  serviciosMoficables ← serviciosPlaneadosDia(auxiliar, tramo)
49 |  else
50 |  |  if soloReplanificables = true then
51 |  |  |  serviciosMoficables ← serviciosPlaneadosAuxilaires(serviciosReplanificables)
52 |  |  else
53 |  |  |  if eliminarFicticias = true then
54 |  |  |  |  serviciosMoficables ← serviciosDescansoSolapamiento(solucionActual)
55 |  |  |  else
56 |  |  |  |  serviciosMoficables ←
   |  |  |  |   serviciosDescansoSolapamientoFicticias(solucionActual)
57 |  |  |  end
58 |  |  end
59 |  end
   |  Paso e.- Generar el vecino a partir de la solución actual y del movimiento,
   |  seleccionar las auxiliares que intervienen en él y ordenar los servicios de
   |  la solución candidata
```

```
60
61  candidata ← generarVecino(solucionActual, movimiento)
62  auxiliaresCandidata ← auxiliares(candidata)
63  candidataOrdenada ← auxiliares(candidata)
    Paso f.- Evaluar la candidata y la solución auxiliar en la función objetivo
    y obtener la que mejor valor proporciona
64  if funcionV1 = true then
65      mejorCandidata ← funcionObjetivoLexicograficaV1(candidataOrdenada,
          solucionActualOrdenada)
66  else
67      mejorCandidata ← funcionObjetivoLexicograficaV2(candidataOrdenada,
          solucionActualOrdenada)
68  end
    Paso g.- Si es mejor la solución candidata actualizar el contador del
    movimiento utilizado para generarla.
69  if mejorCandidata = candidatOrdenada y candidataOrdenada ≠ solucionActual then
70      if movimiento = ReplanificarServicio then
71          mejoraReplanificar ← mejoraReplanificar +1
72      else if movimiento = DesplazarServicios then
73          mejoraDesplazar ← mejoraDesplazar +1
74      else if movimiento = IntercambiarDentro then
75          mejoraIntercambiar ← mejoraIntercambiar +1
76      else if movimiento = IntercambiarFuera then
77          mejoraIntercambiarFuera ← mejoraIntercambiarFuera +1
78      else if movimiento = IntercambiarFranja then
79          mejoraIntercambiarFranja ← mejoraIntercambiarFranja +1
80      else movimiento = PasarServicioFuera
81          mejoraPasar ← mejoraPasar +1
82      end
83  end
    Paso h.- Actualizar el contador del movimiento utilizado para generar la
    solución candidata
84  if movimiento = ReplanificarServicio then
85      vecinosReplanificar ← vecinosReplanificar +1
86  else if movimiento = DesplazarServicios then
87      vecinosDesplazar ← vecinosDesplazar +1
88  else if movimiento = IntercambiarDentro then
89      vecinosIntercambiar ← vecinosIntercambiar +1
90  else if movimiento = IntercambiarFuera then
91      vecinosIntercambiarFuera ← vecinosIntercambiarFuera +1
92  else if movimiento = IntercambiarFranja then
93      vecinosIntercambiarFranja ← vecinosIntercambiarFranja +1
94  else movimiento = PasarServicioFuera
95      vecinosPasar ← mejoraPasar +1
96  end
```

```
97
    Paso i.- Actualizar la mejor solución del vecindario
98  if mejorSolucionVecindario = ∅ then
99      mejorSolucionVecindario ← candidataOrdenada
100     auxiliaresMejorSolucionVecindario ← auxiliaresCandidata
101 else
102     if funcionV1 = true then
103         mejorSolucionVecindario ←
            funcionObjetivoLexicograficaV1(candidataOrdenada,
            mejorSolucionVecindario)
104     else
105         mejorSolucionVecindario ←
            funcionObjetivoLexicograficaV2(candidataOrdenada,
            mejorSolucionVecindario)
106     end
107 end
    Paso j.- Comprobar si el mejor vecino es óptimo, en cuyo caso se detiene la
    obtención de vecinos
108 if soloDiasServicioNuevo = true o soloServicioNuevo = true o soloReplanificables = true
109     solucionUtil ← serviciosUtiles(mejorSolucionVecindario, soloServicioNuevo,
          soloDiasServicioNuevo, soloReplanificables)
110     solapamientosUtiles ← solapamiento(solucionUtil)
111     descansosUtiles ← descanso(solucionUtil)
112     penalizacionFicticiasUtiles ← penalizacionAuxilairesFicticias(solucionUtil)
113     if eliminarFicticia = false then
114         if haySolapamientoSolucionActual = false then
115             if solapamientosUtiles = 0 then
116                 Pasar al siguiente vecindario
117             end
118         else
119             if descansosUtiles = 0 then
120                 Pasar al siguiente vecindario
121             end
122         end
123     else
124         if haySolapamientoSolucionActual = true then
125             if solapamientosUtiles = 0 y penalizacionFicticiasUtiles = 0 then
126                 Pasar al siguiente vecindario
127             end
128         else
129             if solapamientosUtiles = 0 y penalizacionFicticiasUtiles = 0 y descansosUtiles
                  = 0 then
130                 Pasar al siguiente vecindario
131             end
132         end
133     end
```

```
134
135   else
136     solapamientosSolucion ← solapamiento(mejorSolucionVecindario)
137     descansosSolucion ← descanso(mejorSolucionVecindario)
138     penalizacionFicticiasSolucion ←
          penalizacionAuxilairesFicticias(mejorSolucionVecindario)
139     if eliminarFicticia = false then
140       if haySolapamientoSolucionActual = false then
141         if solapamientosSolucion = 0 then
142           Pasar al siguiente vecindario
143         end
144       else
145         if descansosSolucion = 0 then
146           Pasar al siguiente vecindario
147         end
148       end
149     else
150       if haySolapamientoSolucionActual = true then
151         if solapamientosSolucion = 0 y penalizacionFicticiasSolucion = 0 then
152           Pasar al siguiente vecindario
153         end
154       else
155         if solapamientosSolucion = 0 y penalizacionFicticiasSolucion = 0 y
              descansosSolucion = 0 then
156           Pasar al siguiente vecindario
157         end
158       end
159     end
160   fin
      Paso k.- Obtener la diferencia entre la mejor solución del vecindario y la
      solución actual, evaluar la solución actual y la mejor solución del
      vecindario en la función objetivo para obtener la que toma mejor valor
161   diferencia ← diferenciaEntreSoluciones((mejorSolucionVecindario, solucionActual)
162   mejora ← false
163   if funcionV1 = true then
164     mejor ← funcionObjetivoLexicograficaV1(mejorSolucionVecindario,
          solucionActual)
165   else
166     mejor ← funcionObjetivoLexicograficaV2(mejorSolucionVecindario,
          solucionActual)
167   end
      Paso l.- Actualizar la solución actual (si la candidata es mejor se
      actualiza, si son iguales o si la candidata es peor se actualiza con cierta
      probabilidad)
168   if mejorSolucionVecindario = solucionActual then
169     solucionActual ← mejorSolucionVecindario
170   else if mejor = mejorSolucionVecindario then
171     solucionActual ← mejor
172     mejora ← true
```

```
173
174     else movimiento = ?PasarServicioFuera?
175         x ← numeroAleatorio(0,1)
176         if x < probAcaptacion(diferencia) then
177             solucionActual ← mejorSolucionVecindario
178         else
179             solucionActual ← solucionActualOrdenada
180         end
181     end
        Paso m.- Actualizar la función objetivo en caso de que sea necesario
182     if cambiarFuncionObjetivo = true then
183         if se llevan varias iteraciones sin mejorar el objetivo y funcionV1 = true then
184             funcionV1 ← false
185             funcionV2 ← true
186         end
187         if se llevan varias iteraciones sin mejorar el objetivo y funcionV2 = true then
188             funcionV1 ← true
189             funcionV2 ← false
190         end
191     end
        Paso n.- Actualizar la solución actual ordenada, el óptimo, el número de
        iteraciones, la temperatura
192     solucionActualOrdenada ← ordena(solucionActual)
193     optimo ← solucionActualOrdenada
194     nIteraciones ← nIteraciones +1
195     temperatura ← enfria(tempInicial, parametroEnfriamiento, nIteraciones) Paso ñ.-
        Comprobar si hay que finalizar el algoritmo, es decir, si la solución es la
        óptima o si se alcanza el número máximo de iteraciones sin mejora
196     if soloDiasServicioNuevo = true o soloServicioNuevo = true o soloReplanificables = true
197         solucionUtil ← serviciosUtiles(solucionActualOrdenada, soloServicioNuevo,
              soloDiasServicioNuevo, soloReplanificables)
198         solapamientosUtiles ← solapamiento(solucionUtil)
199         descansosUtiles ← descanso(solucionUtil)
200         penalizacionFicticiasUtiles ← penalizacionAuxilairesFicticias(solucionUtil)
201         if eliminarFicticia = false then
202             if solapamientosUtiles = 0 y descansosUtiles = 0 then
203                 continuar ← false
204             end
205         else
206             if solapamientosUtiles = 0 y penalizacionFicticiasUtiles = 0 y descansosUtiles = 0
                  then
207                 continuar ← false
208             end
209         end
```

```
211
212   else
213   |  solapamientosSolucion ← solapamiento(solucionActualOrdenada)
214   |  descansosSolucion ← descanso(solucionActualOrdenada)
215   |  penalizacionFicticiasSolucion ←
      |   penalizacionAuxilairesFicticias(solucionActualOrdenada)
216   |  if eliminarFicticia = false then
217   |  |  if solapamientosSolucion = 0 y descansosSolucion = 0 then
218   |  |  |  continuar ← false
219   |  |  end
220   |  else
221   |  |  if solapamientosSolucion = 0 y penalizacionFicticiasSolucion = 0 y
      |  |   descansosSolucion = 0 then
222   |  |  |  continuar ← false
223   |  |  end
224   |  end
225   fin
      Paso o.- Actualizar los contabilizadores de no mejora (finalizador), si se
      llega al máximo detener el algoritmo
226   if mejora = false then
227   |  contadorFinalizador ← contadorFinalizador +1
228   else
229   |  contadorFinalizador ← 0
230   end
231   if contadorFinalizador = iterMaxNoMejora then
232   |  continuar ← false
233   end
      Paso p.- Obtener los servicios útiles de la solución actual y actualizar
      los minutos que se pueden utilizar en los movimientos
234   if soloDiasServicioNuevo = true o  soloServicioNuevo = true o  soloReplanificables = true
       then
235   |  solucionUtil ← serviciosUtiles(solucionActualOrdenada, soloServicioNuevo,
      |   soloDiasServicioNuevo, soloReplanificables)
236   |  solapamientosUtiles ← solapamiento(solucionUtil)
237   |  descansosUtiles ← descanso(solucionUtil)
238   |  maximoMinutos ← máximo(solapamientosUtiles,descansosUtiles)
239   else
240   |  solapamientosSolucion ← solapamiento(solucionActualOrdenada)
241   |  descansosSolucion ← descanso(solucionActualOrdenada)
242   |  maximoMinutos ← máximo(solapamientosSolucion,descansosSolucion)
243   end
      Paso q.- Actualizar el parámetro asignado a cada movimiento, de modo que
      aumenta el del movimiento que mejores vecinos haya proporcionado
244   proporcionReplanificar ← 0
245   proporcionDesplazar ← 0
246   proporcionIntercambiar ← 0
247   uproporcionIntercambiarFuera ← 0
248   proporcionIntercambiarFranja ← 0
249   proporcionPasar ← 0
```

```
250
251   if vecinosReplanificar > 0 then
252   |  proporcionReplanificar ← mejoraReplanificar / vecinosReplanificar
253   end
254   if vecinosDesplazar > 0 then
255   |  proporcionDesplaza ← mejoraDesplaza / vecinosDesplaza
256   end
257   if vecinosIntercambiar > 0 then
258   |  proporcionIntercambiar ← mejoraIntercambiar / vecinosIntercambiar
259   end
260   if vecinosIntercambiarFuera > 0 then
261   |  proporcionFuera ← mejoraFuera / vecinosFuera
262   end
263   if vecinosIntercambiarFranja > 0 then
264   |  proporcionntercambiarFranja ← mejorantercambiarFranja / vecinosntercambiarFranja
265   end
266   if vecinosPasar > 0 then
267   |  proporcionPasar ← mejoraPasar / vecinosPasar
268   end
269   if las proporcones son distintas then
270   |  if proporcionReplanificar > las demás proporciones then
271   |  |  movReplanificar ← movReplanificar +1
272   |  else if proporcionDesplazar > las demás proporciones then
273   |  |  movDesplazar ← movDesplazar +1
274   |  else if proporcionIntercambiar > las demás proporciones then
275   |  |  movIntercambiar ← movIntercambiar +1
276   |  else if proporcionIntercambiarFuera > las demás proporciones then
277   |  |  movIntercambiarFuerar ← movIntercambiarFuerar +1
278   |  else if proporcionIntercambiarFranja > las demás proporciones then
279   |  |  movIntercambiarFranja ← movIntercambiarFranja +1
280   |  else if proporcionPasar > las demás proporciones
281   |  |  movPasar ← movPasar +1
282   |  end
283   end
284   sumaTotal ← movReplanificar + movDesplazar + movIntercambiarDentro +
      |  movIntercambiarFranja + movIntercambiarFuera + movPasarFuera +
      |  movDesplDos + movInterDespl
    Paso 7.- Devolver la planificación optimizada
285 return optimo
```

A.4. Movimientos

En este apartado se describen los diferentes movimientos utilizados durante la optimización, todos estos movimientos consisten en realizar diferentes tipos de modificaciones a una planificación inicial.

Para cada movimiento, en primer lugar, se describen las variables de entrada de la función y, seguidamente, se presentan aquellas que se utilizan internamente en ella. Por último, se presenta el pseudocódigo de la función, explicando de los pasos que se están siguiendo.

A.4.1. Generar vecino

Esta es la función que se utiliza para generar un vecino (planificación tras aplicarle un movimiento) a partir de la planificación actual y del movimiento que se quiere utilizar.

Nombre y clase	Descripción
Variables de entrada	
solucionCandidata, `List<ServicioPlan>`	Lista de servicios
serviciosModificables, `List<ServicioPlan>`	Lista de servicios que se pueden modificar
movimiento, `String`	Movimiento a utilizar
Variables que se utilizan en la función	
vecinoFinal, `List<ServicioPlan>`	Solución modificada

Algoritmo A.4: Generar vecino

```
1  Inicio
   Paso 1.- Iniciar la solución final y el número máximo de iteraciones por
   movimiento
2  vecinoFinal
3  maximoItr
4  factible ← false Paso 2.- Generar el vecino
5  while factible = false y iter < maximoIter do
6  |  if movimiento = ReplanificarServicio then
7  |  |  vecinoFinal ← replanificarServicio(solucionCandidata, serviciosModificables)
8  |  else if movimiento = DesplazarServicios then
9  |  |  vecinoFinal ← desplazarServicios(solucionCandidata, serviciosModificables)
10 |  else if movimiento = IntercambiarDentro then
11 |  |  vecinoFinal ← intercambiarDentro(solucionCandidata, serviciosModificables)
12 |  else if movimiento = IntercambiarFuera then
13 |  |  vecinoFinal ← intercambiarFuera(solucionCandidata, serviciosModificables)
14 |  else if movimiento = IntercambiarFranja then
15 |  |  vecinoFinal ← imtercambiarFranja(solucionCandidata, serviciosModificables)
16 |  else if movimiento = PasarServicioFuera then
17 |  |  vecinoFinal ← pasarServicioFuera(solucionCandidata, serviciosModificables)
18 |  else if movimiento = DesplazarDos then
19 |  |  vecinoFinal ← desplazarDos(solucionCandidata, serviciosModificables)
20 |  else movimiento = IntercambiarDesplazar
21 |  |  vecinoFinal ← intercambiarDesplazar(solucionCandidata, serviciosModificables)
22 |  end
23 end
   Paso 3.- Comprobar si se respetan las ventanas, los vetos y las jornadas
24 factVentanas ← factibilidadVentanas(vecinoFinal)
25 factAfinidad ← factibilidadafinidades(vecinoFinal)
26 factHoras ← factibilidadHorass(vecinoFinal)
27 if factVentanas = true y factAfinidad = true y factHoras = true then
28 |  factible = true
29 end
30 iter ← iter +1 Paso 4.- Devolver el vecino
31 return vecinoFinal
```

A.4.2. Replanificar un servicio

El movimiento *replanificar un servicio* se utiliza para modificar el horario en que se debe realizar un servicio. De modo que dado un servicio se añade a la hora de inicio, y a la de fin, cierta cantidad de minutos, de forma que si dicha cantidad es positiva el servicio se retrasa y si es positiva se adelanta.

Nombre y clase	Descripción
Variables de entrada	
solucionCandidata, `List<ServicioPlan>`	Lista de servicios
serviciosModificables, `List<ServicioPlan>`	Lista de servicios que se pueden modificar
Variables que se utilizan en la función	
serviciosNuevos, `List<ServicioPlan>`	Lista de servicios que se usan para aplicar el movimiento
Servicio, `ServicioPlan`	Servicio que va a ser modificado
Minutos, `int`	Minutos que se va a retrasar o adelantar el horario del servicio
vecinoFinal, `List<ServicioPlan>`	Solución modificada
horaInicioServicio, `Calendar`	Hora de inicio del servicio
horaFinServicio, `Calendar`	Hora de fin del servicio

Algoritmo A.5: Replanificar un servicio

```
1  Inicio
   Paso 1.- Seleccionar los servicios que se pueden modificar
2  x ← numeroAleatorio(0, 1)
3  if x < 0,9 then
4  |  serviciosNuevos ← serviciosSolapamientoDescanso(solucionCandidata)
5  else
6  |  serviciosNuevos ← serviciosModificables
7  end
8  vecinoFinal ← solucionCandidata
9  if serviciosNuevos ≠ ∅
   |  Paso 2.- Seleccionar el servicio a replanificar
10 |  servicio ← servicioAleatorio(serviciosNuevos)
   |  Paso 3.- Si solo se tienen descansos elegir los minutos para eliminarlos
   |  (esto se hace con cierta probabilidad, en los demás casos se eligen los
   |  minutos de forma aleatoria)
11 |  descanso ← descansos(serviciosNuevos)
12 |  solapamiento ← solapamientos(serviciosNuevos)
13 |  if solpamiento = 0
14 |  |  if Si el servicio es el primero que genera descanso
15 |  |  |  y ← numeroAleatorio(0, 1)
16 |  |  |  if x < 0,7
   |  |  |  |  Paso a.- Tomar los minutos como el descanso entre los servicios
17 |  |  |  |  servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
18 |  |  |  |  minutos ← (inicio de servicioPosterior − fin de servicio) − minutos de
   |  |  |  |    desplazamiento entre servicio y servicioPosterior
```

```
19
20
21
22          else if x ≥ 0,7 y x < 0,9
                Paso b.- Tomar los minutos como los necesarios para hacer que el
                descanso entre los servicios sea mayor que el descanso máximo (ya
                que dicho descanso no se cuenta)
23              servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
24              descansoActual ← tiempoDescanso(servicio, servicioPosterior)
25              minutos ← descansoMaximo − descansoActual
26          else
                Paso c.- Tomar los minutos de forma aleatoria
27              minutos ← minutos al azar
28          end
29      else if Si el servicio es el último que genera descanso
30          y ← numeroAleatorio(0, 1)
31          if x < 0,7 then
                Paso a.- Tomar los minutos como el descanso entre los servicios
32              servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
33              minutos ← (inicio de servicio − fin de servicioPrevio) − minutos de
                  desplazamiento entre servicio y servicioPrevio
34          else if x ≥ 0,7 y x < 0,9 then
                Paso b.- Tomar los minutos como los necesarios para hacer que el
                descanso entre los servicios sea mayor que el descanso máximo (ya
                que este descanso no se tiene en cuenta)
35              servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
36              descansoActual ← tiempoDescanso(servicioprevio, servicio)
37              minutos ← descansoMaximo − descansoActual
38          else
                Paso c.- Tomar los minutos de forma aleatoria
39              minutos ← minutos al azar
40          end
41      else
            Paso d.- Tomar los minutos de forma aleatoria
42          minutos ← minutos al azar
43      end
44  else
        Paso 4.- Si se tienen solapamientos elegir los minutos para eliminarlos
        (esto se hace con cierta probabilidad, en los demás casos se eligen los
        minutos de forma aleatoria)
45      if Si el servicio es el primero que genera solapamiento
46          y ← numeroAleatorio(0, 1)
47          if x < 0,9 then
                Paso a.- Tomar los minutos como el solapamiento entre los servicios
48              servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
49              minutos ← (fin de servicio − inicio de servicioPrevio) + minutos de
                  desplazamiento entre servicio y servicioPosterior
50          else
                Paso b.- Tomar los minutos de forma aleatoria
51              minutos ← minutos al azar
52          end
```

```
53
54
55      else if Si el servicio es el último que genera solapamiento
56          y ← numeroAleatorio(0, 1)
57          if x < 0,9 then
                Paso a.- Tomar los minutos como el solapamiento entre los servicios
58              servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
59              minutos ← (fin de servicioPrevio − inicio de servicio) + minutos de
                  desplazamiento entre servicioPrevio y servicio
60          else
                Paso b.- Tomar los minutos de forma aleatoria
61              minutos ← minutos al azar
62          end
63      else
            Paso c.- Tomar los minutos de forma aleatoria
64          minutos ← minutos al azar
65      end
66  end
    Paso 5.- Añadir al horario del servicio los minutos obtenidos
67  horaInicioServicio ← horaInicioServicio + minutos
68  horaFinServicio ← horaFinServicio + minutos
69 end
70 return vecinoFinal
```

A.4.3. Desplazar varios servicios

El movimiento *desplazar servicios* consiste en adelantar o retrasar el inicio (y por consiguiente el fin) de varios servicios consecutivos, para ello se parte de un servicio y se añade cierta cantidad de minutos a todos los servicios anteriores o posteriores a él.

Nombre y clase	Descripción
Variables de entrada	
solucionCandidata, `List<ServicioPlan>`	Lista de servicios
serviciosModificables, `List<ServicioPlan>`	Lista de servicios que se pueden modificar
Variables que se utilizan en la función	
serviciosNuevos, `List<ServicioPlan>`	Lista de servicios que se usan para aplicar el movimiento
Servicio, `ServicioPlan`	Servicio que va a ser modificado
Minutos, `int`	Minutos que se va a retrasar o adelantar el horario del servicio
sentido, `boolean`	Parámetro que determina si se van a despasar los servicios anteriores a servicio (true) o los servicios posteriores (false)
*vecinoFina*l, `List<ServicioPlan>`	Solución modificada
horaInicioServicio, `Calendar`	Hora de inicio del servicio
horaFinServicio, `Calendar`	Hora de fin del servicio

Algoritmo A.6: Desplazar servicios

```
1  Inicio
   Paso 1.- Seleccionar los servicios que se pueden modificar
2  x ← numeroAleatorio(0, 1)
3  if x < 0,9 then
4  |  serviciosNuevos ← serviciosSolapamientoDescanso(solucionCandidata)
5  else
6  |  serviciosNuevos ← serviciosModificables
7  end
8  vecinoFinal ← ∅
9  if serviciosNuevos ≠ ∅
   |  Paso 2.- Seleccionar el servicio pivote
10 |  servicio ← servicioAleatorio(serviciosNuevos)
11 |  minutos ← minutos aleatorios
12 |  sentido ← sentido aleatorio
   |  Paso 3.- Si solo se tienen descansos elegir los minutos para eliminarlos
   |  (esto se hace con cierta probabilidad, en los demás casos se eligen los
   |  minutos de forma aleatoria)
13 |  descanso ← descansos(serviciosNuevos)
14 |  solapamiento ← solapamientos(serviciosNuevos)
15 |  if solpamiento = 0
16 |  |  if Si el servicio es el primero que genera descanso
17 |  |  |  y ← numeroAleatorio(0, 1)
18 |  |  |  if x < 0,7
   |  |  |  |  Paso a.- Tomar los minutos como el descanso entre los servicios
19 |  |  |  |  servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
20 |  |  |  |  minutos ← (inicio de servicioPosterior − fin de servicio) − minutos de
   |  |  |  |    desplazamiento entre servicio y servicioPosterior sentido ← true
21 |  |  |  else if x ≥ 0,7 y x < 0,9
   |  |  |  |  Paso b.- Tomar los minutos como los necesarios para hacer que el
   |  |  |  |  descanso entre los servicios sea mayor que el descanso máximo
22 |  |  |  |  servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
23 |  |  |  |  descansoActual ← tiempoDescanso(servicio, servicioPosterior)
24 |  |  |  |  minutos ← descansoMaximo − descansoActual sentido ← true
25 |  |  |  else
   |  |  |  |  Paso c.- Tomar los minutos de forma aleatoria
26 |  |  |  |  minutos ← minutos al azar sentido ← true
27 |  |  |  end
28 |  |  else if Si el servicio es el último que genera descanso
29 |  |  |  y ← numeroAleatorio(0, 1)
30 |  |  |  if x < 0,7 then
   |  |  |  |  Paso a.- Tomar los minutos como el descanso entre los servicios
31 |  |  |  |  servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
32 |  |  |  |  minutos ← (inicio de servicio − fin de servicioPrevio) − minutos de
   |  |  |  |    desplazamiento entre servicio y servicioPrevio
33 |  |  |  |  sentido ← false
34 |  |  |  else if x ≥ 0,7 y x < 0,9 then
   |  |  |  |  Paso b.- Tomar los minutos como los necesarios para hacer que el
   |  |  |  |  descanso entre los servicios sea mayor que el descanso máximo
35 |  |  |  |  servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
36 |  |  |  |  descansoActual ← tiempoDescanso(servicioprevio, servicio)
37 |  |  |  |  minutos ← descansoMaximo − descansoActual
38 |  |  |  |  sentido ← false
```

```
39
40  |
41  |   |
42  |   |   |   else
    |   |   |   |   Paso c.- Tomar los minutos de forma aleatoria
43  |   |   |   |   minutos ← minutos al azar sentido ← false
44  |   |   |   end
45  |   |   else
    |   |   |   Paso c.- Tomar los minutos de forma aleatoria
46  |   |   |   minutos ← minutos al azar sentido ← false
47  |   |   end
48  |   else
    |   |   Paso 4.- Si se tienen solapamientos elegir los minutos para eliminarlos
    |   |   (esto se hace con cierta probabilidad, en los demás casos se eligen los
    |   |   minutos de forma aleatoria)
49  |   |   if Si el servicio es el primero que genera solapamiento
50  |   |   |   y ← numeroAleatorio(0, 1)
51  |   |   |   if y < 0,9 then
    |   |   |   |   Paso a.- Tomar los minutos como el solapamiento entre los servicios
52  |   |   |   |   servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
53  |   |   |   |   minutos ← (fin de servicio − inicio de servicioPrevio) + minutos de
    |   |   |   |     desplazamiento entre servicio y servicioPosterior
54  |   |   |   |   sentido ← true
55  |   |   |   else
    |   |   |   |   Paso b.- Tomar los minutos de forma aleatoria
56  |   |   |   |   minutos ← minutos al azar sentido ← true
57  |   |   |   end
58  |   |   else if Si el servicio es el último que genera solapamiento
59  |   |   |   y ← numeroAleatorio(0, 1)
60  |   |   |   if x < 0,9 then
    |   |   |   |   Paso a.- Tomar los minutos como el solapamiento entre los servicios
61  |   |   |   |   servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
62  |   |   |   |   minutos ← (fin de servicioPrevio − inicio de servicio) + minutos de
    |   |   |   |     desplazamiento entre servicioPrevio y servicio
63  |   |   |   |   sentido ← false
64  |   |   |   else
    |   |   |   |   Paso b.- Tomar los minutos de forma aleatoria
65  |   |   |   |   minutos ← minutos al azar sentido ← false
66  |   |   |   end
67  |   |   else
68  |   |   |   y ← numeroAleatorio(0, 1)
69  |   |   |   if y < 0,3 then
    |   |   |   |   Paso a.- Separar del servicio siguiente
70  |   |   |   |   servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
71  |   |   |   |   minutos ← (fin de servicio − inicio de servicioPrevio) + minutos de
    |   |   |   |     desplazamiento entre servicio y servicioPosterior
72  |   |   |   |   sentido ← true
```

```
73
74  |
75  |   |
76  |   |   else if y < 0,6 then
            |   Paso b.- Separar del servicio siguiente
77  |   |   |   servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
78  |   |   |   minutos ← (fin de servicioPrevio − inicio de servicio) + minutos de
            |     desplazamiento entre servicioPrevio y servicio
79  |   |   |   sentido ← false
80  |   |   else if y < 0,9 then
            |   Paso c.- Tomar los minutos de forma aleatoria
81  |   |   |   minutos ← minutos al azar
82  |   |   else
            |   Paso d.- Modificar el horario en que se realizan los servicios
            |   siguientes al servicio dado
83  |   |   |   servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
84  |   |   |   minutos ← (fin de servicio − inicio de servicioPosterior) + minutos de
            |     desplazamiento entre servicio y servicioPosterior
85  |   |   |   foreach servicioPosterior a servicio do
86  |   |   |   |   horaInicioServicioPosterior ← horaInicioServicioPosterior + minutos
87  |   |   |   |   horaFinServicioPosterior ← horaFinServicioPosterior + minutos
88  |   |   |   end
89  |   |   |   servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
90  |   |   |   minutos ← (fin de servicioPrevio − inicio de servicio) + minutos de
            |     desplazamiento entre servicioPrevio y servicio
91  |   |   |   foreach servicioPrevio a servicio do
92  |   |   |   |   horaInicioServicioPrevio ← horaInicioServicioPrevio + minutos
93  |   |   |   |   horaFinServicioPrevio ← horaFinServicioPrevio + minutos
94  |   |   |   end
95  |   |   |   vecinoFinal ← solucionCandidata con los servicios recién modificados
96  |   |   end
97  |   end
98  end
        Paso 5.- Modificar el horario en que se realizan los servicios según el
        sentido y los minutos obtenidos anteriormente (si no se ha hecho antes)
99  if vecinoFinal = ∅ then
100 |   if sentido = true then
101 |   |   foreach servicioPosterior a servicio do
102 |   |   |   horaInicioServicioPosterior ← horaInicioServicioPosterior + minutos
103 |   |   |   horaFinServicioPosterior ← horaFinServicioPosterior + minutos
104 |   |   end
105 |   else
106 |   |   foreach servicioPrevio a servicio do
107 |   |   |   horaInicioServicioPrevio ← horaInicioServicioPrevio + minutos
108 |   |   |   horaFinServicioPrevio ← horaFinServicioPrevio + minutos
109 |   |   end
110 |   end
111 |   vecinoFinal ← solucionCandidata con los servicios recién modificados
112 end
113 else
114 |   vecinoFinal ← solucionCandidata
115 end
116 return vecinoFinal
```

A.4.4. Intercambiar servicios de una auxiliar

El movimiento *intercambiar servicios de una auxiliar* se basa en modificar el horario en que una auxiliar realiza dos servicios, de modo que se intercambian los inicios y se adaptan los finales según la duración de cada uno de ellos.

Nombre y clase	Descripción
Variables de entrada	
solucionCandidata, `List<ServicioPlan>`	Lista de servicios
serviciosModificables, `List<ServicioPlan>`	Lista de servicios que se pueden modificar
Variables que se utilizan en la función	
serviciosNuevos, `List<ServicioPlan>`	Lista de servicios que se usan para aplicar el movimiento
Servicio1, `ServicioPlan`	Servicio al que se le intercambiará el horario
Servicio2, `ServicioPlan`	Servicio al que se le intercambiará el horario
horaInicioServicio1, `Calendar`	Hora de inicio del *servicio1*
horaFinServicio1, `Calendar`	Hora de fin del *servicio1*
duracionServicio1, `int`	Duración en minutos del *servicio1*
horaInicioServicio2, `Calendar`	Hora de inicio del *servicio2*
horaFinServicio2, `Calendar`	Hora de fin del *servicio2*
duracionServicio2, `int`	Duración en minutos del *servicio2*
vecinoFinal, `List<ServicioPlan>`	Solución modificada

Algoritmo A.7: Intercambiar servicios de una auxiliar

```
1  Inicio
   Paso 1.- Seleccionar los servicios que se pueden modificar
2  x ← numeroAleatorio(0, 1)
3  if x < 0,9 then
4  |  serviciosNuevos ← serviciosSolapamientoDescanso(solucionCandidata)
5  else
6  |  serviciosNuevos ← serviciosModificables
7  end
8  vecinoFinal ← ∅
9  if serviciosNuevos ≠ ∅
   |  Paso 2.- Seleccionar los servicios que se van a intercambiar
10 |  servicio1 ← servicioAleatorio(serviciosNuevos)
11 |  serviciosAuxDia ← serviciosAuxiliarDia(auxiliar(servicio1),
   |    dia(servicio1),serviciosNuevos)
12 |  y ← numeroAleatorio(0, 1)
13 |  if y < 0,2 then
14 |  |  serviciosAuxDia ← serviciosSolapamientoDescanso(serviciosAuxDia)
15 |  else
16 |  |  serviciosAuxDia ← serviciosVentanasDisponiblesDistintas(servicio1,
   |  |    serviciosAuxDia)
17 |  end
18 |  serviciosAuxDia ← serviciosAuxDia \ servicio1
```

```
19
    | Paso 3.- Con cierta probabilidad modificar el nuevo horario de inicio de
    | los servicios
20  | y ← numeroAleatorio(0, 1)
21  | if y < 0,9 then
22  |   | replanificar ← true
23  | else
24  |   | replanificar ← false
25  | end
    | Paso 4.- Intercambiar los horarios de inicio y se es necesario replanificar
    | los nuevos horarios
26  | vecinoFinal ← intercambiarServcios(servicio1, servicio2, replanificar)
27 else
28  | vecinoFinal ← solucionCandidata
29 end
   Paso 5.- Devolver los servicios
30 return vecinoFinal
```

Intercambiar servicios y modificar los nuevos horarios en caso de que se requiera.

Algoritmo A.8: Intercambiar el horario de los servicios

```
1  Inicio
2  serviciosOrdenados ← ordenar(servicios)
   Paso 1.- Buscar los servicios anteriores y siguientes a los servicios dados
3  servicioPrevio1 ← servicioAnterior(servicio1, serviciosOrdenados)
4  servicioPosterior1 ← servicioSiguiente(servicio1, serviciosOrdenados)
5  servicioPrevio2 ← servicioAnterior(servicio2, serviciosOrdenados)
6  servicioosterior2 ← servicioSiguiente(servicio2, serviciosOrdenados)
   Paso 2.- Intercambiar las horas de inicio de los servicios y adaptar las horas
   de fin según sus duraciones
7  horaInicioServicio1 ← horaInicioServicio2Original
8  horaFinServicio1 ← horaInicioServicio2Original + duracionServicio1
9  horaInicioServicio2 ← horaInicioServicio1Original
10 horaFinServicio2 ← horaInicioServicio1Original + duracionServicio2
   Paso 3.- Iniciar los minutos que se van a modificar los nuevos horarios de los
   servicios
11 minutos1 ← 0
12 minutos2 ← 0
13 x ← numeroAleatorio(0, 1)
14 if x < 0,3
   | Paso 4.- Tomar minutos1 al azar de entre todos los minutos que se puede
   | mover el servicio1 dentro de su ventana disponible
15 | minutos1 ← minutos al azar en la ventana de servicio1
   | Paso 5.- Tomar minutos2 al azar de entre todos los minutos que se puede
   | mover el servicio2 dentro de su ventana disponible
16 | minutos2 ← minutos al azar en la ventana de servicio2
```

```
17 else
       Paso 6.- Considerar el caso en que el nuevo horario del servicio1 lo coloca
       por delante del servicio2
18     if horaInicioServicio1 es anterior a horaInicioServicio2
19         if servicioPrevio2 ≠ ∅ then
               Paso a.- Si el servicio1 tiene un servicio antes
20             if servicioPrevio2 es anterior a inicioVentanDispServicio1 then
                   Paso b.- Si el servicio anterior no se solapa con la ventana
                   disponible del servicio1, considerar los posibles minutos como la
                   diferencia entre el inicio del servicio1 y el inicio de su ventana
                   disponible
21                 posiblesMinutos1 ← inicioVentanaDispServicio1 − inicioServicio1
22                 minutos1 ← minutos al azar de posiblesMinutos1
23             else
                   Paso c.- Si el servicio anterior se solapa con la ventana
                   disponible del servicio1, considerar como posibles minutos la
                   diferencia entre el inicio del servicio1 y el fin del
                   servicioPrevio
24                 posiblesMinutos1 ← finServicioPrevio2 − inicioServicio1
25                 minutos1 ← minutos al azar de posiblesMinutos1
26             end
27         else
               Paso d.- Si el servicio1 no tiene un servicio anterior, considerar
               como posibles minutos la diferencia entre el inicio del servicio1 y
               el inicio de su ventana disponible
28             posiblesMinutos1 ← inicioVentanaDispServicio1 − inicioServicio1
29             minutos1 ← minutos al azar de posiblesMinutos1
30         end
31         if servicioPosterior2 ≠ ∅ then
               Paso e.- Si el servicio2 tiene un servicio siguiente
32             if servicioPosterior1 es posterior a inicioVentanDispServicio2 then
                   Paso f.- Si el horario del servicio siguiente no se solapa con la
                   ventana disponible del servicio2, considerar como minutos posibles
                   la diferencia entre el final del servicio2 y el final de su ventana
                   disponible
33                 posiblesMinutos2 ← finServicio2 − finVentanaDispServicio2
34                 minutos2 ← minutos al azar de posiblesMinutos2
35             else
                   Paso g.- Si el horario del servicio siguiente se solapa con la
                   ventana disponible del servicio2, considerar como minutos posibles
                   la diferencia entre el fin del servicio2 y el inicio del servicio
                   siguiente
36                 posiblesMinutos2 ← finServicio2 − inicioServicioPosterior1
37                 minutos2 ← minutos al azar de posiblesMinutos2
38             end
39         else
               Paso h.- Si el servicio2 no tiene servicio siguiente, considerar como
               minutos posibles la diferencia entre el fin del servicio2 y el fin de
               su ventana disponible
40             posiblesMinutos2 ← finServicio2 − finVentanaDispServicio2
41             minutos2 ← minutos al azar de posiblesMinutos2
42         end
```

```
43
44
        Paso i.- Si las duraciones de los servicios son distintas, adaptar los
        minutos a la diferencia de duraciones
45      if duracionServicio1 ≠ duracionServicio2 then
46          if duracionServicio2 > duracionServicio1 then
47              minutos2 ← minutos2 − ( duracionServicio2 − duracionServicio1 )
48          else
49              minutos1 ← minutos1 + ( duracionServicio1 − duracionServicio2 )
50          end
51      end
    Paso 7.- Considerar el caso en que el nuevo horario del servicio2 lo coloca
    por delante del servicio1
52  else
53      if servicioPrevio1 ≠ ∅ then
            Paso a.- Si el servicio2 tiene un servicio antes
54          if servicioPrevio1 es anterior a inicioVentanDispServicio2 then
                Paso b.- Si el servicio anterior no se solapa con la ventana
                disponible del servicio2, considerar los posibles minutos como la
                diferencia entre el inicio del servicio2 y el inicio de su ventana
                disponible
55              posiblesMinutos2 ← inicioVentanaDispServicio2 − inicioServicio2
56              minutos2 ← minutos al azar de posiblesMinutos2
57          else
                Paso c.- Si el servicio anterior se solapa con la ventana
                disponible del servicio2, considerar como posibles minutos la
                diferencia entre el inicio del servicio2 y el fin del
                servicioPrevio
58              posiblesMinutos2 ← finServicioPrevio1 − inicioServicio2
59              minutos2 ← minutos al azar de posiblesMinutos2
60          end
61      else
            Paso d.- Si el servicio2 no tiene un servicio anterior, considerar
            como posibles minutos la diferencia entre el inicio del servicio2 y
            el inicio de su ventana disponible
62          posiblesMinutos2 ← inicioVentanaDispServicio2 − inicioServicio2
63          minutos2 ← minutos al azar de posiblesMinutos2
64      end
65      if servicioPosterior2 ≠ ∅
            Paso e.- Si el servicio2 tiene un servicio siguiente
66          if servicioPosterior1 es posterior a inicioVentanDispServicio2
                Paso f.- Si el horario del servicio siguiente no se solapa con la
                ventana disponible del servicio1, considerar como minutos posibles
                la diferencia entre el final del servicio1 y el final de su ventana
                disponible
67              posiblesMinutos1 ← finServicio1 − finVentanaDispServicio1
68              minutos1 ← minutos al azar de posiblesMinutos1
69          else
                Paso g.- Si el horario del servicio siguiente se solapa con la
                ventana disponible del servicio1, considerar como minutos posibles
                la diferencia entre el fin del servicio1 y el inicio del servicio
                siguiente
```

```
70
71      |
72      |   |
73      |   |   |
74      |   |   |   posiblesMinutos1 ← finServicio1 − inicioServicioPosterior2
75      |   |   |   minutos1 ← minutos al azar de posiblesMinutos1
76      |   |   end
77      |   |   else
        |   |   |   Paso h.- Si el servicio1 no tiene servicio siguiente, considerar
        |   |   |   como minutos posibles la diferencia entre el fin del servicio1 y el
        |   |   |   fin de su ventana disponible
78      |   |   |   posiblesMinutos1 ← finServicio1 − finVentanaDispServicio1
79      |   |   |   minutos1 ← minutos al azar de posiblesMinutos1
80      |   |   end
81      |   end
82      end
83  end
    Paso 8.- Añadir al horario del servicio1 los minutos1
84  horaInicioServicio1 ← horaInicioServicio1 + minutos1
85  horaFinServicio1 ← horaFinServicio1 + minutos1
    Paso 8.- Añadir al horario del servicio2 los minutos2
86  horaInicioServicio2 ← horaInicioServicio2 + minutos2
87  horaFinServicio2 ← horaFinServicio2 + minutos2
    Paso 10.- Devolver la lista de servicios ordenados con los servicios
    modificados
88  return serviciosOrdenados con los horarios modificados
```

A.4.5. Intercambiar servicios entre auxiliares

El movimiento *intercambiar servicios entre auxiliares* consiste en intercambiar la auxiliar que realiza los servicios, de modo que dados dos servicios (que deben realizarse el mismo día) la función intercambia sus auxiliares.

Nombre y clase	Descripción
Variables de entrada	
solucionCandidata, `List<ServicioPlan>`	Lista de servicios
serviciosModificables, `List<ServicioPlan>`	Lista de servicios que se pueden modificar
Variables que se utilizan en la función	
serviciosNuevos, `List<ServicioPlan>`	Lista de servicios que se usan para aplicar el movimiento
Servicio1, `ServicioPlan`	Servicio al que se le intercambiará la auxiliar
Servicio2, `ServicioPlan`	Servicio al que se le intercambiará la auxiliar
auxiliarServicio1Original, `Auxiliar`	Auxiliar que atiende al servicio1 en la *solucionCandidata*
auxiliarServicio2Original, `Auxiliar`	Auxiliar que atiende al servicio2 en la *solucionCandidata*

auxiliarServicio1, `Auxiliar`	Nueva auxiliar que atiende al servicio1
auxiliarServicio2, `Auxiliar`	Nueva auxiliar que atiende al servicio2

Algoritmo A.9: Intercambiar servicios entre auxiliares

```
1  Inicio
   Paso 1.- Seleccionar los servicios que se pueden modificar
2  serviciosNuevos ← serviciosModificables
3  if serviciosNuevos ≠ ∅ then
       Paso 2.- Seleccionar el primer servicio a intercambiar
4      servicio1 ← servicioAleatorio(serviciosNuevos)
       Paso 3.- Seleccionar las posibles auxiliares
5      auxMismoNivelAfinidad ← auxiliaresMismoNivelAfinidad(auxiliares, servicio1)
6      auxDesplazamientoGrande ←
         auxiliaresDesplazamientoGrande(auxMismoNivelAfinidad, servicio1)
7      auxMismoNivelAfinidad ← auxMismoNivelAfinidad \ auxDesplazamientoGrande
8      auxiliar ← auxiliarAleatorio(auxMismoNivelAfinidad)
       Paso 4.- Seleccionar los posibles servicios a intercambiar
9      serviciosAuxDia ←
         serviciosAuxiliarDiaAltaAfinidadyBajoDesplazamiento(auxiliar, dia(servicio1))
       Paso 5.- Seleccionar el otro servicio a intercambiar
10     servicio2 ← servicioAleatorio(serviciosAuxDia)
       Paso 6.- Intercambiar las auxiliares de los servicios
11     auxiliarServicio1 ← auxiliarServicio2Original
12     auxiliarServicio2 ← auxiliarServicio1Original
13     vecinoFinal ← solucionCandidata con los servicios modificados
14 else
15     vecinoFinal ← solucionCandidata
16 end
17 return vecinoFinal
```

A.4.6. Cambiar un servicio de auxiliar

El movimiento *cambiar un servicio de auxiliar* consiste en modificar la auxiliar que realiza un servicio. Dado un servicio y una auxiliar, la función devuelve el servicio, pero ahora siendo realizado por la nueva auxiliar.

Nombre y clase	Descripción
Variables de entrada	
solucionCandidata, `List<ServicioPlan>`	Lista de servicios
serviciosModificables, `List<ServicioPlan>`	Lista de servicios que se pueden modificar
Variables que se utilizan en la función	
serviciosNuevos, `List<ServicioPlan>`	Lista de servicios que se usan para aplicar el movimiento
Servicio, `ServicioPlan`	SServicio al que se le intercambiará la auxiliar
auxiliarNueva, `Auxiliar`	Auxiliar que realizará el servicio
auxiliarServicio, `Auxiliar`	Auxiliar que realiza al servicio

Algoritmo A.10: Intercambiar servicios entre auxiliares

```
1  Inicio
   Paso 1.- Seleccionar los servicios que se pueden modificar
2  serviciosNuevos ← serviciosModificables
3  if serviciosNuevos ≠ ∅ then
   |  Paso 2.- Seleccionar el primer servicio a intercambiar
4  |  servicio1 ← servicioAleatorio(serviciosNuevos)
   |  Paso 3.- Seleccionar la nueva auxiliar
5  |  auxiliarNueva ← auxiliarAleatoria(mejoresAuxiliares(servicio))
   |  Paso 4.- Aginar la nueva auxiliar al servicio
6  |  auxiliarServicio ← auxiliarNueva
7  |  vecinoFinal ← solucionCandidata con los servicios modificados
   |  Paso 5.- Con cierta probabilidad se modifica el horario en que se realiza
   |  el servicio
8  |  x ← numeroAleatorio(0, 1)
9  |  if x < 0,6 then
10 |  |  vecinoFinal ← replanificarServicio( vecinoFnal, servicio)
11 |  end
12 else
13 |  vecinoFinal ← solucionCandidata
14 end
15 return vecinoFinal
```

A.4.7. Desplazar doble sentido

El movimiento *desplazar doble sentido* es similar al *movimiento desplazar servicios*, salvo que en este caso se realizan dos desplazamientos en un solo movimiento

Nombre y clase	Descripción
Variables de entrada	
solucionCandidata, `List<ServicioPlan>`	Lista de servicios
serviciosModificables, `List<ServicioPlan>`	Lista de servicios que se pueden modificar
Variables que se utilizan en la función	
serviciosNuevos, `List<ServicioPlan>`	Lista de servicios que se usan para aplicar el movimiento
Servicio, `ServicioPlan`	Servicio que va a ser modificado
Minutos, `int`	Minutos que se va a retrasar o adelantar el horario del servicio
sentido, `boolean`	Parámetro que determina si se van a despasar los servicios anteriores a servicio (true) o los servicios posteriores (false)
vecinoFinal, `List<ServicioPlan>`	Solución modificada
horaInicioServicio, `Calendar`	Hora de inicio del servicio
horaFinServicio, `Calendar`	Hora de fin del servicio

Algoritmo A.11: Desplazar doble sentido

```
1  Inicio
   Paso 1.- Seleccionar los servicios que se pueden modificar
2  x ← numeroAleatorio(0, 1)
3  if x < 0,9 then
4  |  serviciosNuevos ← serviciosSolapamientoDescanso(solucionCandidata)
5  else
6  |  serviciosNuevos ← serviciosModificables
7  end
8  vecinoFinal ← ∅
9  if serviciosNuevos ≠ ∅
   |  Paso 2.- Seleccionar el servicio pivote
10 |  servicio ← servicioAleatorio(serviciosNuevos)
11 |  minutos ← minutos aleatorios
12 |  sentido ← sentido aleatorio
   |  Paso 3.- Si solo se tienen descansos elegir los minutos para eliminarlos
   |  (esto se hace con cierta probabilidad, en los demás casos se eligen los
   |  minutos de forma aleatoria)
13 |  descanso ← descansos(serviciosNuevos)
14 |  solapamiento ← solapamientos(serviciosNuevos)
15 |  if solpamiento = 0
16 |  |  y ← numeroAleatorio(0, 1)
17 |  |  if y < 0,5
18 |  |  |  if servicio no es el último
   |  |  |  |  Paso a.- Si se tienen servicios después del servicio, adelantar los
   |  |  |  |  servicios posteriores
19 |  |  |  |  servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
20 |  |  |  |  z ← numeroAleatorio(0, 1)
21 |  |  |  |  if y < 0,9 then
22 |  |  |  |  |  minutos ← (fin de servicio − inicio de servicioPrevio) + minutos de
   |  |  |  |  |    desplazamiento entre servicio y servicioPosterior
23 |  |  |  |  else
24 |  |  |  |  |  minutos ← minutos aleatorios
25 |  |  |  |  end
26 |  |  |  |  foreach servicioPosterior a servicio do
27 |  |  |  |  |  horaInicioServicioPosterior ← horaInicioServicioPosterior + minutos
28 |  |  |  |  |  horaFinServicioPosterior ← horaFinServicioPosterior + minutos
29 |  |  |  |  end
   |  |  |  |  Paso b.- Atrasar los servicios anteriores al servicioPosterior (es
   |  |  |  |  decir, atrasar también el servicio)
30 |  |  |  |  servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
31 |  |  |  |  z ← numeroAleatorio(0, 1)
32 |  |  |  |  if y < 0,9 then
33 |  |  |  |  |  minutos ← (fin de servicioPrevio − inicio de servicio) + minutos de
   |  |  |  |  |    desplazamiento entre servicioPrevio y servicio
34 |  |  |  |  else
35 |  |  |  |  |  minutos ← minutos aleatorios
36 |  |  |  |  end
37 |  |  |  |  foreach servicioPrevio a servicioPosterior do
38 |  |  |  |  |  horaInicioServicioPrevio ← horaInicioServicioPrevio + minutos
39 |  |  |  |  |  horaFinServicioPrevio ← horaFinServicioPrevio + minutos
40 |  |  |  |  end
```

```
41
42  |
43  |   |
44  |   |   else if si el servicio no es el primero
    |   |   |   Paso c.- Si se tienen servicios antes que servicio, atrasar los
    |   |   |   servicios anteriores
45  |   |   |   servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
46  |   |   |   z ← numeroAleatorio(0, 1)
47  |   |   |   if y < 0,9 then
48  |   |   |   |   minutos ← (fin de servicioPrevio − inicio de servicio) + minutos de
    |   |   |   |     desplazamiento entre servicioPrevio y servicio
49  |   |   |   else
50  |   |   |   |   minutos ← minutos aleatorios
51  |   |   |   end
52  |   |   |   foreach servicioPrevio a servicio do
53  |   |   |   |   horaInicioServicioPrevio ← horaInicioServicioPrevio + minutos
54  |   |   |   |   horaFinServicioPrevio ← horaFinServicioPrevio + minutos
55  |   |   |   end
    |   |   |   Paso d.- Adelantar los servicios posteriores al servicioAnterior
    |   |   |   (es decir, adelantar también el servicio)
56  |   |   |   servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
57  |   |   |   z ← numeroAleatorio(0, 1)
58  |   |   |   if y < 0,9 then
59  |   |   |   |   minutos ← (fin de servicio − inicio de servicioPosterior) + minutos de
    |   |   |   |     desplazamiento entre servicio y servicioPosterior
60  |   |   |   else
61  |   |   |   |   minutos ← minutos aleatorios
62  |   |   |   end
63  |   |   |   foreach servicioPosterior a servicioPrevio do
64  |   |   |   |   horaInicioServicioPosterior ← horaInicioServicioPosterior + minutos
65  |   |   |   |   horaFinServicioPosterior ← horaFinServicioPosterior + minutos
66  |   |   |   end
67  |   |   end
68  |   else
69  |   |   if si el servicio no es el primero
    |   |   |   Paso e.- Si se tienen servicios antes que servicio, atrasar los
    |   |   |   servicios anteriores
70  |   |   |   servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
71  |   |   |   z ← numeroAleatorio(0, 1)
72  |   |   |   if y < 0,9 then
73  |   |   |   |   minutos ← (fin de servicioPrevio − inicio de servicio) + minutos de
    |   |   |   |     desplazamiento entre servicioPrevio y servicio
74  |   |   |   else
75  |   |   |   |   minutos ← minutos aleatorios
76  |   |   |   end
77  |   |   |   foreach servicioPrevio a servicio do
78  |   |   |   |   horaInicioServicioPrevio ← horaInicioServicioPrevio + minutos
79  |   |   |   |   horaFinServicioPrevio ← horaFinServicioPrevio + minutos
80  |   |   |   end
    |   |   |   Paso f.- Adelantar los servicios posteriores al servicioAnterior
    |   |   |   (es decir, adelantar también el servicio)
81  |   |   |   servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
82  |   |   |   z ← numeroAleatorio(0, 1)
```

```
83
84
85
86
87          if y < 0,9 then
88              minutos ← (fin de servicio − inicio de servicioPosterior) + minutos de
                  desplazamiento entre servicio y servicioPosterior
89          else
90              minutos ← minutos aleatorios
91          end
92          foreach servicioPosterior a servicioPrevio do
93              horaInicioServicioPosterior ← horaInicioServicioPosterior + minutos
94              horaFinServicioPosterior ← horaFinServicioPosterior + minutos
95          end
96      end
97      else if si el servicio no es el último
            Paso g.- Si se tienen servicios después del servicio, adelantar los
            servicios posteriores
98          servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
99          z ← numeroAleatorio(0, 1)
100         if y < 0,9 then
101             minutos ← (fin de servicio − inicio de servicioPrevio) + minutos de
                  desplazamiento entre servicio y servicioPosterior
102         else
103             minutos ← minutos aleatorios
104         end
105         foreach servicioPosterior a servicio do
106             horaInicioServicioPosterior ← horaInicioServicioPosterior + minutos
107             horaFinServicioPosterior ← horaFinServicioPosterior + minutos
108         end
            Paso h.- Atrasar los servicios anteriores al servicioPosterior (es
            decir, atrasar también el servicio)
109         servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
110         z ← numeroAleatorio(0, 1)
111         if y < 0,9 then
112             minutos ← (fin de servicioPrevio − inicio de servicio) + minutos de
                  desplazamiento entre servicioPrevio y servicio
113         else
114             minutos ← minutos aleatorios
115         end
116         foreach servicioPrevio a servicioPosterior do
117             horaInicioServicioPrevio ← horaInicioServicioPrevio + minutos
118             horaFinServicioPrevio ← horaFinServicioPrevio + minutos
119         end
120     end
121 end
    Paso 4.- Si se tienen solapamientos se eligen los minutos para eliminarlos
    (con cierta probabilidad, en el resto de los casos se eligen de forma
    aleatoria)
```

```
123
124   else
125     y ← numeroAleatorio(0, 1)
126     if y < 0,5
127       if servicio no es el último
            Paso a.- Si se tienen servicios después del servicio, adelantar los
            servicios posteriores
128         servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
129         z ← numeroAleatorio(0, 1)
130         if y < 0,9 then
131           minutos ← (fin de servicio − inicio de servicioPrevio) + minutos de
                desplazamiento entre servicio y servicioPosterior
132         else
133           minutos ← minutos aleatorios
134         end
135         foreach servicioPosterior a servicio do
136           horaInicioServicioPosterior ← horaInicioServicioPosterior + minutos
137           horaFinServicioPosterior ← horaFinServicioPosterior + minutos
138         end
            Paso b.- Atrasar los servicios anteriores al servicioPosterior (es
            decir, atrasar también el servicio)
139         servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
140         z ← numeroAleatorio(0, 1)
141         if y < 0,9 then
142           minutos ← (fin de servicioPrevio − inicio de servicio) + minutos de
                desplazamiento entre servicioPrevio y servicio
143         else
144           minutos ← minutos aleatorios
145         end
146         foreach servicioPrevio a servicioPosterior do
147           horaInicioServicioPrevio ← horaInicioServicioPrevio + minutos
148           horaFinServicioPrevio ← horaFinServicioPrevio + minutos
149         end
150       else if si el servicio no es el primero
            Paso c.- Si se tienen servicios antes que servicio, atrasar los
            servicios anteriores
151         servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
152         z ← numeroAleatorio(0, 1)
153         if y < 0,9 then
154           minutos ← (fin de servicioPrevio − inicio de servicio) + minutos de
                desplazamiento entre servicioPrevio y servicio
155         else
156           minutos ← minutos aleatorios
157         end
158         foreach servicioPrevio a servicio do
159           horaInicioServicioPrevio ← horaInicioServicioPrevio + minutos
160           horaFinServicioPrevio ← horaFinServicioPrevio + minutos
161         end
```

```
162
163 |
164 |   |
    |   |   Paso d.- Adelantar los servicios posteriores al servicioAnterior (es
    |   |   decir, adelantar también el servicio)
165 |   |   servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
166 |   |   z ← numeroAleatorio(0, 1)
167 |   |   if y < 0,9 then
168 |   |   |   minutos ← (fin de servicio − inicio de servicioPosterior) + minutos de
    |   |   |    desplazamiento entre servicio y servicioPosterior
169 |   |   else
170 |   |   |   minutos ← minutos aleatorios
171 |   |   end
172 |   |   foreach servicioPosterior a servicioPrevio do
173 |   |   |   horaInicioServicioPosterior ← horaInicioServicioPosterior + minutos
174 |   |   |   horaFinServicioPosterior ← horaFinServicioPosterior + minutos
175 |   |   end
176 |   else
177 |   |   if si el servicio no es el primero
    |   |   |   Paso e.- Si se tienen servicios antes que servicio, atrasar los
    |   |   |   servicios anteriores
178 |   |   |   servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
179 |   |   |   z ← numeroAleatorio(0, 1)
180 |   |   |   if y < 0,9 then
181 |   |   |   |   minutos ← (fin de servicioPrevio − inicio de servicio) + minutos de
    |   |   |   |    desplazamiento entre servicioPrevio y servicio
182 |   |   |   else
183 |   |   |   |   minutos ← minutos aleatorios
184 |   |   |   end
185 |   |   |   foreach servicioPrevio a servicio do
186 |   |   |   |   horaInicioServicioPrevio ← horaInicioServicioPrevio + minutos
187 |   |   |   |   horaFinServicioPrevio ← horaFinServicioPrevio + minutos
188 |   |   |   end
    |   |   |   Paso f.- Adelantar los servicios posteriores al servicioAnterior
    |   |   |   (es decir, adelantar también el servicio)
189 |   |   |   servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
190 |   |   |   z ← numeroAleatorio(0, 1)
191 |   |   |   if y < 0,9 then
192 |   |   |   |   minutos ← (fin de servicio − inicio de servicioPosterior) + minutos de
    |   |   |   |    desplazamiento entre servicio y servicioPosterior
193 |   |   |   else
194 |   |   |   |   minutos ← minutos aleatorios
195 |   |   |   end
196 |   |   |   foreach servicioPosterior a servicioPrevio do
197 |   |   |   |   horaInicioServicioPosterior ← horaInicioServicioPosterior + minutos
198 |   |   |   |   horaFinServicioPosterior ← horaFinServicioPosterior + minutos
199 |   |   |   end
```

```
200
201 |
202 |   |
203 |   |   else if si el servicio no es el último
    |   |   |   Paso g.- Si se tienen servicios después del servicio, adelantar los
    |   |   |   servicios posteriores
204 |   |   |   servicioPosterior ← servicioSiguiente(servicio, serviciosNuevos)
205 |   |   |   z ← numeroAleatorio(0, 1)
206 |   |   |   if y < 0,9 then
207 |   |   |   |   minutos ← (fin de servicio − inicio de servicioPrevio) + minutos de
    |   |   |   |     desplazamiento entre servicio y servicioPosterior
208 |   |   |   else
209 |   |   |   |   minutos ← minutos aleatorios
210 |   |   |   end
211 |   |   |   foreach servicioPosterior a servicio do
212 |   |   |   |   horaInicioServicioPosterior ← horaInicioServicioPosterior + minutos
213 |   |   |   |   horaFinServicioPosterior ← horaFinServicioPosterior + minutos
214 |   |   |   end
    |   |   |   Paso h.- Atrasar los servicios anteriores al servicioPosterior (es
    |   |   |   decir, atrasar también el servicio)
215 |   |   |   servicioPrevio ← servicioAnterior(servicio, serviciosNuevos)
216 |   |   |   z ← numeroAleatorio(0, 1)
217 |   |   |   if y < 0,9 then
218 |   |   |   |   minutos ← (fin de servicioPrevio − inicio de servicio) + minutos de
    |   |   |   |     desplazamiento entre servicioPrevio y servicio
219 |   |   |   else
220 |   |   |   |   minutos ← minutos aleatorios
221 |   |   |   end
222 |   |   |   foreach servicioPrevio a servicioPosterior do
223 |   |   |   |   horaInicioServicioPrevio ← horaInicioServicioPrevio + minutos
224 |   |   |   |   horaFinServicioPrevio ← horaFinServicioPrevio + minutos
225 |   |   |   end
226 |   |   end
227 |   end
228 end
229 vecinoFinal ← solucionCandidata con los servicios modificados
230 else
231 |   vecinoFinal ← solucionCandidata
232 end
233 return vecinoFinal
```

A.4.8. Intercambiar varios servicios entre auxiliares

El movimiento *intercambiar varios servicios de auxiliar* consiste en intercambiar las auxiliares que realizan dos grupos de servicios. Estos grupos constan de todos los servicios de un usuario que se deben realizar en determinada franja durante un grupo de días (diferenciando días de semana de días de fin de semana, de modo que grupos de servicios de días de semana sólo se pueden intercambiar con otro grupo de servicios de días de semana y, del mismo modo, solamente se pueden intercambiar dos grupos de servicios de fin de semana).

Nombre y clase	Descripción
Variables de entrada	
solucionCandidata, `List<ServicioPlan>`	Lista de servicios
serviciosModificables, `List<ServicioPlan>`	Lista de servicios que se pueden modificar
Variables que se utilizan en la función	
Servicio1, `ServicioPlan`	Servicio al que se le intercambiará la auxiliar
Servicio2, `ServicioPlan`	Servicio al que se le intercambiará la auxiliar
Servicios1, `List<ServicioPlan>`	Lista de servicios del mismo tipo que *servicio1*
Servicios2, `List<ServicioPlan>`	Lista de servicios del mismo tipo que *servicio2*
auxiliarServicio1Original, `Auxiliar`	Auxiliar original que atiende al *servicio1*
auxiliarServicio2Original, `Auxiliar`	Auxiliar original que atiende al *servicio2*

Algoritmo A.12: Intercambiar servicios entre auxiliares

```
1  Inicio
   Paso 1.- Seleccionar los servicios que se pueden modificar
2  serviciosNuevos ← serviciosModificables
3  if serviciosNuevos ≠ ∅ then
       Paso 2.- Seleccionar el primer servicio a intercambiar
4      servicio1 ← servicioAleatorio(serviciosNuevos)
       Paso 3.- Seleccionar las posibles auxiliares
5      auxMismoNivelAfinidad ← auxiliaresMismoNivelAfinidad(auxiliares, servicio1)
6      auxDesplazamientoGrande ←
         auxiliaresDesplazamientoGrande(auxMismoNivelAfinidad, servicio1)
7      auxMismoNivelAfinidad ← auxMismoNivelAfinidad \ auxDesplazamientoGrande
8      auxiliar ← auxiliarAleatorio(auxMismoNivelAfinidad)
       Paso 4.- Seleccionar los posibles servicios a intercambiar
9      serviciosAuxDia ←
         serviciosAuxiliarDiaAltaAfinidadyBajoDesplazamiento(auxiliar, dia(servicio1))
       Paso 5.- Seleccionar el otro servicio a intercambiar
10     servicio2 ← servicioAleatorio(serviciosAuxDia)
       Paso 6.- Para cada servicio determinar los conjuntos de servicios que son
       del mismo tipo que él (servicios de la misma persona, misma auxiliar y
       misma franja que el servicio dado)
11     servicios1 ← serviciosMismaFranjaUsuario(servicio1,solucionCandidata)
12     servicios2 ← serviciosMismaFranjaUsuario(servicio2,solucionCandidata)
       Paso 7.- Intercambiar las auxiliares de los servicios
13     foreach servicio1 de servicios1 do
14         auxiliarServicio1 ← auxiliarServicio2Original
15     end
16     foreach servicio2 de servicios2 do
17         auxiliarServicio2 ← auxiliarServicio1Original
18     end
19     vecinoFinal ← solucionCandidata con los servicios modificados
20 else
21     vecinoFinal ← solucionCandidata
22 end
23 return vecinoFinal
```

A.4.9. Intercambiar y desplazar

El movimiento *intercambiar y desplazar* consiste en intercambiar el horario de dos servicios y seguidamente aplicar el movimiento *desplazar servicios*.

Nombre y clase	Descripción
Variables de entrada	
solucionCandidata, `List<ServicioPlan>`	Lista de servicios
serviciosModificables, `List<ServicioPlan>`	Lista de servicios que se pueden modificar
Variables que se utilizan en la función	
serviciosNuevos, `List<ServicioPlan>`	Lista de servicios que se usan para aplicar el movimiento
vecinoIntermedio, `List<ServicioPlan>`	Lista de servicios tras realizar el intercambio de auxiliares
Servicio1, `ServicioPlan`	Servicio al que se le intercambiará el horario
Servicio2, `ServicioPlan`	Servicio al que se le intercambiará el horario
horaInicioServicio1, `Calendar`	Hora de inicio del *servicio1*
horaFinServicio1, `Calendar`	Hora de fin del *servicio1*
duracionServicio1, `int`	Duración en minutos del *servicio1*
horaInicioServicio2, `Calendar`	Hora de inicio del *servicio2*
horaFinServicio2, `Calendar`	Hora de fin del *servicio2*
duracionServicio2, `int`	Duración en minutos del *servicio2*
vecinoFinal, `List<ServicioPlan>`	Solución modificada

Algoritmo A.13: Intercambiar y desplazar

```
1  Inicio
   Paso 1.- Seleccionar los servicios que se pueden modificar
2  x ← numeroAleatorio(0, 1)
3  if x < 0,9 then
4  |  serviciosNuevos ← serviciosSolapamientoDescanso(solucionCandidata)
5  else
6  |  serviciosNuevos ← serviciosModificables
7  end
8  vecinoFinal ← ∅
9  if serviciosNuevos ≠ ∅
   |  Paso 2.- Seleccionar el primer a intercambiar
10 |  servicio1 ← servicioAleatorio(serviciosNuevos)
   |  Paso 3.- Seleccionar los servicios que realiza la auxiliar en el día en que
   |  lleva a cabo servicio1
11 |  serviciosAuxDia ← serviciosAuxiliarDia(auxiliar(servicio1),
   |    dia(servicio1),serviciosNuevos)
   |  Paso 4.- Obtener los bloques en que se divide serviciosAuxiliarDia (los
   |  grupos de servicios que están divididos por un descanso grande)
12 |  bloque1 ← prierBloque(serviciosAuxDia)
13 |  bloque2 ← segundoBloque(serviciosAuxDia)
```

```
14
    | Paso 5.- Seleccionar el bloque en que se encuentra servicio1
15  | if servicio1 ∈ bloque1 then
16  |   | bloque ← bloque1
17  | else
18  |   | bloque ← bloque2
19  | end
    | Paso 6.- Seleccionar un servicio del bloque
20  | servicio2 ← servicioAleatorio(bloque)
    | Paso 7.- Intercambiar las horas de inicio de los servicios y adaptar las
    | horas de fin según sus duraciones
21  | horaInicioServicio1 ← horaInicioServicio2Original
22  | horaFinServicio1  ← horaInicioServicio2Original + duracionServicio1
23  | horaInicioServicio2 ← horaInicioServicio1Original
24  | horaFinServicio2 ← horaInicioServicio1Original + duracionServicio2
25  | vecinoIntermedio ← solucionCandidata con los servicios modificados
    | Paso 8.- Seleccionar un servicio para hacer de pivote
26  | servicio2 ← servicioAleatorio(vecinoIntermedio)
    | Paso 9.- Desplazar los servicios (análogo que el movimiento desplazar
    | servicios)
27  | vecinoFinal ← despazarVariosServicios(servicio, vecinoIntermedio)
28 else
29  | vecinoFinal ← solucionCandidata
30 end
31 return vecinoFinal
```

Apéndice B

Código del problema de programación lineal

En este anexo se presenta el código en java empleado para resolver problemas exactos utilizando la librería Gurobi.

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.opencsv.CSVReader;
import com.opencsv.CSVWriter;

import gurobi.*;

public class problema_latex_gurobi {

// Funcion para calcular el factorial de un numero (necesario para
   obtener el numero de posibles descansos que puede haber en un dia)
public static long factorial(int n) {
long resultado = 1;
for (int i = 1; i <= n; i++) {
resultado *= i;
}
return resultado;
}

public static void main(String[] args) {

try {
// Ubicacion de los datos y nobre del problema
String archivo = "C:/Users/ejemplos_reales_mayores/";
String ejemplo = "ejemplo00b";
```

```
// Parametros del porblema:
int s = 0; // Numero de tareas
int m = 0; // Numero de períodos
int n = 0; // Numero de auxiliares
int ndias = 0; // Numero de dias
int Nvec, Ncol, Nrow, ret = 0; // Tamaño de vectores y matrices de
    datos
double peso_tiemp = 1; // Peso que se le da a la duracion de la
    jornada laboral semanal que realiza una auxiliar en la funcion
    objetivo
double peso_afin = 1; // Peso que se le de a la aafinidad entre una
    auxiliar y un servicio en la funcion objetivo
double peso_vopt = 1; // Peso que se le da a que un servicio se
    realice dentro de su ventana optima en la funcion objetivo
double peso_despl = 1; // Peso que se le da al desplazamiento entre
    servicios
int C = 8000; // Constante para la restriccion 7 pt. 3
int D = 4000; // Constante para la restriccion 7 pt. 2 y 3
int E = 4000; // Constante para la restricicion de la cota del mayor
    descanso
int mayorDescanso = 24; // Limite inferior para el descanso que no se
    considera


// Numero de dias y de períodos
try{
CSVReader reader = null;
reader = new CSVReader(new FileReader(archivo + ejemplo + "/" +
    "dias.csv"), ';'); // Leer el archivo
String[] row; // Filas del fichero
row = reader.readNext(); // Primera fila con nombres de columnas
while ((row = reader.readNext()) != null) {
// numero de dias
ndias = Integer.parseInt(row[0]); // convertir el string en un int
System.out.println("ndias. " + ndias);
// numero de períodos (coincide con el ultimo período del ultimo dia)
m = ndias*216;
System.out.println("m: " + m);
}
reader.close();
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}


// Numero de auxiliares
try{
CSVReader reader = null;
```

```
reader = new CSVReader(new FileReader(archivo + ejemplo + "/" +
    "auxiliares.csv"), ';');
String[] row; // Filas del fichero
row = reader.readNext(); // Primera fila con nombres de columnas
int contador = 0;
while ((row = reader.readNext()) != null) {
contador++;
}
// numero de auxiliares (contador)
n = contador;
System.out.println("n: " + n);
reader.close();
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}


// Numero de servicios
try{
CSVReader reader = null;
reader = new CSVReader(new FileReader(archivo + ejemplo + "/" +
    "servicios.csv"), ';');
String[] row; // Filas del fichero
row = reader.readNext(); // Primera fila con nombres de columnas
int contador = 0;
while ((row = reader.readNext()) != null) {
contador++;
}
// numero de auxiliares (contador)
s = contador;
System.out.println("s: " + s);
reader.close();
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}


// Vectores y matrices que almacenan los datos de entrada del problema
int[] mdias = new int [ndias+1]; // Vector que almacena el ultimo
    período de cada dia (de modo que el ultimo período del ultimo dia
    es el ultimo período de la semana)
int[][] afin = new int [n][s+2]; // Matriz que amacena las afinidades
    entre auxiliares y servicios (incluyendo los servicios ficticios
    que tienen afinidad 0)
int[][] H = new int[ndias+1][n]; // Matriz que almacena para cada
    auxiliar el numero de horas maximo que pueden trabajar cada dia,
    y, en ultimo lugar, cada semana (cada columna es una auxiliar y en
    las filas se almacena la jornada maxima para cada dia, y en la
```

```
   ultima fila la jornada maxima semanal)
int[][] A =  new int[5][s];     // Matriz que almacena, para cada
   servicio, la duracion, ventana de tiempo disponible y ventana de
   tiempo optimo (cada columna es un servicio, en la primera fila se
   tiene la duracion, en las segunda y tercera la ventana disponible
   y en las cuatra y quinta la ventana optima)
int[][] T = new int[s+2][s+2]; // Matriz que almacena los tiempos
   empleados en desplazarse entre dos servicios (tambien se
   consideran los servicios ficticios, fijando los desplazamientos
   desde y hasta ellos iguales a 0)
T[0][0] = m; // El tiempo de desplazamiento entre la ficticia inicial
   y ella misma es m (para que no tener problemas en la restriccion 5)
T[s+2-1][s+2-1] = m; // El tiempo de desplazamiento entre la ficticia
   final y ella misma es m (para no tener porblemas en la restriccion
   5)

// Leer datos de los dias
try{
CSVReader reader = null; // Crear el reader
reader = new CSVReader(new FileReader(archivo + ejemplo + "/" +
   "dias.csv"), ';'); // Leer el fichero
String[] row; // String que leera las filas
row = reader.readNext(); // Primera fila con nombres de columnas
while ((row = reader.readNext()) != null) { // Mientras haya filas no
   vacias se continua leyendo el fichero
// Numero de dias
ndias = Integer.parseInt(row[0]); // El numero de dias es el primer
   elemento de la fila
// Vector que almacena el ultimo período de cada dia
mdias[0] = 0; // El primer dia empieza en el período 1
System.out.println("períodos dias 0: " + mdias[0]);
for (int j = 1; j <= ndias; j++) { // Para cada elemento de la fila
   (a excepcion del primero)
mdias[j] = j*216; // Ultimo período de cada dia  mdias[j-1] =
   Integer.parseInt(row[j]);
System.out.println("períodos dias "+ (j) + ":" + mdias[j]);
}
}
reader.close(); // Cerrar el reader
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}


// Leer datos de las auxiliares
try{
CSVReader reader = null;
reader = new CSVReader(new FileReader(archivo + ejemplo + "/" +
   "auxiliares.csv"), ';');
String[] row; // Filas del fichero
```

```
row = reader.readNext(); // Primera fila con nombres de columnas
int contador = 0; // Contador para determinar la fila en la que nos
    encontramos (cada fila contiene los datos de una auxilair)
while ((row = reader.readNext()) != null) {
// Matriz que almacena las horas maximas que puede trabajar una
    auxiliar al dia y a la semana
for (int j = 1; j < row.length; j++) { // Recorrer todas las columnas
    (horas diarias, horas semanales)
H[j-1][contador] = (Integer.parseInt(row[j])*60)/5; // Cada columna
    es una auxiliar
System.out.println("maximo aux: " + H[j-1][contador] );
}
contador++;
}
reader.close();  // Cerrar el reader
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}


// Leer los datos de los servicios
try{
CSVReader reader = null;
reader = new CSVReader(new FileReader(archivo + ejemplo + "/" +
    "servicios.csv"), ';');
String[] row; // Filas del fichero
row = reader.readNext(); // Primera fila con nombres de columnas
int contador = 0; // Contador para determinar la fila en la que nos
    encontramos (cada fila contiene los datos de un servicio)
while ((row = reader.readNext()) != null) {
// Matriz que almacena la duración y las ventanas de tiempo
    disponible de cada servicio
for (int j = 1; j < (row.length-1); j++) { // Recorrer todas las
    columnas (duracion, ventana factible, ventana optima) meno la
    ultima
// Pasar las horas a instantes de tiempo
if(j==1){ // Si estamos tratando con la duracion del servicio
int duracion = Integer.parseInt(row[j])/5; // La duracion original se
    expresa en minutos
A[j-1][contador] = duracion;
System.out.println("duracion servicio " + A[j-1][contador]);

} else { // Si estamos tratando con las ventanas
int dia = Integer.parseInt(row[row.length-1]); // Dia en que se debe
int ventana =
    (int)(((Double.parseDouble(row[j])-6)*60)/5)+((dia-1)*216)+1;
A[j-1][contador] = ventana;
System.out.println("datos ventana " + A[j-1][contador] + " dia: " +
    dia);
}
```

```
// Cada columna es un servicio
}
contador++;
}
reader.close();  // Cerrar el reader
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}


// Leer la matriz de tiempos de desplazamiento entre servicios
try{
CSVReader reader = null;
reader = new CSVReader(new FileReader(archivo + ejemplo + "/" +
    "tiempos.csv"), ';');
String[] row; // Filas del fichero
row = reader.readNext();// Primera fila con nombres de columnas
int contador = 0; // Contador para determinar en que fila (servicio
    inicio) nos encontramos
while ((row = reader.readNext()) != null) {
// Matriz que almacena el tiempo de desplazamiento entre dos servicios
for (int j = 1; j < row.length+1; j++) { // Recorrer todos las
    columnas (servicio fin)
if(contador == j-1){ // El elemento se encuentra en la diagonal de la
    matriz (es decir, el servicio inicio y el servicio fin son el
    mismo)
T[contador+1][j-1+1] = m; // El tiempo de desplazamiento entre un
    servicio y el mismo se define como m
} else { // El elemento no se encuentra en la diagonal (es decir, el
    servicio inicio y el servicio fin son distintos)
T[contador+1][j-1+1] = (Integer.parseInt(row[j-1])/5); // Tiempo de
    desplazamiento entre servicios distintos
}}
contador++;
}
reader.close();  // Cerrar el reader
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}


// Leer las afinidades entre auxiliares y servicios
try{
CSVReader reader = null;
reader = new CSVReader(new FileReader(archivo + ejemplo + "/" +
    "afinidad.csv"), ';');
String[] row; // Filas del fichero
row = reader.readNext(); // Primera fila con nombres de columnas
```

```
int contador = 0; // Contador para determinar la fila en la que
    estamos (cada fila es una auxilair)
while ((row = reader.readNext()) != null) {
// Matriz que almacena las afinidades
for (int j = 1; j < row.length; j++) { // Recorremos todas las
    columnas (cada columna es un servicio)
afin[contador][j] = Integer.parseInt(row[j]); // Afinidad entre una
    auxiliar y un servicio
}
contador++;
}
reader.close();  // Cerrar el reader
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}


// Servicios al dia

// Variables que almacenarán el numero de servicios de cada día
int Lunes = 0;
int Martes = 0;
int Miercoles = 0;
int Jueves = 0;
int Viernes = 0;
int Sabado = 0;
int Domingo = 0;

// Contabilixar cuantos servicios se realizan en cada dia
for(int q = 0; q < ndias; q++){ // Dias
for (int k = 1; k < s+1; k++) { // Servicios iniciales
if(A[2-1][k-1] <= mdias[q+1] && A[2-1][k-1] > mdias[q]){ // Si la
    ventana factible esta dentro de los períodos del dia q
// Mirar que dia se tiene que realizar el servicio y sumar 1 a la
    variable que contabiliza el numero de servicios que se realizan
    ese dia
if(q+1==1){ // El servicio tiene su ventana factible en el lunes
Lunes++;
} else if (q+1==2){ // El servicio tiene su ventana factible en el
    martes
Martes++;
} else if (q+1==3){ // El servicio tiene su ventana factible en el
    miercoles
Miercoles++;
} else if (q+1==4){ // El servicio tiene su ventana factible en el
    jueves
Jueves++;
} else if (q+1==5){ // El servicio tiene su ventana factible en el
    viernes
Viernes++;
```

```
} else if (q+1==6){ // El servicio tiene su ventana factible en el
    sabado
Sabado++;
} else if (q+1==7){ // El servicio tiene su ventana factible en el
    domingo
Domingo++;
}}}}

// --------------- Varibles y --------------

// Numero de variables y que se tienen para cada dia de la semana (es
    decir, numero de formas de obtener un posible descanso)
long yLunes = 0; // Inicializar el numero de variables a 0 para el
    Lunes
if(Lunes == 1 || Lunes == 0){ // Si el Lunes solo hay un servicio
    posible (o ninguno)
yLunes = 0; // No hay que añadir variables y para este dia
} else if(Lunes == 2){ // Si el Lunes solo hay dos servicios
yLunes = 2; // Solo se tienen dos formas de obtener un posible
    descanso
} else { // Si el Lunes se tienen mas de 2 servicios el numero de
    formas de obtener un posible descanso se calcula segun la formula:
    Lunes!/(Lunes-1)! + Lunes!/(Lunes-2)!
yLunes = (factorial(Lunes)/factorial((Lunes-3)))+
    (factorial(Lunes)/factorial((Lunes-2))); // Aplicacion de la
    formula
}
long yMartes = 0; // Inicializar el numero de variables a 0 para el
    Martes
if(Martes == 1 || Martes == 0){ // Si el Martes solo hay un servicio
    posible (o ninguno)
yMartes = 0; // No hay que añadir variables para este dia
} else if(Martes == 2){ // Si el Martes solo hay dos servicios o
yMartes = 2; // Solo se tienen dos formas de obtener un posible
    descanso
} else { // Si el Martes se tienen mas de dos servicios el numero de
    formas de obtener un posible descanso se calcula segun la formula:
    Martes!/(Martes-1)! + Martes!/(Martes-2)!
yMartes = (factorial(Martes)/factorial(Martes-3))+
    (factorial(Martes)/factorial((Martes-2))); // Aplicar la formula
}
long yMiercoles = 0; // Inicializar el numero de variables a 0 para
    el Miercoles
if(Miercoles == 1 || Miercoles == 0){ // Si el Miercoles solo hay un
    servicio posible (o ninguno)
yMiercoles = 0; // No hay que añadir variables para este dia
} else if(Miercoles == 2){ // Si el Miercoles solo hay dos servicios
yMiercoles = 2; // Solo hay dos formas de obtener un posible descanso
} else { // Si el Miercoles se tienen mas de dos servicios el numero
    de formas de obtener un posible descanso se calcula segun la
    formula: Miercoles!/(Miercoles-1)! + Miercoles!/(Miercoles-2)!
```

```
yMiercoles = (factorial(Miercoles)/factorial(Miercoles-3))+
    (factorial(Miercoles)/factorial((Miercoles-2))); // Aplicacion de
    la formula
}
long yJueves = 0; // Inicializar el numero de variables a 0 para el
    Jueves
if(Jueves == 1 || Jueves == 0){ // Si el Jueves solo hay un servicio
    posible (o ninguno)
yJueves = 0; // No hay que añadir variables y para este dia
} else if(Jueves == 2){ // Si el Jueves solo hay dos servicios
yJueves = 2; // Solo se tienen dos formas de obtener un posible
    descanso
} else { // Si el Jueves se tienen mas de dos servicios el numero de
    formas de obtener un posible descanso se calcula segun la formula:
    Jueves!/(Jueves-1)! + Jueves!/(Jueves-2)!
yJueves = (factorial(Jueves)/factorial(Jueves-3))+
    (factorial(Jueves)/factorial((Jueves-2))); // Aplicacion de la
    formula
}
long yViernes = 0; // Inicializar el numero de variables a 0 para el
    Viernes
if(Viernes == 1 || Viernes == 0){ // Si el Viernes solo hay un
    servicio posible (o ninguno)
yViernes = 0; // No hay que añadir variables y para este dia
} else if(Viernes == 2){ // Si el Viernes solo hay dos servicios
yViernes = 2; // Solo se tienen dos formas de obtener un posible
    descanso
} else { // Si el Viernes se tienen mas de dos servicios el numero de
    formas de obtener un posible descanso se calcula segun la formula:
    Viernes!/(Viernes-1)! + Viernes!/(Viernes-2)!
yViernes = (factorial(Viernes)/factorial(Viernes-3))+
    (factorial(Viernes)/factorial((Viernes-2))); // Aplicacion de la
    formula
}
long ySabado = 0; // Inicializar el numero de variablesa a 0 para el
    Sabado
if(Sabado == 1 || Sabado == 0){ // Si el Sabado solo hay un servicio
    posible (o ninguno)
ySabado = 0; // No hay que añadir variables y para este dia
} else if(Sabado == 2){ // Si el Sabado solo hay dos servicios
ySabado = 2; // Solo se tienen dos formas de obtener un posible
    descanso
} else { // Si el Sabado se tienen mas de dos servicios el numero de
    formas de obtener un posible descanso se calcula segun la formula:
    Sabado!/(Sabado-1)! + Sabado!/(Sabado-2)!
ySabado = (factorial(Sabado)/factorial(Sabado-3))+
    (factorial(Sabado)/factorial((Sabado-2))); // Aplicacion de la
    formula
}
long yDomingo = 0; // Inicializar el numero de variables a 0 para el
    Domingo
```

```
if(Domingo == 1 || Domingo == 0){ // Si el Domingo solo hay un
    servicio posible (o ninguno)
yDomingo = 0; // No hay que añadir variables para este dia
} else if(Domingo == 2){ // Si el Domingo solo hay dos servicios
yDomingo = 2; // Solo hay dos formas de obtener un posible descanso
} else { // Si el Domingo se tienen mas de dos servicios el numero de
    formas de obtener un posible descanso se calcula segun la formula:
    Domingo!/(Domingo-1)! + Domingo!/(Domingo-2)!
yDomingo = (factorial(Domingo)/factorial(Domingo-3))+
(factorial(Domingo)/factorial((Domingo-2))); // Aplicacion de la
    formula
}

// Numero total de formas para obtener posibles descansos (es decir,
    numero total de variables y)
long yTotal = yLunes + yMartes + yMiercoles + yJueves + yViernes +
    ySabado + yDomingo;

// Vector que almacena el numero de variables y (es decir, numero de
    formas de obtener un posible descanso) para cada dia de la semana
long[] dias = {yLunes, yMartes, yMiercoles, yJueves, yViernes,
    ySabado, yDomingo};

// Vector que almacena, para cada dia de la semana, el numero de
    variables y acumuladas en los dias anteriores
long[] yDias = {0, yLunes, yLunes + yMartes, yLunes + yMartes+
    yMiercoles, yLunes + yMartes + yMiercoles+ yJueves, yLunes +
    yMartes + yMiercoles + yJueves + yViernes, yLunes + yMartes +
    yMiercoles + yJueves + yViernes + ySabado, yLunes + yMartes +
    yMiercoles + yJueves + yViernes + ySabado + yDomingo};


// Comprobar a que dia pertenece cada servicio
int[] diaServicio = new int[s+1]; // Vector que almacena, para cada
    servicio, el dia de la semana en que se realiza
for(int q = 0; q < ndias; q++){ // Dias (se empieza a contar en 0, de
    modo que despues sera necesario sumar 1 a q para tener el numero
    del dia)
for (int k = 1; k < s+2; k++) { // Servicios
if(k==s+1){ // Si el servicio a tratar es el ficticio final
diaServicio[k-1] = 0; // Se le asigna el dia 0 puesto que al ser
    ficticio no tiene un dia especifico asignado
} else if(A[2-1][k-1] <= mdias[q+1] && A[2-1][k-1] > mdias[q]){ // Si
    el servicio tiene su ventana disponible entre los períodos del dia
    q
diaServicio[k-1] = q+1; // Asignamos al servicio el dia de la semana
    que le corresponde (q+1)
}}}
```

```
// Crear variables que intervienen en el problema (en lpSolve las
   variables se identifican con numeros que denotan su posicion en el
   problema, empeando a contar en 1, la primera variable, y
   terminando en N, la ultima varible)
Nvec = (s+2)*(s+2)*m*n; // Numero de variables x_{ijkl}
Ncol = (s+2)*m; // Numero de columnas de la matriz de x_{ijkl} para
   definir su estructura, se tienen tantos grupos de columnas como
   períodos tiene la semana y en cada grupo se tienen s+2 (numero de
   servicios + ficticios) columnas
Nrow = (s+2)*n; // Numero de filas de la matriz de x_{ijkl} para
   definir su estructura, se tienen tantos grupos de filas como
   auxiliares y en cada grupo hay s+2 (numero de servicios +
   ficticios) filas

// ---------------------- DEFINIR EL MODELO ------------------------

// Crear el entorno y el modelo
GRBEnv    env   = new GRBEnv(archivo + ejemplo + "/" +
   "ejemplo.log"); // Crear enviroment (en el .log se guardan los
   resultados)
GRBModel  model = new GRBModel(env); // Crear modelo


// -------------------- DEFINIR LAS VARIABLES ---------------------

// Definir las variables x_{ijkl}
GRBVar[][] x = new GRBVar[Nrow][Ncol];
int a = 0, b = 0, cont = 1;
for (int i = 1; i < n+1; i++){ // Auxiliare
for(int k = 0; k < s+2; k++){ // Tarea que se realiza k
for (int j = 1; j < m+1; j++){ // Periodo en que se realiza j
for (int l = 0; l < s+2; l++){ // Servicio siguiente l
x[a][b] =  model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "xi" + i + "j" + j
   + "k" + k + "l" + l);
b++; // Aumentar la columna
cont++;
}}
a++; // Aumentar la fila
b = 0; // Reiniciar la columna
}}
System.out.println("x: " + cont);

// Definir las variables mu_{iq}
GRBVar[][] mu = new GRBVar[n][ndias];
for (int i = 1; i < (n+1); i++){ // Auxiliares
for (int j = 1; j < (ndias+1); j++){ // Dias
mu[i-1][j-1] = model.addVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER, "mu"
   + i + "" + j);
cont++;
}}

// Definirt las variables y_{iqklp}
```

```
GRBVar[][] y = new GRBVar[n][(int)yTotal];
for (int i = 1; i < (n+1); i++){ // Auxiliares
for (int j = 1; j < (yTotal+1); j++){ // Posibles descansos
y[i-1][j-1] = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "y" + i + "" +
    j);
cont++;
}}

// Definir las variables y_{iq}
GRBVar[][] y_comodin = new GRBVar[n][(int)ndias];
for (int i = 1; i < (n+1); i++){ // Auxiliares
for (int j = 1; j < (ndias+1); j++){ // Dias
y_comodin[i-1][j-1] = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "y_" +
    i + "" + j);
cont++;
}}

// Definir las variables z_{iq}
GRBVar[][] z = new GRBVar[n][ndias];
for (int i = 1; i < (n+1); i++){ // Auxiliares
for (int j = 1; j < (ndias+1); j++){ // Dias
z[i-1][j-1] = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "z" + i + "" +
    j);
cont++;
}}

// Definir las variables d_{iq}
GRBVar[][] d = new GRBVar[n][ndias];
for (int i = 1; i < (n+1); i++) { // Auxiliares
for (int j = 1; j < (ndias+1); j++){ // Dias
d[i-1][j-1] = model.addVar(0.0, GRB.INFINITY, 0.0, GRB.INTEGER, "d" +
    i + "" + j);
cont++;
}}


// ------ Print -------
System.out.println("Calculando...");
// --------------------

// Fijar variables imposibles a 0
System.out.println("     Fijar variables imposibles a 0");
if(ret == 0) {
// Igualar a cero las variables con servicio final l = 0
for (int i = 0; i < n*(s+2); i++) { // Auxiliares
int ndim2 = m; // Tamaño del vector
for(int j=0; j < ndim2; j++){ // Indice del vector
x[i][j*(s+2)].set(GRB.DoubleAttr.UB, 0); // Fijar el valor de la
    variable a 0 (cota inferior, superior)
}}

// Igualar a cero las variables con k = 4
```

```
for (int i = 1; i < n+1; i++) { // Auxiliares
int ndim2 = (s+2)*m; // Tamaño del vector
for(int j=0; j < ndim2; j++){ // Indice del vector
x[(s+2-1)+((i-1)*(s+2))][j].set(GRB.DoubleAttr.UB, 0); // Fijar el
   valor de la variable a 0 (cota inferior, superior)
}}
// Igualar a cero las variables con k=l
int aa = 0, bb = 0;
for(int i = 1; i<n+1; i++){ // Auxiliares
for (int k = 0; k < s+2; k++){ // Servicios iniciales
for(int j = 1; j<m+1; j++){ // Instantes de tiempo
for(int l = 0; l<s+2; l++){ // Servicios finales
if(k == l){
x[aa][bb].set(GRB.DoubleAttr.UB, 0); // Fijar el valor de la variable
   a 0 (cota inferior, superior)
}
bb++; // Aumentar la columna
}}
aa++; // Aumentar la fila
bb=0; // Reiniciar la columna
}}}



// ----------------- Añadir las restricciones al problema
   --------------------

// Primera restriccion (cada servicio se realiza una sola vez y
   ademas se realiza dentro de su vantana de tiempo dsiponible)
System.out.println("     Restriccion 1: Cada servicio solo se realiza
   una vez y se hace dentro de su ventana factible");
if(ret == 0) {
int aa = 0, bb = 0;
for (int k = 1; k < s+1; k++){  // Servicios iniciales
GRBLinExpr expr = new GRBLinExpr();
for(int i = 1; i<n+1; i++){ // Auxiliares
for(int j = 1; j<m+1; j++){ // Instantes de tiempo
for(int l = 0; l<s+2; l++){ // Servicios finales
if(A[1][k-1] <= j && j <= (A[2][k-1] - A[0][k-1])){
expr.addTerm(1, x[(aa)*(s+2)+k][bb]);
} else {
x[(aa)*(s+2)+k][bb].set(GRB.DoubleAttr.UB, 0);
}
bb++; // Aumentar la columna
}}
aa++; // Aumentar la fila
bb=0; // Reiniciar la columna
}
aa = 0; // Reiniciar la fila
bb = 0; // Reiniciar la columna

// Añadir la restricción al problema
```

```
model.addConstr(expr, GRB.EQUAL, 1.0, "Restr1"); // Añadir la
    restriccion al problema, restricción = 1
}}


// Segunda restriccion (cada servicio solo puede tener un servicio
    anterior)
System.out.println("     Restriccion 2: Cada servicio solo puede
    tener un servicio anterior");
if(ret == 0) {
int aa = 0, bb = 0;
for (int l = 1; l < s+1; l++){ // Servicios (finales)
GRBLinExpr expr = new GRBLinExpr();
for(int j = 1; j<m+1; j++){ // Instantes de tiempo
for(int i = 1; i<n+1; i++){ // Auxiliares
for (int k = 0; k < s+2; k++){  // Servicios (iniciales)
expr.addTerm(1, x[aa][bb*(s+2)+l]);
aa++; // Aumentar la fila
}}
aa = 0; // Reiniciar la fila
bb++; // Aumentar la columan
}
bb=0; // Reiniciar la columna

// Añadir la restricción al problema
model.addConstr(expr, GRB.EQUAL, 1.0, "Restr2"); // Añadir la
    restriccion al problema, restricción = 1
}}


// Tercera restriccion (empezar en el servicio 0, para cada dia)
System.out.println("     Restriccion 3: Comenzar en el servicio
    ficticio inicial (cada dia)");
if(ret == 0) {
int aa = 0, bb = 0;
for(int i = 1; i<n+1; i++){ // Auxiliares
for(int q = 1; q < ndias+1; q++){ // Dias de la semana
GRBLinExpr expr = new GRBLinExpr();
for(int j = 1; j<m+1; j++){ // Instantes de tiempo
for (int l = 0; l < s+2; l++){ // Servicios (finales)
if(mdias[q-1] < j && j <= mdias[q]){
if(l==0 || l==(s+2-1)){ // Si el servicio l es ficticio
expr.addTerm(1, x[aa*(s+2)][bb]);
} else if (diaServicio[l-1]==q){ // Si el servicio l se realiza el
    dia q
expr.addTerm(1, x[aa*(s+2)][bb]);
}}
bb++; // Aumentar la columna
}}

// Añadir la restricción al problema
```

```
model.addConstr(expr, GRB.EQUAL, 1.0, "Restr3"); // Añadir la
   restriccion al problema, restricción = 1
bb=0; // Reiniciar la columna
}
aa++; // Aumentar la fila
bb=0; // Reiniciar la columna
}}


// Cuarta restriccion (finalizar en el servicio s+1, para cada dia)
System.out.println("     Restriccion 4: Finalizar en el servicio
   ficticio final (cada dia)");
if(ret == 0) {
int aa = 0, bb = 0;
for(int i = 1; i<n+1; i++){ // Auxiliares
for(int q = 1; q < ndias+1; q++){ // Dias de la semana
GRBLinExpr expr = new GRBLinExpr();
for(int j = 1; j<m+1; j++){ // Instantes de tiempo
for (int k = 0; k < s+2; k++){ // Servicios (finales)
if(mdias[q-1] < j && j <= mdias[q]){ // Si el período pertenece al dia
if(k==0 || k==(s+2-1)){ // Si el servicio l es ficticio
expr.addTerm(1, x[aa+(s+2)*(i-1)][(s+1)+bb*(s+2)]);
} else { // Si el servicio l se realiza el dia q - if
   (diaServicio[k-1]==q)
expr.addTerm(1, x[aa+(s+2)*(i-1)][(s+1)+bb*(s+2)]);
}}
aa++; // Aumentar la fila
}
aa=0; // Reiniciar la fila
bb++; // Aumentar la columna
}

// Añadir la restricción al problema
model.addConstr(expr, GRB.EQUAL, 1.0, "Restr4"); // Añadir la
   restriccion al problema, restricción = 1
aa=0; // Reiniciar la fila
bb=0; // Reiniciar la columna
}}}


// Quinta restriccion (respetar desplazamientos)
System.out.println("     Restriccion 5 pt. 1: Respetar los tiempos de
   desplazamiento y las duraciones de los servicios");
if(ret == 0) {
int contador = 0;
for (int i = 1; i < n+1; i++){ // Auxiliares
for (int j = 1; j < m+1; j++){ // Periodos
for (int k = 1; k < s+1; k++){ // Servicios iniciales
for (int l = 1; l < s+1; l++){ // Servicios finales
if(diaServicio[k-1] == diaServicio[l-1]){ // Solo se tiene en ceunta
   el tiempo de desplazamiento si los servicios se realizan el mismo
   dia
```

```
if(A[1][k-1] <= j && j <= (A[2][k-1] - A[0][k-1])){
int v = j + A[1-1][k-1] + T[k][l]; // Variable v=j+a_{1k}+t_{kl}

if (v <= m){ // Si la variable v es menor que m el sumatorio se puede
   definir correctamente
contador++;
GRBLinExpr expr = new GRBLinExpr();
// Variable x_{ijkl}
expr.addTerm(-1, x[k+(s+2)*(i-1)][l+(s+2)*(j-1)]); // Variable xijlk,
   que tendra coeficiente -1 (lado izquierdo de la desigualdad, por
   tanto con coef -1 si se coloca en el lado derecho)

// Sumatorio de las variables x_{ivlr}
int ndim = (s+2)*m-(s+2)*(v-1); // Tamaño del vector (es decir,
   numero de variables que de la restriccion)
for (int r = 1; r < ndim; r++){ // Indice del vector
expr.addTerm(1, x[(l)+(s+2)*(i-1)][((s+2)*(v-1))+r]); // Variables
   xijlk, que tendra coeficiente 1 (lado derecho de la desigualdad,
   por tanto con coef +1)
}

// Añadir la restricción al problema
model.addConstr(expr, GRB.GREATER_EQUAL, 0.0, "Restr5pt.1"); //
   Añadir la restriccion al problema, restricción = 1
} else { // Si la variable v es mayor que m, el sumatorio no se puede
   definir
x[k+(s+2)*(i-1)][l+(s+2)*(j-1)].set(GRB.DoubleAttr.UB, 0);  // Fijar
   el valor de las variables x_{ijkl} a 0 (cota inferior, superior)}
} else { // Si la variable v es mayor que m, el sumatorio no se puede
   definir
x[k+(s+2)*(i-1)][l+(s+2)*(j-1)].set(GRB.DoubleAttr.UB, 0);  // Fijar
   el valor de las variables x_{ijkl} a 0 (cota inferior, superior)
}

} else { // Si los servicios no son del mismo dia el tiempo de
   desplazamiento entre ellos no se tiene en cuenta, solo la duracion
x[k+(s+2)*(i-1)][l+(s+2)*(j-1)].set(GRB.DoubleAttr.UB, 0);
}}}}}
System.out.println(" restricciones: " + contador);
}


// Restriccion (respetar el orden deespues de la ficticia 0)
System.out.println("     Restriccion 5 pt. 2: Respetar el orden
   despues del servicio ficticio inicial");
if(ret == 0) {
for(int q = 1; q < ndias+1; q++){ // Dias
for (int i = 1; i < n+1; i++){ // Auxiliares
for (int j = mdias[q-1]+1; j < mdias[q]+1; j++){ // Periodos de cada
   dia
for (int l = 1; l < s+2-1; l++){ // Servicio siguiente (no se incluye
   el s+1) ya que puede ser que una auxiliar un dia no trabaje,
```

```
    entonces se tiene la variable x_{ij0s+1}
GRBLinExpr expr = new GRBLinExpr();

// Variable x_{ijkl}
expr.addTerm(1, x[0+(s+2)*(i-1)][l+(s+2)*(j-1)]); // Variable xijlk,
    que tendra coeficiente -1 (lado izquierdo de la desigualdad, por
    tanto con coef -1 si se coloca en el lado derecho)

// Sumatorio de las variables x_{ijlp}
int ndim = (s+2)+1;; // Tamaño del vector (es decir, numero de
    variables que de la restriccion)
for (int kk=1; kk < ndim; kk++){ // Indice del vector
expr.addTerm(-1, x[l+(s+2)*(i-1)][(kk-1)+(s+2)*(j-1)]); // Variables
    xijlk, que tendra coeficiente 1 (lado derecho de la desigualdad,
    por tanto con coef +1)
}

// Añadir la restricción al problema
model.addConstr(expr, GRB.LESS_EQUAL, 0.0, "Restr5pt.2"); // Añadir
    la restriccion al problema, restricción = 1
}}}}}


// Restriccion (no realizar asignaciones veto)
System.out.println("     Restriccion 6: No realizar asignaciones
    vetadas");
if(ret == 0) {
for (int i = 1; i < n+1; i++){ // Auxiliar
for (int j = 1; j < m+1; j++){ // Periodo en que se realiza el
    servicio
for (int k = 1; k < s+2; k++){ // Servicio anterior (k)
for (int l = 1; l < s+2; l++){ // Servicio siguiente (l)
GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1, x[k+(s+2)*(i-1)][l+(s+2)*(j-1)]); // Variable xijlk,
    que tendra coeficiente -1 (lado izquierdo de la desigualdad, por
    tanto con coef -1 si se coloca en el lado derecho)

// Añadir la restricción al problema
model.addConstr(expr, GRB.LESS_EQUAL,afin[i-1][k], "Restr6"); //
    Añadir la restriccion al problema, restricción < Afin
}}}}}


// Septima restriccion (Fijar mu imposibles a 0)
System.out.println("     Restriccion 7 pt. 0: Fijar descansos
    imposibles a 0");
if(ret == 0) {
for(int q=0; q<ndias; q++){ // Dias
if(dias[q]==0){ // Si el numero de variables y_{iq} del dia q es 0
    (numero de formas de obtener un descanso ese dia)
for(int i=0; i<n; i++){ // Auxiliares
```

```
mu[i][q].set(GRB.DoubleAttr.UB, 0); // Fijar el valor de la variable
    a 0 (cota inferior, superior)
}}}}


//Septima restriccion (jornada maxima diaria)
System.out.println("     Restriccion 7 pt. 1: Respetar la jornada
    diaria maxima de las auxiliares");
if(ret == 0) {
for(int q = 0; q < ndias; q++){ // Dias
for (int i = 1; i < n+1; i++){ // Auxiliares
GRBLinExpr expr = new GRBLinExpr();
for(int j = mdias[q]; j < mdias[q+1]; j++){ // Periodos del dia
for(int filas=0; filas<s+2-1; filas++){ // Primer sumando, servicio k
if(filas==0 ){ // Si el servicio es el ficticio inicial
if(!(0 == (s+2)*(j+1) % (s+2))){ // Si la tarea l no es la ficticia
    final
expr.addTerm(j+1, x[filas+(s+2)*(i-1)][(s+2)*(j+1)-1]); // Variable
    x_{ijks+1}}
} else if(filas!=0 && diaServicio[filas-1]==q+1){ // Si la tarea
    filas se realiza el dia q y el servicio no es el ficticio inicial
expr.addTerm((j+1)+(A[0][filas-1]),
    x[filas+(s+2)*(i-1)][(s+2)*(j+1)-1]); // Variable x_{ijks+1}
}}
for(int l=1; l < s+2; l++){ // Segundo sumando, servicio l
if(l==s+2-1 && !(0 == (s+2) % (l+1))){ // Si l es el servicio
    ficticio final y el servicio no es el ficticio final
expr.addTerm(-(j+1), x[0+((i-1)*(s+2))][l+(j*(s+2))]);  // Variable
    x_{ij0l}
} else if(l!=s+2-1 && diaServicio[l-1]==q+1){ // Si l no es el
    ficticio final y el servicio se realiza el dia q
expr.addTerm(-(j+1), x[0+((i-1)*(s+2))][l+(j*(s+2))]); // Variable
    x_{ij0l}
}}}
// Tercer sumando
expr.addTerm(-1, d[i-1][q]); // Variable mu_{iq]

// Añadir la restricción al problema
model.addConstr(expr, GRB.LESS_EQUAL, H[q][i-1], "Restr7pt.1"); //
    Añadir la restriccion al problema, restricción = H[q][i-1]
}}}


// Septima restriccion pt. 2
System.out.println("     Restriccion 7 pt. 2: El mayor de los
    descansos tiene que ser superior a todos los descansos para cada
    auxilair y dia");
if(ret == 0) {
for (int i = 1; i < n+1; i++){ // Auxilaires
for(int q = 0; q < ndias; q++){ // Dias
for (int l = 1; l<s+2-1; l++){ // Servicio l
for(int pp=1; pp<s+2; pp++){ // Segundo sumando, servicio p
```

```
for(int k=1; k<s+2-1; k++){ // Tercer sumando, servicio k
// Mirar si se hacen todos los servicios en el mismo día (q)
if(l != k && l != pp && k!=pp ){ // Si todas las tareas tareas son
    distintas
GRBLinExpr expr = new GRBLinExpr();
if(pp==s+2-1){ // Si la tarea p es la ficticia final
diaServicio[pp-1] = q + 1; // El dia que se realiza la tarea es el
    dia en que estamos (ya que la tarea ficticia final se realiza
    todos los dias)
}
if(diaServicio[l-1]== q+1 && diaServicio[l-1]==diaServicio[k-1] &&
    diaServicio[l-1]==diaServicio[pp-1] &&
    diaServicio[k-1]==diaServicio[pp-1]){ // Si las tareas se realizan
    en el dia q
// Variable mu_{iq}
expr.addTerm(1, mu[i-1][q]); // Primer termino (sumando): Variable
    mu_{iq}

// Sumatorio (en j) de las variables x_{ijlp}
for(int j = mdias[q]; j < mdias[q+1]; j++){ // Periodos del dia q
expr.addTerm(-(j+1), x[l+(s+2)*(i-1)][pp+(s+2)*(j)]);  // Variable
    x_{ijlp}
}

// Sumatorio (en j) de las variables x_{ijkl}
for(int j = mdias[q]; j < mdias[q+1]; j++){ // Periodos del dia q
expr.addTerm(j+1 + T[k][l] + A[0][k-1] - D,
    x[k+(s+2)*(i-1)][l+(s+2)*(j)]);  // Variable x_{ijkl}
}

// Añadir la restricción al problema
model.addConstr(expr, GRB.GREATER_EQUAL, -D, "Restr7pt.2"); // Añadir
    la restriccion al problema, restricción >= -D
}}}}}
// Añadir la restricción que determina mu_{iq} >= 0
GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1, mu[i-1][q]);
model.addConstr(expr, GRB.GREATER_EQUAL, 0, "Restr7pt.2");  // Añadir
    la restriccion al problema (>= 0)
}}}


// Septima restriccion pt. 3
System.out.println("     Restriccion 7 pt. 3: Seleccionar el mayor de
    los descansos para cada auxiliar y dia");
if(ret == 0) {
for (int i = 1; i < n+1; i++){ // Auxiliares
int w = 0; // Inicializar el contador que determina la y_{iqklp} que
    se añade a la restriccion
for(int q = 0; q < ndias; q++){ // Dias
for (int l = 1; l<s+2-1; l++){ // Servicio l
for(int pp=1; pp<s+2; pp++){ // Segundo sumando, servicio p
```

```
for(int k=1; k<s+2-1; k++){ // Tercer sumando, servicio k
// Mirar si se hacen todas en el mismo día
if(l != k && l != pp && k!=pp ){ // Si todos los servicios son
    distintos
GRBLinExpr expr = new GRBLinExpr();
if(pp==s+2-1){ // Si el servicio p es el ficticio final
diaServicio[pp-1] = q + 1; // Se considera que se realiza en el dia q
    (ya que se realiza en todos los dias)
}
if(diaServicio[l-1]== q+1 && diaServicio[l-1]==diaServicio[k-1] &&
    diaServicio[l-1]==diaServicio[pp-1] &&
    diaServicio[k-1]==diaServicio[pp-1]){ // Si todos los servicios se
    realizan el mismo dia q
// Variable mi_{iq}
expr.addTerm(1, mu[i-1][q]); // Variable mu_{iq}

// Sumatorio (en j) de las varables x_{ijlp}
for(int j = mdias[q]; j < mdias[q+1]; j++){ // Periodos del dia q
expr.addTerm(-(j+1) + 0, x[l+(s+2)*(i-1)][pp+(s+2)*(j)]); // variable
    x_{ijlp} (Primer sumando)
}

// Sumatorio (en j) de las variables x_{ijkl}
for(int j = mdias[q]; j < mdias[q+1]; j++){ // Periodos del dia q
expr.addTerm(j+1 + T[k][l] + A[0][k-1] - D,
    x[k+(s+2)*(i-1)][l+(s+2)*(j)]); // Variable x_{ijkl} (segundo
    Sumando)
}

// Añadir la variable y_{iqklp}
if( dias[q] >= 1 ){ // Si tenemos variables y_{iqklp} para el dia q
expr.addTerm(C, y[i-1][w]); // Variable y_{iqklp}
w++; // Aumentar el contador
}
int terminoIndependiente =  C - D; // Termino independiente de la
    sertriccion

// Añadir la restricción al problema
model.addConstr(expr, GRB.LESS_EQUAL, terminoIndependiente,
    "Restr7pt.3"); // Añadir la restriccion al problema (<=C-D)
}}}}}

// Añadir la restricción mu_{iq} <= 0 + C(1-y_{iq}
GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1, mu[i-1][q]); // Variable mu_{iq}
expr.addTerm(C, y_comodin[i-1][q]); // Variable y_{iq}
model.addConstr(expr, GRB.LESS_EQUAL, C, "Restr7pt.3"); // Añadir la
    restriccion al problema (<=C)
}}}


// Septima restriccion pt. 4
```

```
System.out.println("     Restriccion 7 pt. 4: Se tiene que
   seleccionar uno de los descansos");
if(ret == 0) {
for (int i = 1; i < n+1; i++){ // Auxiliares
for(int q = 0; q < ndias; q++){ // Sias
if( dias[q] >= 1 ){ // Si tenemos variables y_{iqklp} para el dia q
GRBLinExpr expr = new GRBLinExpr();
// Sumatorio con las variables y_{iqklp}
for (int w=(int) yDias[q]; w<yDias[q+1]; w++){ // Recorrer las
   variables y_{iqklp} para el dia q
expr.addTerm(1, y[i-1][w]); // Variables y_{iqklp}
}
// Variable y_{iq}
expr.addTerm(1, y_comodin[i-1][q]);  // Variable y_{iq}

// Añadir la restricción al problema
model.addConstr(expr, GRB.EQUAL, 1, "Restr7pt.4"); // Añadir la
   restriccion al problema restricción = 1
} else { // Si no hay variables y_{iqklp} para el dia q
GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1, y_comodin[i-1][q]);  // Variable y_{iq}

// Añadir la restricción al problema
model.addConstr(expr, GRB.EQUAL, 1, "Restr7pt.4"); // Añadir la
   restriccion al problema (=1)
}}}}


// Octava restriccion pt. 1
System.out.println("     Restriccion 8 pt. 1: Definir la variable que
   comprueba si los descansos son mayores que 2");
if(ret == 0) {
for (int i = 1; i < n+1; i++){ // Auxiliares
for(int q = 0; q < ndias; q++){ // Dias
// Añadir la restricción z_{iq} <= 1/2*mu_{iq}
GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1, z[i-1][q]); // Variable z_{iq}
expr.addTerm((double)-1/mayorDescanso, mu[i-1][q]); // Variable
   -0.5*mu_{iq}
model.addConstr(expr, GRB.LESS_EQUAL, 0, "Restr8pt.1"); // Añadir la
   restriccion al problema (<=0)
}}}


// Octava restriccion pt. 2
System.out.println("     Restriccion 8 pt. 2: Definir la variable que
   comprueba si los descansos son mayores que 2");
if(ret == 0) {
for (int i = 1; i < n+1; i++){ // Auxiliares
for(int q = 0; q < ndias; q++){ // Dias
// Añadir la restricción z_{iq} <= 1/2*mu_{iq}
GRBLinExpr expr = new GRBLinExpr();
```

```
expr.addTerm(E, z[i-1][q]); // Variable z_{iq}
expr.addTerm((double)-1/mayorDescanso, mu[i-1][q]); // Variable
    -0.5*mu_{iq}
model.addConstr(expr, GRB.GREATER_EQUAL, -1, "Restr8pt.2"); // Añadir
    la restriccion al problema (>=-1)
}}}


// Novena restriccion pt. 1
System.out.println("     Restriccion 9 pt. 1: Seleccionar los
    descansos son mayores que 2");
if(ret == 0) {
for (int i = 1; i < n+1; i++){ // Auxiliares
for(int q = 0; q < ndias; q++){ // Dias
// Añadir la restricción z_{iq} <= 1/2*mu_{iq}
GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1, d[i-1][q]); // Variable d_{iq}
expr.addTerm(-E, z[i-1][q]); // Variable -E*mu_{iq}
model.addConstr(expr, GRB.LESS_EQUAL, 0, "Restr9pt.1"); // Añadir la
    restriccion al problema (<=0)
}}}


// Novena restriccion pt. 2
System.out.println("     Restriccion 9 pt. 2: Seleccionar los
    descansos son mayores que 2");
if(ret == 0) {
for (int i = 1; i < n+1; i++){ // Auxiliares
for(int q = 0; q < ndias; q++){ // Dias
// Añadir la restricción z_{iq} <= 1/2*mu_{iq}
GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1, d[i-1][q]); // Variable d_{iq}
expr.addTerm(-1, mu[i-1][q]); // Variable -E*mu_{iq}
model.addConstr(expr, GRB.LESS_EQUAL, 0, "Restr9pt.2"); // Añadir la
    restriccion al problema (<=0)
}}}


// Septima restriccion pt. 9
System.out.println("     Restriccion 9 pt. 3: Seleccionar los
    descansos son mayores que 2");
if(ret == 0) {
for (int i = 1; i < n+1; i++){ // Auxiliares
for(int q = 0; q < ndias; q++){ // Dias
// Añadir la restricción z_{iq} <= 1/2*mu_{iq}
GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1, d[i-1][q]); // Variable d_{iq}
expr.addTerm(-1, mu[i-1][q]); // Variable -E*mu_{iq}
expr.addTerm(-E, z[i-1][q]); // Variable -E*mu_{iq}
model.addConstr(expr, GRB.GREATER_EQUAL, -E, "Restr9pt.3"); // Añadir
    la restriccion al problema (>=-E)
}}}
```

```
// Septima restriccion pt. 10
System.out.println("     Restriccion 9 pt. 4: Seleccionar los
   descansos son mayores que 2");
if(ret == 0) {
for (int i = 1; i < n+1; i++){ // Auxiliares
for(int q = 0; q < ndias; q++){ // Dias
// Añadir la restricción z_{iq} <= 1/2*mu_{iq}
GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1, d[i-1][q]); // Variable d_{iq}
//                                        System.out.println("i: " + i
   + " q: " + q +  " coef: " + 1 + " " +
   d[i-1][q].get(GRB.StringAttr.VarName));
model.addConstr(expr, GRB.GREATER_EQUAL, 0, "Restr9pt.4"); // Añadir
   la restriccion al problema (>=0)
}}}


//Octava restriccion (jornada maxima semanal)
System.out.println("     Restriccion 10: Respetar la jornada semanal
   maxima de las auxiliares");
if(ret == 0) {
for (int i = 1; i < n+1; i++){ // Auxiliares
GRBLinExpr expr = new GRBLinExpr();
for(int q = 0; q < ndias; q++){ // Dias
for(int j = mdias[q]; j < mdias[q+1]; j++){ // Periodos del dia q
// Primer sumando
for(int filas=0; filas<s+2-1; filas++){// Primer sumando, servicio k
if(filas==0 ){ // Si el servicio es el ficticio inicial
if(!(0 == (s+2)*(j+1) % (s+2))){ // Si la tarea l no es la ficticia
   final
expr.addTerm(j+1, x[filas+(s+2)*(i-1)][(s+2)*(j+1)-1]); // Variable
   x_{ijks+1}}
} else if(filas!=0 && diaServicio[filas-1]==q+1){ // Si la tarea
   filas se realiza el dia q y el servicio no es el ficticio inicial
expr.addTerm((j+1)+(A[0][filas-1]),
   x[filas+(s+2)*(i-1)][(s+2)*(j+1)-1]); // Variable x_{ijks+1}
}}
// Segundo sumando
for(int l=1; l < s+2; l++){ // Segundo sumando, servicio l
if(l==s+2-1 && !(0 == (s+2) % (l+1))){ // Si l es el servicio
   ficticio final y el servicio no es el ficticio final
expr.addTerm(-(j+1), x[0+((i-1)*(s+2))][l+(j*(s+2))]); // Variable
   x_{ij0l}
} else if(l!=s+2-1 && diaServicio[l-1]==q+1){ // Si l no es el
   ficticio final y el servicio se realiza el dia q
expr.addTerm(-(j+1), x[0+((i-1)*(s+2))][l+(j*(s+2))]); // Variable
   x_{ij0l}
}}}
// Tercer sumando
expr.addTerm(-1, d[i-1][q]); // Variable mu_{iq}
```

```
}
// Añadir la restricción al problema
model.addConstr(expr, GRB.LESS_EQUAL, H[ndias][i-1], "Restr10"); //
    Añadir la restriccion al problema (<=H)
}}


// --------------- FUNCION OBJETIVO ---------------

// Determinar la funcion objetivo
System.out.println("\nCalculando: Funcion Objetivo");
GRBLinExpr expr = new GRBLinExpr(); // Iniciar la expresion de la
    funcion objetivo


// Variables x_{ijkl}
boolean primero, segundo, tercero, cuarto, quinto, sexto, septimo;
int aa = 0, bb = 0;
for (int i = 1; i < n+1; i++){ // Auxiliares
for(int k = 0; k < s+2; k++){ // Tarea que se realiza k
for (int j = 1; j < m+1; j++){ // Periodo en que se realiza j
for (int l = 0; l < s+2; l++){ // Servicio siguiente l
// Añadir los desplazamientos y las afinidades
if (k==l || k == (s+2-1) || l == 0 || (k==0 && l==(s+2-1)) ){
// Si los sercicios son iguales, l es el ficticio inicial, k es el
    ficiticio final o k es el ficticio inicial y l el ficticio fianal
expr.addTerm(peso_tiemp*0 - peso_afin*0,
    x[aa+(s+2)*(i-1)][bb*(s+2)+l]);  // El coeficiente en la funcion
    objetivo es 0
} else if(l == (s+2-1)){ // Si el servicio es el ficticio final
primero = (k != 0); // k no es el ficticio inicial
segundo = (j >= (A[3][k-1])); // j es mayot que el inicio de la
    ventana optima
tercero = ((j + (A[0][k-1]) ) <= (A[4][k-1])); // j + duracion es
    menor que el final de la ventana optima
if(primero && segundo && tercero){ // Si j esta en la ventana optima
    [a_{4k}, a_{5k}], entonces se añade la cosntante peso_vopt (w).
    NOTA: se usa k-1 ya que en la matriz A solo estan los servicios NO
    FICTICIOS
expr.addTerm(peso_tiemp*(j+A[0][k-1]) - peso_afin*afin[i-1][k] -
    peso_vopt, x[aa+(s+2)*(i-1)][bb*(s+2) + l]); // El coeficiente en
    la funcion objetivo es
// (j + duracion del servicio) menos la afinidad menos el peso de
    pertenecer a la ventana optima (para poder minimizar)
} else { // Si j no esta en la ventana optima no se añade el peso
expr.addTerm(peso_tiemp*(j+A[0][k-1]) - peso_afin*afin[i-1][k] - 0,
    x[aa+(s+2)*(i-1)][bb*(s+2) + l]);  // El coeficiente en la fucnion
    objetivo es
// (j + duracion del servicio) menos la afinidad menos el peso de
    vopt, 0, (para poder minimizar)}
} else if(k==0){ // Si k es la ficticia inicial
```

```
expr.addTerm(peso_tiemp*(-j) - peso_afin*afin[i-1][k] - 0,
   x[aa+(s+2)*(i-1)][bb*(s+2) + l]); // El coeficiente en la fucnion
   objetivo es -j menos la afinidad menos el peso de vopt, 0, (para
   poder minimizar)
} else { // En otro caso (ningino de los servicios es ficticio)
cuarto = (k != 0);
quinto = (k != (s+2-1));
sexto = (j >= (A[3][k-1]));
septimo = ( (j + (A[0][k-1]) ) <= (A[4][k-1]));
if(cuarto && quinto && sexto && septimo){ // Si j + A[0][k] esta en
   la ventana optima [a_{4k}, a_{5k}], entonces se añade la cosntante
   peso_vopt (w). NOTA: se usa k-1 ya que en la matriz A solo estan
   los servicios NO FICTICIOS
expr.addTerm(peso_tiemp*0 - peso_afin*afin[i-1][k] - peso_vopt +
   peso_despl*T[k][l], x[aa+(s+2)*(i-1)][bb*(s+2) + l]);  // El
   coeficiente en la fucnion
// objetivo es la afinidad menos el peso de pertenecer a la ventana
   optima (para poder minimizar) + el tiempo de desplazamiento
} else { // Si j no esta en la ventana optima no se añade el peso
expr.addTerm(peso_tiemp*0 - peso_afin*afin[i-1][k] - 0 +
   peso_despl*T[k][l], x[aa+(s+2)*(i-1)][bb*(s+2) + l]); // El
   coeficiente en la funcion objetivo es la afinidad  +
// el desplazamietno entre los servicios
}}}
bb++; // Aumentar la columna
}
aa++; // Aumentar la fila
bb=0; // Reiniciar la columna
}
aa = 0; // Reiniciar la fila
}

// Variables mu
for (int i = 1; i < (n+1); i++){ // Auxiliar
for(int q=1; q<(ndias+1); q++){ // Dia de la semana
expr.addTerm(0, mu[i-1][q-1]);  // Añadir coeficiente para las nuevas
   variables mu
}}

// Variables y
for (int i = 1; i < (n+1); i++){ // Auxiliar
for(int q=1; q<(yTotal+1); q++){ // Variables y
expr.addTerm(0, y[i-1][q-1]);  // Añadir coeficiente para las nuevas
   variables mu
}}

// Variables y comodin
for (int i = 1; i < n+1; i++){ // Auxiliar
for(int q=1; q<ndias+1; q++){ // Dia de la semana
expr.addTerm(0, y_comodin[i-1][q-1]);  // Añadir coeficiente para las
   nuevas variables mu
}}
```

```
// Variables z
for (int i = 1; i < (n+1); i++){ // Auxiliar
for(int q=1; q<(ndias+1); q++){ // Dia de la semana
expr.addTerm(0, z[i-1][q-1]);  // Añadir coeficiente para las nuevas
    variables mu
}}

// Variables d
for (int i = 1; i < (n+1); i++){ // Auxiliar
for(int q=1; q<(ndias+1); q++){ // Dia de la semana
expr.addTerm(-peso_tiemp, d[i-1][q-1]);  // Añadir coeficiente para
    las nuevas variables mu
model.update();
}}

// Añadir la funcion objetivo al modelo
model.setObjective(expr, GRB.MINIMIZE);

// ----- Guardar el problema -----
model.update();
model.write(archivo + ejemplo + "/" +"problema.lp"); // Guardar el
    problema
//--------------------------------

long start=System.currentTimeMillis(); // Contador de tiempo

// -------- Print ---------
System.out.println(" ");
System.out.println("Resolviendo el problema...");
System.out.println(" ");
// ------------------------

// Optimizar el modelo
model.optimize();

// -------- Print ---------
System.out.println( "La solución obtenida es: ");
System.out.println("Valor de la función objetivo: " +
    model.get(GRB.DoubleAttr.ObjVal)); // Funcion objetivo
// ------------------------

// ----- Guardar la solucion -----
model.write(archivo + ejemplo + "/" +"solucion.sol"); // Guardar la
    solución
//--------------------------------


// Resultado
aa = 0; bb = 0;
int[][] solucion = new int [s][7]; // Matriz en la que se almacena la
    solucion de las variables x_{ijkl}
```

```
int r = 0; // Indicador de la fila de la matriz en la que nos
   encontramos
List<String> variables = new ArrayList<String>();
for (int i = 1; i < n+1; i++){ // Auxiliare
for(int k = 0; k < s+2; k++){ // Tarea que se realiza k
for (int j = 1; j < m+1; j++){ // Periodo en que se realiza j
for (int l = 0; l < s+2; l++){ // Servicio siguiente l
int dia = 0, período_dia = 0;
if(Math.round(x[aa+(s+2)*(i-1)][bb*(s+2)+l].get(GRB.DoubleAttr.X)) !=
   0){
for(int q = 0; q < ndias; q++){ // Dias
if(j > mdias[q] && j <= mdias[q+1]){ // Si el período esta en el dia q
dia = q+1; // dia en que se encuentra el período
período_dia = j - mdias[q]; // período del dia en que se comienza el
   servicio
//Nota: mdias marca el ultimo período del dia
}}
if (k != 0){ // Solamente queremos los servicios reales, no los
   ficticios
solucion[r][0] = k; // Almacenar el servicio
solucion[r][1] = i; // Almacenar la auxiliar
solucion[r][2] = j; // Almacenar el período de la semana
solucion[r][3] = dia; // Almacenar el dia de la semana
solucion[r][4] = período_dia; // Almacenar el período del dia
double período_real = (double)((período_dia-1)*5)/60;
int hora_real = 6 + (int) período_real;
double decimal = (período_real - (int) período_real);
int minuto_real = (int)(Math.round(decimal*60));
solucion[r][5] = hora_real;
solucion[r][6] = minuto_real;
// Mostrar los resultados por pantalla
   x[aa+(s+2)*(i-1)][bb*(s+2)+l].get(GRB.StringAttr.VarName)
   Variable x" + i + "" + j + "" + k + "" + l + "        "+
System.out.println("Variable x" +
   x[aa+(s+2)*(i-1)][bb*(s+2)+l].get(GRB.StringAttr.VarName) + " "+
" Es decir, la auxiliar " + i +
" comienza el servicio " + k + " en el período " + j +" (período " +
   período_dia + " hora: " + hora_real + ":" + minuto_real + " del
   dia " + dia +
") y seguidamente se traslada al servicio " + l );
r++; // Aumentar el contador de numero de fila
} else {
double período_real = (double)((período_dia-1)*5)/60;
int hora_real = 6 + (int) período_real;
double decimal = (período_real - (int) período_real);
int minuto_real = (int)(Math.round(decimal*60));
// Mostrar los resultados por pantalla
System.out.println("Variable x"
   +x[aa+(s+2)*(i-1)][bb*(s+2)+l].get(GRB.StringAttr.VarName) + "  "
   +  " Es decir, la auxiliar " + i +
" comienza el servicio " + k + " en el período " + j +" (período " +
   período_dia + " hora: " + hora_real + ":" + minuto_real + " del
```

```
    dia " + dia +
") y seguidamente se traslada al servicio " + l );
}
variables.add(x[aa+(s+2)*(i-1)][bb*(s+2)+l].get(GRB.StringAttr.VarName));
}}
bb++; // Aumentar la columan
}
aa++; // Aumentar la fila
bb=0; // Reiniciar la columna
}
aa = 0; // Reiniciar la fila
}

// Variables mu
double[][] solucion_descanso = new double [n*ndias][3]; // Matriz en
    la que se almacena la solucion de los descansos
int rr = 0; // Indicador de la fila de la matriz en la que nos
    encontramos
for (int i = 1; i < n+1; i++){ // Auxiliar
for(int q=1; q<ndias+1; q++){ // Dia de la semana
// Solucion (para almacenar los resultados en un fichero)
solucion_descanso[rr][0] = i; // Almacenar la auxiliar
solucion_descanso[rr][1] = q; // Almacenar el dia
solucion_descanso[rr][2] = mu[i-1][q-1].get(GRB.DoubleAttr.X); //
    Almacenar el descanso
rr++; // Aumentar el contador de numero de fila
System.out.println("El descanso de la auxiliar " + i + " en el dia "
    + q+  " es de " + mu[i-1][q-1].get(GRB.DoubleAttr.X) + " minutos
    -> "
+ d[i-1][q-1].get(GRB.DoubleAttr.X) + " minutos"  );
}}


// -------- Tiempo de ejecucion --------
long finish=System.currentTimeMillis(); // Contador de tiempo
long duration=finish-start; // Tiempo de ejecucion
double segundos = duration * 0.001; // Tiempo de ejecucion en segundos
double minutos = segundos/60; // Tiempo de ejecucion en segundos
System.out.println("\nTiempo de ejecucion: " + duration + "
    millisegundos");
System.out.println("Tiempo de ejecucion: " + segundos + " segundos");
System.out.println("Tiempo de ejecucion: " + minutos + " minutos");
// ------------------------------------


// Guardar los resultados de x_{ijkl} en un fichero
try{
CSVWriter writer = new CSVWriter(new FileWriter(archivo + ejemplo +
    "/" + "resultados.csv"), ';'); // Crear un writer

String[] colnames = {"servicio", "auxiliar", "período_general",
    "dia", "período_dia", "hora_dia", "minuto_dia"}; // Nombres de las
```

```
    columnas del fichero
writer.writeNext(colnames); // Añadir los nombres de las columnas al
    documento

for ( int fila = 0; fila < s; fila++){ // Servicios reales
String[] resul = {String.valueOf(solucion[fila][0]),
    String.valueOf(solucion[fila][1]),
    String.valueOf(solucion[fila][2]),
    String.valueOf(solucion[fila][3]),
    String.valueOf(solucion[fila][4]),
    String.valueOf(solucion[fila][5]),
    String.valueOf(solucion[fila][6])};
// String que almacena los valores de la solucion para cada servicio
writer.writeNext(resul); // Añadir la solucion al fichero
}
writer.close(); // Cerrar el writer

// Mensaje para indicar que se ha creado el fichero
System.out.println(" ");
System.out.println("El fichero con los resultados se ha creado
    correctamente.");

} catch (IOException e) {
e.printStackTrace();
}

// Guardar los resultados de mu_{iq}
try{
CSVWriter writer = new CSVWriter(new FileWriter(archivo + ejemplo +
    "/" + "resultados_descansos.csv"), ';'); // Crear un writer

String[] colnames = {"auxiliar", "dia", "descanso"}; // Nombres de
    las columnas del fichero
writer.writeNext(colnames); // Añadir los nombres de las columnas al
    documento

for ( int fila = 0; fila < n*ndias; fila++){ // auxiliares
String[] resul = {String.valueOf(solucion_descanso[fila][0]),
    String.valueOf(solucion_descanso[fila][1]),
    String.valueOf(solucion_descanso[fila][2])};
// String que almacena los valores de la solucion mu para cada
    auxiliar
writer.writeNext(resul); // Añadir la solucion al fichero
}
writer.close(); // Cerrar el writer

// Mensaje para indicar que se ha creado el fichero
System.out.println("El fichero con los resultados se ha creado
    correctamente.");

} catch (IOException e) {
e.printStackTrace();
```

```
}

// Guardar variables
try{
CSVWriter writer = new CSVWriter(new FileWriter(archivo + ejemplo +
    "/" + "resultados_variables.csv"), ';'); // Crear un writer

String[] colnames = {"variables"}; // Nombres de las columnas del
    fichero
writer.writeNext(colnames); // Añadir los nombres de las columnas al
    documento

for(String var: variables){
String[] variable = {var};
// String que almacena los valores de la solucion mu para cada
    auxiliar
writer.writeNext(variable); // Añadir la solucion al fichero
}
writer.close(); // Cerrar el writer

// Mensaje para indicar que se ha creado el fichero
System.out.println("El fichero con las variables se ha creado
    correctamente.");

} catch (IOException e) {
e.printStackTrace();
}

// Guardar funcion objetivo
try{
CSVWriter writer = new CSVWriter(new FileWriter(archivo + ejemplo +
    "/" + "resultados_objetivo.csv"), ';'); // Crear un writer

String[] colnames = {"objetivo"}; // Nombres de las columnas del
    fichero
writer.writeNext(colnames); // Añadir los nombres de las columnas al
    documento

String[] fo = {String.valueOf(model.get(GRB.DoubleAttr.ObjVal))};
// String que almacena los valores de la solucion mu para cada
    auxiliar
writer.writeNext(fo); // Añadir la solucion al fichero
writer.close(); // Cerrar el writer

// Mensaje para indicar que se ha creado el fichero
System.out.println("El fichero con la funcion objetivo se ha creado
    correctamente.");

} catch (IOException e) {
e.printStackTrace();
}
```

```
// Guardar tiempos de ejecucion
try{
CSVWriter writer = new CSVWriter(new FileWriter(archivo + ejemplo +
    "/" + "resultados_tiempos.csv"), ';'); // Crear un writer

String[] colnames = {"milisegundos", "segundos", "minutos"}; //
    Nombres de las columnas del fichero
writer.writeNext(colnames); // Añadir los nombres de las columnas al
    documento

String[] resul = {String.valueOf(duration), String.valueOf(segundos),
    String.valueOf(minutos)};
// String que almacena los valores de la solucion mu para cada
    auxiliar
writer.writeNext(resul); // Añadir la solucion al fichero
writer.close(); // Cerrar el writer

// Mensaje para indicar que se ha creado el fichero
System.out.println("El fichero con los tiempos se ha creado
    correctamente.");

} catch (IOException e) {
e.printStackTrace();
}

// Dispose of model and environment
model.dispose();
env.dispose();

} catch (GRBException e) {
System.out.println("Error code: " + e.getErrorCode() + ". " +
e.getMessage());
}
}}
```

Bibliografía

Dantzig, George B, & Ramser, John H. 1959. The truck dispatching problem. *Management science*, **6**(1), 80–91.

Dantzig, George B, & Wolfe, Philip. 1960. Decomposition principle for linear programs. *Operations research*, **8**(1), 101–111.

Dowsland, Kathryn A. 1998. Nurse scheduling with tabu search and strategic oscillation. *European journal of operational research*, **106**(2-3), 393–407.

Glover, Fred. 1998. A template for scatter search and path relinking. *Lecture notes in computer science*, **1363**, 13–54.

Henderson, Darrall, Jacobson, Sheldon H, & Johnson, Alan W. 2003. The theory and practice of simulated annealing. *Pages 287–319 of: Handbook of metaheuristics*. Springer.

Holland, John. 1975. Adaptation in artificial and natural systems. *Ann Arbor: The University of Michigan Press*.

Johnson, David S, Aragon, Cecilia R, McGeoch, Lyle A, & Schevon, Catherine. 1991. Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning. *Operations research*, **39**(3), 378–406.

Kirkpatrick, Scott, Gelatt, C Daniel, & Vecchi, Mario P. 1983. Optimization by simulated annealing. *science*, **220**(4598), 671–680.

Koeleman, Paulien M, Bhulai, Sandjai, & van Meersbergen, Maarten. 2012. Optimal patient and personnel scheduling policies for care-at-home service facilities. *European Journal of Operational Research*, **219**(3), 557–563.

Metropolis, Nicholas, Rosenbluth, Arianna W, Rosenbluth, Marshall N, Teller, Augusta H, & Teller, Edward. 1953. Equation of state calculations by fast computing machines. *The journal of chemical physics*, **21**(6), 1087–1092.

Mladenović, Nenad, & Hansen, Pierre. 1997. Variable neighborhood search. *Computers & operations research*, **24**(11), 1097–1100.

Moscato, Pablo, *et al.* 1989. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, **826**, 1989.

Nickel, Stefan, Schröder, Michael, & Steeg, Jörg. 2012. Mid-term and short-term planning support for home health care services. *European Journal of Operational Research*, **219**(3), 574–587.

Pullen, HGM, & Webb, MHJ. 1967. A computer application to a transport scheduling problem. *The Computer Journal*, **10**(1), 10–13.

Rasmussen, Matias Sevel, Justesen, Tor, Dohn, Anders, & Larsen, Jesper. 2012. The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. *European Journal of Operational Research*, **219**(3), 598–610.

Rendl, Andrea, Prandtstetter, Matthias, Hiermann, Gerhard, Puchinger, Jakob, & Raidl, Günther. 2012. Hybrid heuristics for multimodal homecare scheduling. *Pages 339–355 of: International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer.

Van Laarhoven, Peter JM, & Aarts, Emile HL. 1987. Simulated annealing. *Pages 7–15 of: Simulated annealing: Theory and applications*. Springer.