APLICACIÓN DE TÉCNICAS INTELIGENTES PARA LA RESOLUCIÓN DE PROBLEMAS

GUILLERMO ESQUIVEL GONZÁLEZ

MÁSTER EN INTELIGENCIA ARTIFICIAL, RECONOCIMIENTO DE FORMAS E IMAGEN DIGITAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MIARFID - Departamento de Sistemas Informáticos y Computación - Curso 2018/2019

Septiembre de 2019

Tutor:

Federico Barber Sanchis

Autorización de difusión

Guillermo Esquivel González

01/07/2019

Él abajo firmante, matriculado en el Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital del Departamento de Sistemas Informátiocos y Computación de la Universidad Politécnica de Valencia, autoriza a la Universidad Politécnica de Valencia (UPV) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Resolución de problemas de optimización de asignación de recursos en transporte", realizado durante el curso académico 2018-2019 bajo la dirección de Federico Barber Sanchis en el Dpto. Sistemas Informáticos y Computación.

Resumen en castellano

El principal objetivo que persigue este proyecto es crear una herramienta capaz de realizar una reorganización de la jornada laboral de los conductores de la empresa de transporte público TITSA en caso de que falte un conjunto de conductores. Esta reorganización esta sujeta a distintas restricciones propias de la logística de la empresa y tiene como objetivo maximizar el número de pasajeros recogidos. La implantación de esta herramienta aporta grandes beneficios tanto a TITSA, contando con una herramienta analítica muy potente, como a sus usuarios, garantizando el mejor servicio posible en todo momento.

La organización de una jornada laboral en TITSA se plantea de tal forma que a cada conductor se le asignan una serie de viajes o servicios que debe de realizar dicho día. Al tratarse de una empresa de grande dimensiones, en la que se cuenta con más de 1600 empleados, es común encontrarse con que algún día un conductor no pudo asistir a trabajar. En tal caso todos los servicios que le habían sido asignados serán suspendidos. Es aquí donde surge la problemática abordada en este trabajo en el que se utiliza un modelo matemático para resolver el problema de la mejor forma posible. Específicamente, se pretende hacer un uso inteligente de los recursos disponibles, recogiendo al mayor número de pasajeros posible. Esto, se consigue suspendiendo aquellos servicios que se hayan previsto que tengan un bajo número de pasajeros.

Para conocer el número de pasajeros o usuarios que tendrá un determinado servicio se han utilizado técnicas de aprendizaje automático. A partir de los datos históricos que proporcionó TITSA se desarrollo un perceptrón multicapa (MLP) capaz de predecir el número de pasajeros por hora que tendrá una determina línea de servicio un día. De esta forma, se puede tener una aproximación muy exacto con la que alimentar el modelo matemático encargado de reorganizar la jornada laboral.

En la medida en la que se dispone de datos reales para el desarrollo del trabajo, los resultados obtenidos muestran claramente el funcionamiento de ambas herramientas. Con el objetivo de evaluar la calidad de la propuesta planteada para abordar el problema, se lleva a cabo una fase experimental. Las conclusiones extraídas de los experimentos revelan que la herramienta es capaz de abordar problemas de gran tamaño en un corto periodo de tiempo aportando soluciones que mejoran de manera notable el número de pasajeros recogidos. Por otra parte, el uso de la aplicación desarrollada en este trabajo permite realizar análisis de situaciones críticas y crear patrones de actuación en caso de faltas de recursos.

Palabras clave

Optimización, logística, Cplex, forecasting OPL , Machine Learning, aprendizaje automático, Big Data, Inteligencia Artificial.

Índice general

Agradecimientos

En primer lugar agradecer a mi familia por brindarme la oportunidad de poder estudiar lo que siempre me ha gustado y apoyarme hasta el final de mis estudios. Además me gustaría agradecer a mi tutor la atención y profesionalidad que han tenido durante todo el desarrollo del trabajo.

Capítulo 1

Descripción y Motivación

Este capítulo trata de describir la problemática de la cual surge el presente TFM y la motivación por la que es de interés resolverla. En primer lugar se detalla brevemente el problema abordado así como sus principales características y recursos. Tras esto, se continua explicando la motivación de este problema y finalmente, a modo de comparativa se describe el estado del arte, haciendo un breve análisis de las distintas técnicas de optimización que existen.

1.1. Descripción General

Un factor importante a tener en cuenta es el escenario en el que se plantea el problema. El presente trabajo ha sido desarrollado para la empresa de transporte público de Tenerife, TITSA. Esta empresa de transporte es la tercera más grande a nivel de España, cuenta con 1600 empleados de los cuales hay alrededor de 150 administrativos, 200 mecánicos y el resto conductores. Además, se trata de una empresa pública perteneciente al cabildo de Tenerife. El hecho de realizar el trabajo en un escenario real aporta además de una aplicabilidad directa al entorno, ciertas complicaciones en el desarrollo que se irán detallando a lo largo de la memoria.

La planificación de una jornada de trabajo para la empresa de transporte público Titsa se plantea de tal forma que a cada conductor se le asigna un conjunto de servicios de una o varias líneas distintas, entendiendo como servicio a cada viaje o expedición que realiza el

autobús asignado a este. Estos viajes recogen un determinado número de pasajeros. Existen un conjunto de viajes de gran importancia dado que el número de pasajeros que recogen es mucho mayor. Normalmente los viajes pertenecientes a este conjunto son realizados en horas punteras y de líneas cuyos itinerarios pasan por grandes poblaciones.

En el caso en el que uno o varios conductores falten a su trabajo un determinado día, los servicios que le habían sido asignados quedan ahora descubiertos. Como consecuencia, todos los pasajeros que iban a recoger estos servicios serán desatendidos. Es aquí donde se plantea el problema modelado en este trabajo, la reasignación de conductores a servicios en el caso de que falte uno o varios de ellos de tal forma que el número de pasajeros que se dejen sin recoger sea el menor posible. De esta forma, se plantea un problema de optimización de asignación de recursos con un único objetivo ya mencionado, minimizar el número de pasajeros sin recoger.

1.2. Motivación

Actualmente la planificación de las jornadas de trabajo es realizada por un software externo, llamado CODICE. Para realizar dicha planificación, se le indican al software todos los viajes que deben de ser realizados y este devuelve una asignación conductor-viaje. Esta metodología tiene una serie de ventajas y desventajas, por ejemplo el hecho de incluir nuevas consideraciones a la planificación o conocer su funcionamiento interno y criterios de optimización.

Por otra parte, una vez se dispone de la planificación de una jornada y surge un problema, como puede ser la falta de un conductor de autobús, la reorganización es llevada a cabo por el jefe del área afectada, el cual realiza los cambios pertinentes en el momento según su criterio y experiencia. Esta metodología puede causar que expediciones o viajes que iban a recoger a una gran cantidad de pasajeros queden ahora sin un conductor asignado. De la misma forma, pueden aparecer otros problemas como el incumplimiento del horario de descanso de alguno de los conductores o la imposibilidad real de realizar la nueva planificación, planteada

por el jefe de área, por cuestiones de desplazamiento entre los puntos de salida y llegada de las expediciones asignadas a un determinado conductor.

Con el afán de dar un mejor servicio y optimizar en la mayor medida el uso de los recursos disponibles se plantea este problema del cual, a su vez, surge otro problema de gran interés para la empresa. Debido a que se pretende realizar una nueva planificación en la que se recoja el mayor número de pasajeros posibles, se hace necesario contar con una estimación del número de pasajeros que tendrá un determinado viaje. Es por eso que se plantea el desarrollo de una herramienta que trata de predecir el número de usuarios por hora que tendrá cada línea. Además, Estos datos son de gran valor para otros trabajos de la empresa y al mismo tiempo permiten hacer una reorganización de la jornada más adecuada a los intereses de los usuarios o pasajeros.

1.3. Estado del arte

En este epígrafe se hace una clasificación, en función del tipo de problema, de artículos e investigaciones importantes relacionados con la planificación de horarios.

Cabe destacar que el tema que se aborda es de gran estudio y tiene un gran abanico de variantes, así como de técnicas utilizadas para abordar a los problemas de esta naturaleza. Es por esto que el contenido bibliográfico utilizado es tan amplio.

La revisión bibliográfica realizada se ha hecho desde un punto de vista mucho más genérico pretendiendo así encontrar distintos puntos de vista para plantear el problema y técnicas aplicadas a resolver problemas de la misma tipología.

A continuación, se propone una clasificación según las características del problema :

- **Número de personas por patrón**

  Se centran en un modelo que planifica los periodos libres de toda la plantilla de tal forma que se pueda asegurar que cada jornada trabajen los empleados suficientes para poder cubrir todos los servicios.

La principal característica de los modelos de este tipo es que en caso de que el personal realice distintos tipos de actividad, este no determinará que actividad va a llevar a cabo cada empleado sino simplemente los periodos libres que tendrá.

Por otra parte, cabe destacar que los patrones son datos de partida ya determinados. De esta forma, el objetivo es definir de entre todos los patrones disponibles cuales serán utilizados, que número de veces o que empleado realizará cada patrón.

La aplicación utilizada en TITSA, llamada CODICE, para generar la planificación de las jornadas de trabajo utiliza este planteamiento.

- **Asignación de personas**

  Los problemas pertenecientes a esta categoría, en lugar de determinar el tamaño de la plantilla necesaria para realizar los servicios de transporte, se encargan de concretar el número de personas que realizarán cada una de las tareas.

  Se puede entender como tareas a patrones de días libres, tipos de tareas, turnos horarios, etc...

- **Agrupación de jornadas de trabajo**

  En este caso, se pretende definir una serie de subconjuntos disjuntos que agrupen las jornadas laborales de tal forma que cada jornada solo pertenezca a un solo grupo. De esta forma, una vez se hayan creado todos los grupos de jornadas laborales se trata de asignar a cada grupo un trabajador.

  Por lo general esta metodología es propia de compañías aéreas en las que en lugar de hablar de jornadas laborales se habla directamente de vuelos.

- **Creación de patrones**

  Los problemas englobados en esta tipología tratan de encontrar los patrones más adecuados para cubrir la demanda usando el mínimo número de personal. Se pretende

que la carga de trabajo este repartida equitativamente entre todos los trabajadores y que la satisfacción de estos sea la máxima posible.

1.3.1. Técnicas de optimización

Son herramientas matemáticas que han sido específicamente diseñadas para encontrar una o varias soluciones que maximicen o minimicen un conjunto de objetivos establecidos para un problema de optimización dado. Existen una gran cantidad de técnicas de optimización que pueden ser aplicadas para resolver un mismo problema. Por otro lado, no existe un único método que sea capaz de resolver cualquier tipo de problema de optimización en un tiempo razonable. Debido a esto, el campo de la optimización y sus técnicas está en continuo estudio y crecimiento. Una clasificación de las técnicas de optimización en función de su naturaleza podría ser la siguiente:

- **Deterministas** : son aquellos modelos matemáticos donde conocida la entrada se puede conocer la salida. En estos métodos no interviene el azar por lo que para una misma entrada siempre se producirá la misma salida. Formalmente, estos métodos pueden ser definidos como una máquina de estados de tal forma que para una misma entrada la máquina siempre pasa por los mismo estados para devolver una salida.

- **Estocásticas** : agrupa a todos los modelos de optimización en donde intervienen componentes aleatorios. Estos modelos utilizan un carácter estocástico sobre alguno de los parámetros de entrada por lo que a diferencia de los métodos deterministas no se puede determinar la salida del método aún conociendo la entrada.

A continuación se presenta una clasificación de los métodos de optimización más comunes según su naturaleza. Algunas de estás técnicas serán descritas con mayor detalle posteriormente en este mismo capítulo.

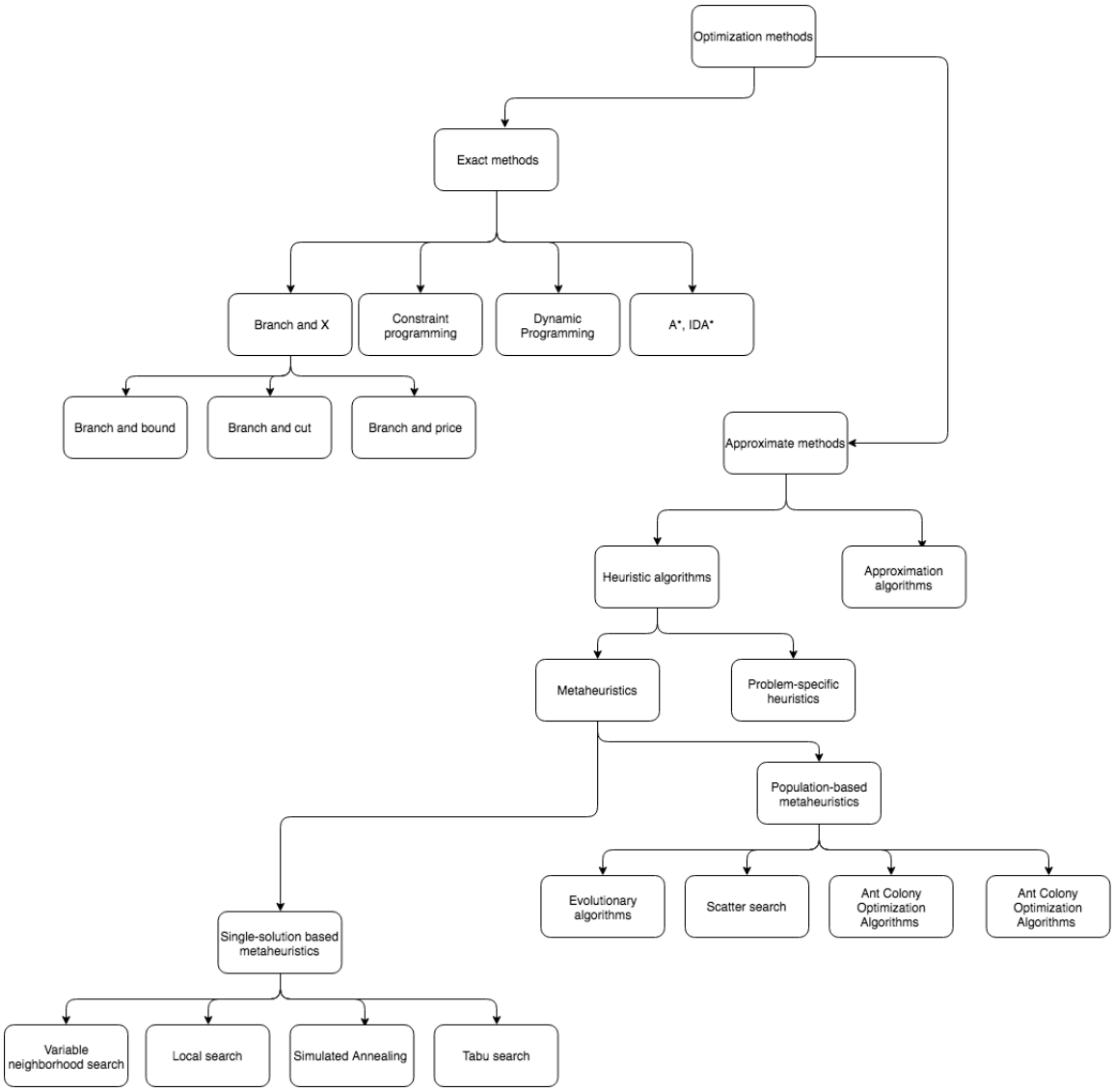


Figura 1.1: *Taxonomía de las principales técnicas de optimización existentes*

Optimización Combinatoria

La optimización combinatoria es conocida como el proceso de búsqueda de un máximo o un mínimo para una función objetivo F cuyo dominio es un espacio de soluciones discreto y de gran cardinalidad. Algunos ejemplo típicos de problemas de optimización combinatoria son:

- **El problema del viajante de comercio** : dado una serie de puntos (x,y) para un

conjunto de ciudades (N) se trata de encontrar el camino de menor recorrido que pase por cada una de las ciudades una única vez.

- **El problema de la mochila** : dado un conjunto (N) de distintos objetos de distinto tamaño S_i se trata de meterlos en distintas maletas o contenedores de distinta capacidad B_j
- **Job-Shop Scheduling** : dado un conjunto de tareas que deben ser realizadas y una serie de herramientas o recursos con los que se realizan estas tareas, se trata de encontrar la planificación que determina cuando y con que recurso se debe realizar cada una de las tareas de forma que esta, minimice el tiempo total de realización de todas las tareas.

Los problemas de optimización combinatoria son formulados mediante un modelo matemático que sigue la siguiente estructura:

1. **Conjunto de variables**

   Permiten modelizar y definir el problema. La asignación de valores a estas variables representan una solución.

2. **Conjunto de restricciones**

   Delimitan el espacio de soluciones factibles, es decir la asignación de valores a las variables del problema.

3. **Función Objetivo**

   Define la evaluación de una solución dada al problema. Usualmente se trata de minimizar esta función aunque también puede maximizarse.

La gran mayoría de técnicas de optimización para modelos matemáticos son parte de la programación lineal. El método tradicionalmente utilizado para resolver este tipo de problemas de programación lineal es el Método Simplex, el cual cuenta con varias variantes.

Sin embargo, no todos los problemas pueden ser resueltos mediante programación lineal y además puede darse el caso que el conjunto de soluciones sea muy grande para ser explorado con un algoritmo de fuerza bruta por lo que existen distintas técnicas de optimización que a diferencia de la programación lineal no garantizan encontrar la solución óptima pero si una solución de calidad en un periodo de tiempo asumible. Durante los siguientes apartados de esta memoria se hará un breve estudio de estas técnicas.

1.4.1.1 Métodos exactos

Son algoritmos que pretenden encontrar la solución óptima al problema dado. Una de las desventajas de estos métodos es que debido a su propia naturaleza pueden ser muy lentos dado que necesitan explorar todo el espacio de soluciones para garantizar que encuentran la solución óptima, de ahí que estos también sean conocidos como algoritmos de búsqueda exhaustiva. Los métodos de optimización exactos han sido aplicados a un gran número de problemas de optimización con éxito. Entre los distintos métodos cabe destacar los algoritmos voraces, la técnica divide y vencerás, los de ramificación y poda (branch and bound)...

Los métodos exactos resuelven de forma óptima y en un tiempo asumible todos aquellos problemas de la familia P. Sin embargo, existe otra familia de problema denominada NP para la cual no existen algoritmos exactos que sean capaces de resolver los problemas de esta familia de manera eficiente. Esto implica que aunque efectivamente exista un algoritmo exacto que sea capaz de encontrar una solución óptima a un problema NP, el tiempo necesario para encontrar esta solución es inviable. Por otra parte, los métodos exactos son completamente dependientes del problema que abordan de tal forma que si se quiere modificar el problema se debe de diseñar un nuevo algoritmo exacto.

Ramificación y poda (Branch and Bound)

La técnica de Branch and Bound, en español ramificación y poda, es una enumeración crítica del espacio de búsqueda. Además de enumerar, este método intenta descartar aquellas

soluciones del espacio de soluciones que no pueden contener la mejor solución.

Ramificación y corte (Branch and Cut)

Al igual que el algoritmo de ramificación y poda es un método de optimización combinacional utilizado para resolver problemas de programación lineal. El algoritmo de ramificación y corte es un mezcla entre el algoritmo de ramificación y poda con métodos de planos de corte.

Branch and price

Este método es un híbrido entre la técnica de ramificación y poda y el método de generación de Columnas[1]. Este método contempla la adición de columnas a la relación lineal en cada nodo del árbol de búsqueda. El algoritmo comienza excluyendo un conjunto de columna de la relajación con el propósito de disminuir la carga computacional y el uso de memoria, durante la ejecución del algoritmo se van añadiendo columnas a la relajación en caso de que sea necesario.

Programación con restricciones (Constraint Programming)

Aunque realmente no es un método exacto, se trata de un paradigma de la programación en el que la relación entre las variables definidas es expresada mediante ecuaciones que expresan una cierta restricción. Es utilizada como una tecnología software para resolver problemas combinatorios difíciles.

El conjunto de restricciones especificadas deben de ser cumplidas por una solución del problema. De esta forma, se puede asegurar que la solución obtenida es factible. A diferencia de otros métodos, en lugar de definir el método para obtener una solución del problema en este paradigma se describe las condiciones (restricciones) que debe de cumplir una solución para ser aceptada como tal.

Programación dinámica (Dynamic Programming)

Tal como pasa con la programación con restricciones, la programación dinámica no es un método exacto sino un paradigma de programación. La programación dinámica al igual que divide y vencerás es una metodología de programación que a menudo sirve para desarrollar algoritmos de tiempo polinomial. Esta metodología se basa en la solución obtenida en subproblemas ya resueltos anteriormente para así poder resolver de manera más eficiente problemas de mayor tamaño.

1.4.1.2 Métodos aproximados

Los métodos exactos exploran todo el espacio de soluciones para poder así obtener la solución óptima al problema. Por lo contrario, los métodos aproximados tratan de buscar una solución de calidad sin necesidad de explorar completamente el espacio de soluciones.

La principal ventaja es que cuando se pretende resolver un problema de gran tamaño el uso de métodos exactos puede hacer que obtener la solución óptima del problema sea muy costoso tanto temporal como computacionalmente. Es por eso, que normalmente para resolver problemas difíciles es común utilizar métodos aproximados ya que a pesar de que no se pueda garantizar que la solución obtenida es la óptima, se pueden adaptar en función de las restricciones temporales y computacionales. Estos métodos pueden ser clasificados en dos grandes grupos: heurísticos y metaheurísticos[6].

- **Heurísticas :** Las heurísticas clásicas comenzaron a desarrollarse entre 1960 y 1990 y, según establecen Toth y Vigo en The Vehicle Routing Problem (2002), se pueden clasificar en tres amplias categorías:
    - Métodos de Descomposición: a partir del problema inicial se desgranan varios subproblemas de menor dimensión y complejidad que permiten una vez resueltos todos conocer la solución del problema original.
    - Métodos Inductivos: pretenden encontrar soluciones a un problemas partiendo de premisas en casos más generales a casos más complejos.

- Métodos de Reducción: el objetivo de este tipo de métodos es encontrar soluciones a través de un espacio de soluciones para el problema relajado.
- Métodos Constructivos: iterativamente se pretende construir la solución al problema. Son por lo general métodos deterministas con políticas elitistas a la hora de escoger una opción en cada iteración.
- Métodos de Búsqueda Local: el procedimiento de búsqueda o mejora local parte de una o varias soluciones que se pretender mejorar progresivamente según avanza el procedimiento. Normalmente, el método finaliza cuando no existe ninguna solución accesible que mejore la anterior, aunque pueden determinarse diferentes condiciones de parada tanto como por coste temporal, calidad de la solución...

Los métodos constructivos y los de búsqueda local son la base de los métodos Metaheurísticos.

- **Metaheurísticas :** Son el conjunto de métodos aproximados que a través de procedimientos iterativos guían una heurística subordinada de búsqueda, combinando de forma inteligente distintos conceptos para explorar y explotar adecuadamente el espacio de búsqueda. Existen distintas metaheurísticas en función de conceptos como:
  - Seguimiento de trayectoria considerado
  - Uso de poblaciones de soluciones
  - Uso de memoria
  - Fuente de inspiración

Existe una gran cantidad de métodos aproximados con diferentes metodologías que se ajustan mejor en función de la naturaleza del problema que va a ser abordado. A continuación se detallan algunos de los métodos aproximados más utilizados.

Algoritmos de enjambre

Los métodos de optimización enmarcados en inteligencia de enjambres, son técnicas inspiradas en el comportamiento en la naturaleza de insectos, generalmente en colonias de hormigas, abejas, termitas...

Estos algoritmos tratan de emular el sistema colaborativo de estos insectos en la naturaleza para llevar a cabo tareas complejas que un solo individuo no podría realizar. En concreto, el caso de las hormigas, cada una de ellas se mueve en busca de un alimento dejando un rastro de feromonas que es seguido por el resto de individuos. Abstrayendo este concepto, el comportamiento colectivo de todos los agentes que conforman el sistema da como resultado una sola entidad capaz de realizar una tarea de mayor complejidad.

Los algoritmos de enjambre se basan en el comportamiento comunitario de las hormigas para realizar búsqueda de soluciones en problemas complejos.

Algoritmos evolutivos

En este tipo de algoritmos se establece una población inicial de individuos, entendiendo que cada uno de ellos representa una solución al problema. Estos individuos son evaluados mediante una función de aptitud que establece la calidad de cada individuo.

Posteriormente, en función de la aptitud de cada individuo se selecciona a un conjunto de estos que serán ahora los padres que producirán la siguiente generación de individuos por medio de un proceso evolutivo de cruce y mutación. Cabe destacar que en función de la aptitud de cada individuo la probabilidad de que este permanezca en la población es mayor. De esta forma se pretende ir mejorando iterativamente la calidad de las soluciones obtenidas.

El criterio de parada de estos algoritmos puede tener en consideración distintas condiciones. Lo más usual es utilizar restricciones temporales de ejecución pero se puede establecer cualquier otro criterio como uso de memoria, calidad de la solución...

Algoritmos de búsqueda local

Este tipo de algoritmo son métodos de mejora iterativa en los que por lo general se parte de una solución inicial. Utilizan una estructura de vecindad, soluciones similares a una solución dada, de tal forma que iterativamente se exploran las soluciones vecinas buscando aquella que mejore el valor objetivo de la mejor solución encontrada hasta el momento. Este proceso hace una exploración del espacio de soluciones en búsqueda de una solución de calidad.

Algoritmo de enfriamiento simulado

Esta técnica de optimización más conocida por su nombre en inglés Simulated Annealing, es un algoritmo de búsqueda meta-heurística para problemas de optimización global.

En cada iteración, el algoritmo de enfriamiento simulado evalúa alguna de las soluciones vecinas de la solución actual. En función de la evaluación de la solución vecina se decide si tomarla directamente como solución actual, en caso de que sea mejor, o hacerlo de forma probabilística, si la solución vecina es peor. Típicamente la comparación entre soluciones vecinas se repite hasta que se encuentre un estado óptimo que minimice la energía del sistema, hasta que se cumpla cierto tiempo computacional u otras condiciones de parada establecidas.

1.3.2. Aplicaciones existentes

Actualmente existen un sinfín de facilidades a la hora de resolver problemas de optimización a partir de una serie de herramientas que tomar como entrada un modelo matemático, que define de forma precisa el problema que queremos solucionar. A continuación se presenta una pequeña recopilación de algunos de los programas que existen en el mercado y que pueden ser utilizados por disponibilidad en la empresa TITSA.

Software propietario o privativo

- **CPLEX** : IBM ILOG CPLEX ofrece bibliotecas C, C++, Java, .NET y Pyhton que son capaces de resolver problemas de programación lineal y relacionados. Este software es capaz de resolver problemas de optimización entera, problemas complejos de programación lineal usando tanto la variante primitiva como dual del método simplex y del método del punto interior de barrera, problemas de programación cuadráticos convexos y no convexos y finalmente, problemas convexos cuadráticamente restringidos.

  Junto con Xpress, software que veremos a continuación, se tratan de aplicaciones específicas del ámbito de la optimización y el análisis de datos, lo que las convierte en dos de las aplicaciones más usadas en el ámbito industrial y empresarial. Cplex se ha adaptado tan bien al mercado debido a la facilidad con la que permite al usuario modificar los distintos parámetros de los algoritmos, así como la modificación de un gran número de opciones para nuestro problema. Además, cuenta con un apartado para el análisis final de nuestras soluciones, dando cierta información valiosa de cara a posibles modificaciones o futuros casos de uso.

- **XPRESS** : El optimizador Xpress, es desarrollado inicialmente por Dash Optimization, el cual más adelante sería adquirido (2008) por la compañía FICO. Desde el año 2002, previo a la correspondiente adquisición por FICO (Compañía de análisis de datos localizada en San Jose, California), dicho optimizador contaba con su propio lenguaje de modelado, Xpress Mosel.

  Principalmente, Xpress es un software diseñado para problemas tanto de programación no lineal como lineal, desde programación lineal entera mixta, programación cuadrática convexa, problemas convexos cuadráticamente restringidos, hasta programación de cono de segundo orden y sus homólogos mixtos enteros.

  Este tipo de software es desarrollado para la resolución de problemas reales gracias a

algoritmos robustos, su alto rendimiento y su gran versatilidad. Xpress Mosel en concreto, contiene una gran variedad de drivers para el acceso a texto, XML, R, Python, MATLAB, CSV, Excel o incluso, bases de datos de Oracle.

Software libre y de código abierto

- **Google OR Tools** : Son un conjunto de herramientas muy potentes disponibles para la resolución de problemas de optimización. Se trata de un paquete de software de código abierto que plantea rastrear los problemas más difíciles. Entres sus principales componentes están:

  - Un solucionador de programación de restricciones.
  - Un solucionador de programación lineal.
  - Distintos paquetes que utilizan solucionadores comerciales de gran potencia y solucionadores de código abierto, entre los que se incluyen algunos capaces de resolver problemas enteros-mixtos.
  - Distintas implementaciones para resolver problemas típicos de la disciplina de la optimización.
  - Algoritmos de optimización en grafos.

  Entre las distintas ventajas de esta herramienta caben a destacar las siguientes:

  - Código abierto y gratis: Todo, incluido los ejemplos, las implementaciones de algoritmos, la documentación, está bajo la licencia Apache 2.0 y está disponible para su descarga.
  - Continua mejora: las librerías están en continuo desarrollo aportándose mejorar casi a diario.
  - Documentado: Aunque se trata de un paquete muy novedoso ya existe una gran cantidad de documentación y ejemplos que permiten abordar problemas de una manera intuitiva.

- Portable: Al ser un paquete desarrollado totalmente por Google este se adapta a las directrices de desarrollo de código de Google.
- Eficiente: OR-Tools es una herramienta potente a las vez que eficiente, haciendo un uso inteligente de los recursos disponibles. Tanto es así que es utilizado internamente en la empresa Google.
- Accesible: A pesar de estar codificado en C++, todo esta disponible a través de SWIG en Python, Java and .NET (usando Mono en Sistemas sin Windows)

- **GLPK** : Software (Bajo el proyecto GNU) para la resolución de problemas lineales de gran escala, programación mixta entera y otros relacionados. Se trata de un conjunto de rutinas escritas en ANSI C y agrupadas en una librería. Los problemas pueden ser modelados en el lenguaje GNU MathProg, el cual comparte gran parte de su sintaxis con AMPL. Además, GLPK utiliza en segundo plano GLPSOL para la resolución de los problemas, mediante diversos métodos como el simplex revisado y el método de punto interior primitivo-dual para problemas no enteros y el método de ramificación y poda conjunto al método de corte entero mixto de Gomory para problemas enteros, por lo cual, en líneas generales, se considera inferior en términos de rendimiento comparado con otros programas privativos.

  GLPK puede ser usado en C como librería y en Java como extensión, gracias al sistema de modelado OptimJ. El proyecto FLPK no ofrece ninguna interfaz de usuario que resulte realmente amigable de cara al uso del cliente. Este último inconveniente, unido a otros muchos en términos de rendimiento, modelado y configuración, hacen que a día de hoy GLPK no pueda competir con otros softwares privativos como CPLEX o XPRESS.

- **CUTEr** : Entorno de testeo para optimización y álgebra lineal. CUTEr provee de una colección de problemas añadido a un conjunto de herramientas que ayudarán a los desarrolladores al diseño, la comparación y mejora de nuevos y existentes pruebas

de solucionadores de problemas. Da soporte para numerosas plataformas y sistemas operativos, pero desde el punto de vista de la facilidad de uso, surge el inconveniente de tener que traducir los ficheros de entrada de los problemas mediante un decoder del formato SIF a un conjunto de subrutinas bien definidas (A pesar de ser un proceso que no lleva mucho tiempo, siempre es un factor en contra).

Más de 1000 problemas están disponibles en la colección predefinida, incluyendo problemas de programación lineal, cuadráticos conexos y no convexos a pequeña y gran escala, problemas de mínimos cuadrados lineales y no lineales entre otros. Pero sin lugar a dudas, se trata de una aplicación que puede ser de gran ayuda para trabajar como complemento al resto de las mencionadas (no como suplemento).

Capítulo 2

Planteamiento General: Asignación de conductores a viajes.

La planificación de una jornada de trabajo para la empresa de transporte público Titsa se plantea de tal forma que a cada conductor se le asigna una serie de servicios determinados, entendiendo como servicio a cada viaje que realiza el autobús asociado a este[7]. El problema que se plantea es la reasignación de conductores, en el caso de que falte uno o varios de ellos, de tal forma que el número de pasajeros que se dejen sin recoger sea el menor posible. De esta forma, se plantea un problema de optimización de asignación de recursos con un único objetivo ya mencionado, minimizar el número de pasajeros sin recoger.

2.1. Datos utilizados

Para poder definir el problema de optimización propuesto es necesario conocer los datos que lo caracterizan. A continuación se hace un desglose de estos.

Líneas (L)

- Conjunto de servicios que realiza, se denota como S_{il} al servicio i de la línea l
- Horario, se puede conocer por el horario de inicio de su primer servicio y el horario de fin del último servicio.

Servicios (S)

- Línea a la que pertenece, S_{il} expresa que el servicio i es de la línea l.
- Conductor asignado al servicio, L_{ik} expresa que el servicio i está planificado que lo realice el conductor k. L es una variable de decisión que vale 1 en el caso de que la asignación del servicio i al conductor k esté planificada. En otro caso su valor será cero.
- t_f^i instante en el que finaliza el servicio i.
- t_i^i instante en el que comienza el servicio i.
- $t_{S_iS_j}$ tiempo de traslado entre el punto en el que finaliza el servicio S_i y el punto en el que comienza el servicio S_j.
- $t_{S_iD_c}$ tiempo de traslado entre el punto en el que finaliza el servicio S_i y el punto en el que se realiza el descanso c.
- Vehículo asignado al servicio. No se utiliza ninguna notación para referirse a esta asignación dado que no es necesaria controlarla en el problema.
- Se denota como SI al subconjunto de S que no pueden ser suspendidos. Se entiende que un servicio es suspendido cuando no se le asigna un conductor.
- Número de pasajeros estimados a los que recogerá el autobús durante el servicio. P_i es la predicción del número de pasajeros totales que habrá recogido el servicio i durante todo su recorrido.

Conductores (C)

Cada conductor disponible puede tener un horario distinto con el único factor común de que todos ellos deben de realizar un descanso de 25' entre la tercera y quinta hora de su jornada. Es por esto que se debe controlar el horario de descanso de cada conductor.

- in_k instante en el que comienza la jornada del conductor k.
- out_k instante en el que acaba la jornada del conductor k.

Conductores de baja ($F \subseteq C$)

- Identificador de cada uno de los conductores que ha faltado al trabajo. Con este identificador se puede conocer el resto de datos del conductor tales como los servicios que le habían sido asignados, su horario... No es necesario contar con ninguna notación que se refiera a estos conductores. Cuando se disponga a solucionar el problema simplemente se utilizará el conjunto C-F para determinar los conductores disponibles.

2.2. Características y restricciones

Es importante conocer las características intrínsecas del problema para poder abordar este de una manera correcta y obtener así soluciones reales. A continuación se enumeran las características del problema:

- Existen líneas en las que se puede dar que haya varias autobuses realizando un servicio distinto de manera simultánea.
- Dos viajes de la misma línea pueden tener distinto itinerario. Pueden tener distintos puntos de comienzo y/o final. Además su recorrido puede variar.
- Conductores distintos pueden realizar servicios de la misma línea. De la misma manera, un conductor puede realizar servicios de líneas distintas.
- Un conductor puede hacer servicios de varias líneas en una misma jornada.
- Los conductores deben acabar su jornada en el mismo punto en el que la iniciaron.
- El hecho de que varios servicios de la misma línea puedan estar realizándose simultáneamente obliga a que una línea pueda contar con más de un autobús para realizar sus servicios.

- Existen líneas con itinerario circular, esto significa que acaban su trayecto en el mismo punto en el que comenzaron.

Por otra parte existen una serie de restricciones :

- Se deben de respetar los horarios de descanso de todos los conductores.
- Los conductores no podrán realizar servicios que estén fuera de su horario.
- Los conductores no pueden realizar dos servicios al mismo tiempo
- De la misma forma un mismo servicio no puede ser asignado más de una vez a un conductor.
- Los servicios pertenecientes al conjunto SI deben de ser realizados por imposición.

2.3. Criterio de Optimización

Como se ha comentado previamente el objetivo del problema es encontrar aquella planificación que minimice el número de pasajeros sin recoger o lo que es lo mismo, que maximice el número de pasajeros recogidos. Por lo tanto la función objetivo del problema es la que se muestra a continuación aunque puede variar en función de la modelización del problema:

$$max \sum_{i}^{S} \sum_{k}^{C-F} L_{ik} * P_i$$

2.4. Solución al problema

La jornada está planificada de tal forma que se conoce para todos los servicios de la líneas el conductor que lo realizará. El problema aparece en cuanto está planificación no puede ser llevada a cabo porque existen un subconjunto de conductores, denotado como $F \subseteq C$, que no asistieron al trabajo ese día. Puede darse el caso de que alguno de estos

conductores fuese a realizar un servicio de muchos pasajeros por lo que es vital tratar de reorganizar la jornada de forma que se recoja al mayor número de pasajeros.

De esta forma, la solución al problema viene definida por la asignación de valores a una variable de decisión, en este caso, L_{ik} que puede tomar el valor 0 ó 1. La matriz binaria que define L debe respetar todas las restricciones establecidas en el problema, para considerarla factible, y maximizar la función objetiva para ser óptima.

De cualquier forma, la estructura de representación de esta solución será dependiente de la técnica de optimización utilizada y de las variables de decisión utilizadas.

Capítulo 3

Predicción de Pasajeros

Uno de los datos más importantes que se manejan en el modelo matemático es el número de pasajeros que se prevé que tendrá cada servicio. El hecho de tener certeza de que se maneja un dato muy similar a la realidad garantiza que la solución al problema que se obtenga será una de las mejores soluciones que se podrá aplicar posteriormente. Es por esto que gran parte del trabajo se centra en el desarrollo de un modelo "forecasting". Durante este capítulo se detallará el modelo usado y su construcción[3 11].

3.1. Datos utilizados

En un principio se partió con un fichero de texto plano en el que se recogían todas las transacciones de pasajeros que se habían producido entre enero de 2017 y marzo de 2019. Como se puede esperar, al tratarse de una empresa de transporte de gran tamaño como es TITSA, la tercera más grande de España, el número de transacciones realizadas superaba con creces los 20 millones. Es por esto último que el fichero que se manejaba era demasiado grande para poder abrirlo con cualquier editor, por lo que para poder hacer un tratamiento de los datos y agruparlos y depurarlos según la necesidad, se realizó un programa en c++ el cual se encargaba básicamente de contar el número de pasajeros por hora y día que había tenido cada línea de transporte. A continuación se muestra el formato de los datos sin tratar.

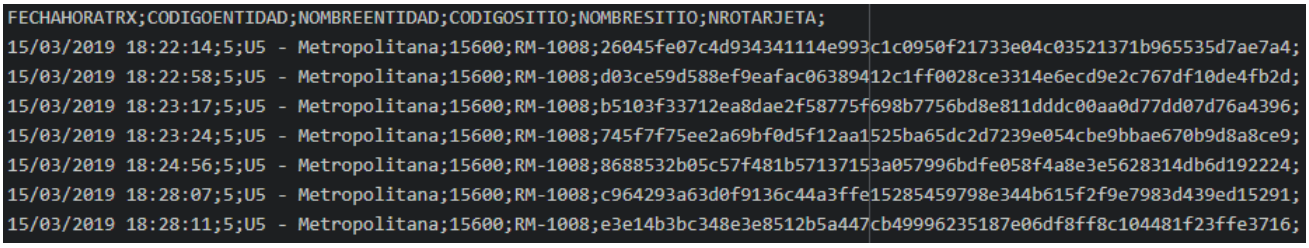


Figura 3.1: *Estructura de los datos de origen.*

Tal y como se muestra en la imagen, en la cabecera del fichero se describe cada uno de los datos recogidos para cada transacción. Dado que se quiere contar el número de pasajeros que se subieron al autobús, solo es de interés la fecha y el código de entidad que representa el identificador de la línea a la que se subió el pasajero.

En A se puede encontrar el código utilizado para obtener el conjunto de datos ya tratados. A continuación, se muestra una pequeña porción de los datos finales y una descripción de cada uno de los atributos utilizados.

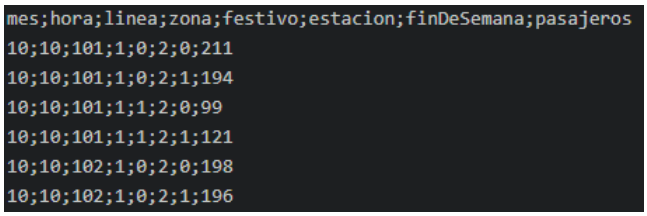


Figura 3.2: *Estructura de los datos ya tratados.*

- **Mes** : indica el mes en el que se realizó el servicio.
- **Hora**: Intervalo de una hora en la que se cuentan todas las transacciones. Este atributo puede tomar valores entre 7 y 23.
- **Linea**: Línea de transporte público.
- **Zona**: Zona geográfica a la que pertenece la línea. Puede ser cualquiera de las siguientes

- Interurbano LL
- Interurbano SC
- Largo Recorrido C <->N
- Largo Recorrido C <->S
- Largo Recorrido N <->S
- Urbano LL
- Urbano SC
- Zona Metropolitano
- Zona Norte
- Zona Sur

- **Festivo**: Determina si la entrada que se esta tratando era un día festivo. Pueden haber varios tipos:
  - Nacional
  - Regional
  - Insular
- **Estación**: Estación del año.
- **Fin de Semana**: Establece si el día era o no un fin de semana.
- **Pasajeros**: Finalmente se establece el número de pasajeros que se contó para esa entrada en concreto.

Llegados a este punto se dispuso a realizar un estudio estadísticos de los atributos seleccionados para conocer la correlación existente entre ellos[2].

	mes	hora	linea	zona	festivo	estacion	finDeSemana	pasajeros
mes	1	-0.0085	0.0067	0.0015	-0.07	0.19	0.02	0.0082
hora	-0.0085	1	0.011	0.0052	0.0076	0.0032	-0.0017	-0.067
linea	0.0067	0.011	1	0.82	0.017	-0.0021	-0.0027	0.0069
zona	0.0015	0.0052	0.82	1	0.029	0.00022	-0.0038	-0.027
festivo	-0.07	0.0076	0.017	0.029	1	0.0038	-0.2	-0.06
estacion	0.19	0.0032	-0.0021	0.00022	0.0038	1	0.03	0.0088
finDeSemana	0.02	-0.0017	-0.0027	-0.0038	-0.2	0.03	1	0.036
pasajeros	0.0082	-0.067	0.0069	-0.027	-0.06	0.0088	0.036	1

Figura 3.3: *Correlación de Pearson entre los atributos*

	mes	hora	linea	zona	festivo	estacion	finDeSemana	pasajeros
count	73187.000000	73187.000000	73187.000000	73187.000000	73187.000000	73187.000000	73187.000000	73187.000000
mean	6.484034	14.159782	390.740938	2.222854	0.340115	1.570006	0.438029	46.285843
std	3.649677	4.458898	303.871273	1.966611	0.629635	1.166091	0.496148	69.908446
min	1.000000	7.000000	11.000000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	3.000000	10.000000	122.000000	0.000000	0.000000	0.000000	0.000000	7.000000
50%	6.000000	14.000000	355.000000	2.000000	0.000000	2.000000	0.000000	20.000000
75%	10.000000	18.000000	480.000000	4.000000	1.000000	3.000000	1.000000	55.000000
max	12.000000	22.000000	972.000000	5.000000	3.000000	3.000000	1.000000	1184.000000

Figura 3.4: *Mediciones estadísticas de los datos*

Los resultados obtenidos mostraban que los datos no guardaban una gran correlación por lo que se indago un poco más en el motivo de esto. Tras una investigación en profundidad de los datos se descubrió que existían entradas que según la intuición eran erróneas. Un ejemplo de estos datos extraños es que por ejemplo para un día laboral la línea 15 a las 10 de la mañana se habían subido alrededor de 400 pasajeros y para otro día de las mismas condiciones a la misma hora el número de pasajeros recogido era de 13. La aparición de estas entradas en los datos estaba causada por el sistema INDRA dado que los vehículos transmitían la información mediante ficheros XML a las bases de datos y en ocasiones si

existía un fallo en el XML o surgía un corte en la conexión a internet este fichero no se transfería completamente por lo que se perdía cierta información.

Para solucionar este problema, se planteó realizar una media del número de pasajeros recogidos. Esto se hizo de tal forma que para todos los días con las mismas características se sumo el número de pasajeros de cada uno de ellos y se dividió entre el número de días contemplado. De esta forma, se redujo el número de líneas de los datos de aprendizaje pero se aumento considerablemente la correlación entre los atributos de ellos. Esto permite que el resultado final sea mucho más preciso ya que durante el proceso de aprendizaje se pueden detectar de manera mucho más fácil la relación existente entre los atributos. Más adelante se mostrarán los resultados obtenidos al usar estos datos tratados.

3.2. Perceptrón Multicapa

Llegados al punto en el que se dispone de los datos en el formato correcto y depurados se pasa a una fase de desarrollo de una herramienta predictiva. En el caso de este trabajo se optó por utilizar un perceptrón multicapa . A continuación, se describe su arquitectura así como el procedimiento de aprendizaje y algunos indicadores de calidad del regresor construido.

3.2.1. Arquitectura

El diseño de la estructura del perceptrón multicapa fue escogida desde el punto de vista del tiempo de aprendizaje. De esta forma, se hicieron varias pruebas con un número de capas fijo, 4 en concreto, a las que se le iba variando el número de neuronas de cada una de ellas. Estas pruebas con cada uno de los modelos fueron realizadas bajo las mismas condiciones con el afán de ser objetivos a la hora de encontrar el que mejores resultados aportase. En concreto, se estableció un máximo de 300 epochs para cada modelo. De esta forma la arquitectura utilizada para el modelo resultante es la que se muestra a continuación.

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 256)	60416
dense_2 (Dense)	(None, 512)	131584
dense_3 (Dense)	(None, 256)	131328
dense_4 (Dense)	(None, 1)	257

Total params: 323,585
Trainable params: 323,585

Figura 3.5: *Arquitectura del perceptrón multicapa.*

Tal y como se puede apreciar en la imagen anterior, el número de parámetros entrenables es considerablemente bajo lo cual acelera notablemente el proceso de entrenamiento. Un aspecto a destacar es el uso de la función de activación ReLu para las tres primeras capas del perceptrón. Esta función de activación se comporta tal y como se muestra en la siguiente imagen.

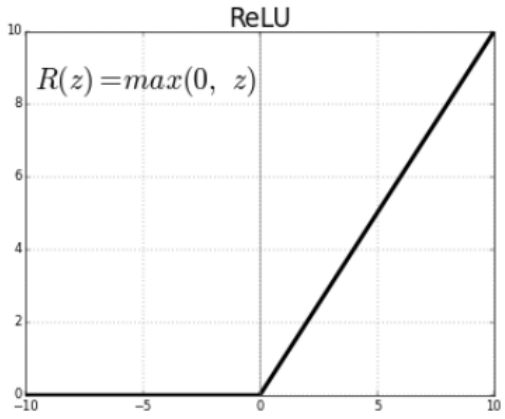


Figura 3.6: *Función de activación ReLu.*

3.2.2. Proceso de aprendizaje

El proceso de aprendizaje es la fase de desarrollo más importante, en este epígrafe se detallarán las distintas herramientas y técnicas que fueron utilizadas en el aprendizaje del

perceptrón como pueden ser optimizadores de learning rate, funciones de callback, representación de los datos, métricas a optimizar...

Representación de los datos

Debido a que la mayoría de los algoritmos de aprendizaje automático no son capaces de trabajar con variables categóricas se hace necesario hacer una transformación de estas variables en tipo numérico. Tal y como se puede apreciar en la imagen 3.2, los valores de los atributos categóricos ya han sido transformados en el proceso de tratamiento de datos a valores enteros. Sin embargo, pueden existir otras representaciones numéricas que aporten ciertos beneficios.

Para las variables categóricas que no presentan una relación ordinal, la representación en números enteros puede no ser suficiente. De hecho, utilizar esta representación y asumir que el modelo interprete un orden natural entre las categorías puede resultar en muchas ocasiones una mala práctica. En este caso, una representación one-hot puede ser aplicada a la representación entera. El valor entero que se utilizaba para representar una categoría es ahora eliminado y pasa a utilizarse una variable binaria para cada valor entero distinto, o lo que es lo mismo para cada categoría. Para entender esta representación se procede a presentar un breve ejemplo en que se utiliza representación one-hot para categorizar los colores.

Rojo	**Azul**	**Verde**
1	0	0
0	1	0
0	0	1

Observando la tabla anterior se puede saber que la dimensión que tendrá la variable binaria va a ser igual al número de categorías que tiene el atributo concreto. Además en función de la posición en la que aparezca el valor 1 en la variable binaria podremos saber a que categoría se refiere. En el caso del problema de predicción de usuarios se cuenta con un dataset con 7 variables y una etiqueta. El uso de la representación one-hot para las

entradas del dataset convierte cada entrada en un conjunto de siete variables binarias y un valor numérico. La dimensión total de la entrada tras ser transformada es de 236 valores. A continuación se muestra el proceso utilizado para hacer la transformación de datos.

```
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.preprocessing import LabelBinarizer

def load_attributes(inputPath):
    cols = ["mes","hora","linea","zona","festivo","estacion"
    ,"finDeSemana","pasajeros"]
    return pd.read_csv(inputPath, sep=";", header=None, names=cols)

def transformToBinary(df, train, test, col):
    # one-hot encode the categorical data (by definition of
    # one-hot encoing, all output features are now in the range [0, 1])
    binarizer = LabelBinarizer().fit(df[col])
    trainCategorical = binarizer.transform(train[col])
    testCategorical = binarizer.transform(test[col])

    return trainCategorical, testCategorical

def process_data_to_predict(df, train, test, predict=False):

    xDayCategorical, testxDayCategorical = transformToBinary(df, train,
   test, "mes")
    xMonthCategorical, testxMonthCategorical = transformToBinary(df, train
   , test, "hora")
    xHourCategorical, testxHourCategorical = transformToBinary(df, train,
   test, "linea")
    xLineCategorical, testxLineCategorical = transformToBinary(df, train,
   test, "zona")
    xTemperatureCategorical, testxTemperatureCategorical =
   transformToBinary(df, train, test, "festivo")
    xtDescriptionCategorical, testxtDescriptionCategorical =
   transformToBinary(df, train, test, "estacion")
    xHolidaysCategorical, testxHolidaysCategorical = transformToBinary(df,
    train, test, "finDeSemana")

    if not predict:
      return (np.hstack([xDayCategorical, xMonthCategorical,
                         xHourCategorical, xLineCategorical,
                         xTemperatureCategorical, xtDescriptionCategorical,
                         xHolidaysCategorical]),
              np.hstack([testxDayCategorical, testxMonthCategorical,
                         testxHourCategorical, testxLineCategorical,
                         testxTemperatureCategorical,
   testxtDescriptionCategorical,
```

```
                        testxHolidaysCategorical]))
    else:
      return np.hstack([xDayCategorical, xMonthCategorical,
   xHourCategorical, xLineCategorical,
                        xTemperatureCategorical, xtDescriptionCategorical,
                        xHolidaysCategorical])

df = load_attributes('datosTitsaRegresor/ficheroAgrupadoTitsa.csv')
(train, test) = train_test_split(df, test_size=0.25, random_state=42)
trainY = train["pasajeros"]
testY = test["pasajeros"]
(trainX, testX) = process_data_to_predict(df, train, test)
```

Listing 3.1: *Código Python One-Hot encoding*

El resultado final de este proceso es la división del conjunto de datos en dos dataframes, uno de entrenamiento y otro de test. La dimensión de estas estructuras de datos es la siguiente:

- Entrenamiento : **54890** filas con una dimensión de 235 valores binarios.
- Test : **18297** filas con una dimensión de 235 valores binarios.

Learning Rate y Callbacks

Las redes neuronales de aprendizaje profundo se entrenan utilizando el algoritmo de optimización de descenso de gradiente estocástico. El learning rate o tasa de aprendizaje es un hiper-parámetro que controla cuánto cambiar el modelo en respuesta al error estimado cada vez que se actualizan los pesos del modelo. Escoger un buen método de configuración del learning rate puede suponer en ocasiones una tarea confusa. Elegir un valor demasiado pequeño puede resultar en un largo proceso de entrenamiento que podría atascarse, mientras que un valor demasiado grande puede resultar en aprender un conjunto de pesos óptimo local demasiado rápido o un proceso de entrenamiento inestable.

La velocidad de aprendizaje puede ser el hiper-parámetro más importante al configurar una red neuronal. Por lo tanto, es vital saber cómo investigar los efectos de la tasa de aprendizaje en el rendimiento del modelo y construir una intuición sobre la dinámica del learning rate en el comportamiento del modelo. En el caso de este trabajo se han probado distintas funciones adaptativas del valor del learning rate obteniendo los mejores resultados con una

función de escalones que modifica el valor en función del epoch. Esta función se presenta a continuación.

```
from keras.callbacks import LearningRateScheduler

def lr_schedule(epoch):
  lrate = 0.001
  if epoch > 75:
    lrate = 0.0005
  elif epoch > 100:
    lrate = 0.0003
  return lrate
```

Listing 3.2: *Función de adaptación del LearningRate*

Por otra parte el uso de funciones callbacks permite tener un control y monitoreo del proceso de aprendizaje del modelo. Existen una serie de funciones ya definidas que han sido utilizadas en este proyecto. En primer lugar, la función *ModelCheckPoint*, esta se encarga de monitorear una determinada métrica para guardar los pesos del modelo en el caso de que durante el entrenamiento se encuentren algunos pesos que la mejoren. Por otra parte se ha utilizado la función *EarlyStopping* la cual para el entrenamiento si un número indicado de epochs no se ha conseguido ninguna mejora. Una de las ventajas de esta función es el ahorro de recursos que supone. En la siguiente porción de código se pueden ver las distintas funciones utilizadas.

```
# checkpoint
filepath="weights-improvement-{epoch:02d}-{mean_squared_error:.3f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='mean_squared_error',
    verbose=1, save_best_only=True, mode='min')
callbacks_list = [checkpoint, LearningRateScheduler(lr_schedule)]

keras_callbacks = [
      ModelCheckpoint('/tmp/keras_checkpoints/model.{epoch:02d}-{val_loss
    :.2f}.hdf5', monitor='val_loss', save_best_only=True, verbose=2)
     # ModelCheckpoint('/tmp/keras_checkpoints/model.{epoch:02d}.hdf5',
    monitor='val_loss', save_best_only=True, verbose=0)
      TensorBoard(log_dir='/tmp/keras_logs/model_3', histogram_freq=0,
    write_graph=True, write_images=True, embeddings_freq=0,
    embeddings_layer_names=None, embeddings_metadata=None),
     EarlyStopping(monitor='mean_squared_error', patience=20, verbose=0)
```

```
]
```

Listing 3.3: *Definición de funciones de Callback*

Finalmente, una vez ya está el entorno de entrenamiento configurado, se procede a lanzar el proceso que comienza el entrenamiento. En este caso se ha utilizado un tamaño del batch de 256 entradas. El código utilizado es presentado a continuación.

```
history = model.fit(trainX, trainY, epochs=500, batch_size=256, verbose=1,
    callbacks = callbacks_list)
```

Listing 3.4: *Entranamiento*

3.3. Resultados obtenidos

Llegados a este punto, tras el entrenamiento, se dispone de un conjunto de pesos con las siguientes características:

- **Error Cuadrático Medio : 0.00070**
- **Error Absoluto Medio: 0.0191**

Una vez se dispone del perceptrón entrenado se pasa a una fase de testeo en la que se comprueba la precisión de las predicciones realizadas. Para conocer mejor el funcionamiento del perceptrón se utilizó un 25 % del conjunto de datos que habían sido reservado para test y con el cual no fue entrenado el MLP. Utilizando estos datos se desarrollo el siguiente código, que permite ver a simple vista una comparación entre el valor obtenido por la herramienta y el valor real de la entrada. De esta forma, se puede apreciar rápidamente la calidad de los resultados obtenidos.

```
for x, y in zip(testX, testY):
  features = np.reshape(x, (1, testX.shape[1]))
  ynew=model.predict(features)
  value = ynew[0]
  print("Value=%s, Predicted=%s" % (y, round(value[0], 0)))
```

Listing 3.5: *Predicciones frente a valor real*

La salida producida al ejecutar esta porción de código es la siguiente:

```
Value=66, Predicted=69.0
Value=7, Predicted=8.0
Value=2, Predicted=4.0
Value=20, Predicted=14.0
Value=19, Predicted=18.0
Value=2, Predicted=0.0
Value=25, Predicted=23.0
Value=43, Predicted=40.0
Value=27, Predicted=30.0
Value=29, Predicted=36.0
Value=11, Predicted=8.0
Value=196, Predicted=193.0
Value=11, Predicted=8.0
Value=11, Predicted=13.0
Value=4, Predicted=15.0
Value=164, Predicted=158.0
Value=13, Predicted=10.0
Value=218, Predicted=198.0
Value=57, Predicted=59.0
Value=2, Predicted=1.0
Value=74, Predicted=62.0
Value=20, Predicted=19.0
Value=46, Predicted=48.0
Value=8, Predicted=8.0
Value=2, Predicted=3.0
Value=25, Predicted=25.0
Value=147, Predicted=141.0
Value=10, Predicted=18.0
Value=83, Predicted=78.0
Value=2, Predicted=4.0
Value=4, Predicted=9.0
Value=35, Predicted=35.0
Value=188, Predicted=201.0
```

Figura 3.7: *Resultado de ejecutar el código de prueba de resultados.*

Observando la imagen anterior se puede apreciar como a pesar de que las predicciones en mayor medida no son iguales al dato real, su valor es muy cercano. Esto sucede, principalmente por la métrica que se ha escogido minimizar, ya que al existir una gran variación del número de pasajeros para las distintas entradas, es necesario utilizar el error cuadrático medio. Por otra parte, los resultados mostrados dan a entender que las predicciones son realmente buenas en líneas generales. Dado que se pretende que las predicciones obtenidas sean iguales o muy similares al valor real, se establece un margen que hace referencia a la cantidad en número de pasajeros que puede desviarse la predicción del valor real. Como es de esperar cuanto mayor sea el margen mayor será la precisión de la herramienta. A conti-

nuación se presenta en una tabla la variación de la precisión de la herramienta en función del margen. Para realizar este estudio el código utilizado es el siguiente:

Margen	0	1	5	10	15	20	25
Precisión	9.81 %	28.16 %	70.54 %	87.68 %	93.79 %	96.76 %	98.01 %

```
'''PRUEBA DE PREDICCIONES SOBRE EL CONJUNTO DE TEST'''
error = 0
#margen refiere el número de pasajeros arriba/abajo con el que se
    considera que la predicción es acertada
margen = 25

for x, y in zip(testX, testY):
  features = np.reshape(x, (1, testX.shape[1]))
  ynew=model.predict(features)
  value = ynew[0]
  if (y < (round(value[0] - (margen)))) or (y > (round(value[0] + (margen)
   ))):
    error += 1

accuracy = (1 - (error / len(testY))) * 100
print('{}%'.format(accuracy))
```

Listing 3.6: *Predicciones frente a valor real*

Por último, para tener otro claro indicador de la calidad del regresor creado se ha calculado el coeficiente R^2. El coeficiente de determinación, se define como la proporción de la varianza total de la variable explicada por la regresión. El coeficiente de determinación, también llamado R cuadrado, refleja la bondad del ajuste de un modelo a la variable que pretender explicar.

$$R^2 = \frac{\sum_{t=1}^{T}(\hat{Y}_t - \bar{Y})^2}{\sum_{t=1}^{T}(Y_t - \bar{Y})^2}$$

La mejor puntuación posible es 1.0 y puede ser negativa (porque el modelo puede ser arbitrariamente peor). Un modelo constante que siempre predice el valor esperado de y, sin tener en cuenta las características de entrada, obtendría una puntuación R^2 de 0.0. Utilizando la función r2_score de la librería sklearn se obtiene que el valor del coeficiente es de $R^2 = \mathbf{0.98228.}$

```
y_true = []
y_pred = []

for x, y in zip(testX, testY):
  features = np.reshape(x, (1, testX.shape[1]))
  ynew = model.predict(features)
  value = ynew[0]
  y_true.append(y)
  y_pred.append(value[0])

print(r2_score(y_true, y_pred))
```

Listing 3.7: *Cálculo del coeficiente* R^2

Capítulo 4

Técnicas aplicables al problema y decisión.

Una vez se conoce el problema detalladamente se pasa a escoger la técnica o herramienta con la que se pretende resolver. En este capítulo se presentarán algunas de las técnicas que han sido planteadas para resolver el problema haciendo un breve análisis de las ventajas y desventajas del uso de cada una de ellas. Finalmente, se presenta la opción escogida y se justifica su elección. Existen una serie de factores externos al problema que influyen en la toma de la decisión de que herramienta utilizar para resolverlo. Entre ellos cabe destacar, en que momento surge el problema, la calidad que debe tener la solución, el tamaño medio del problema, el tiempo de ejecución disponible...

4.1. Posibles técnicas de resolución.

Antes de presentar las distintas técnicas que fueron barajadas es importante conocer el entorno en el que surge el problema para así poder evaluar las ventajas y desventajas de cada una de estas[4][5]. Uno de los factores a destacar es el hecho de que el problema surge con cierta antelación a la propia jornada que se procede a replanificar. Esto es debido en mayor medida a que cuando un conductor falta a trabajo suele avisar con al menos un día de antelación. Este hecho permite invertir mucho tiempo en encontrar una solución óptima por lo que en un principio parece que lo más adecuado es el uso de un método exacto. Por otra

parte, la solución obtenida debe de ser la óptima ya que en todo momento se pretende dar el mejor servicio posible. A continuación se procede al análisis de ventajas e inconvenientes de cada técnica.

4.1.1. Algoritmos Evolutivos

Tal y como se comentó anteriormente en esta memoria, este tipo de algoritmos utiliza un población inicial de individuos, en la que cada uno de los individuos que la conformas representa una solución potencial al problema. Mediante una función de aptitud y una serie de operadores de cruce, mutación y selección se pretende mejorar la aptitud de las soluciones pertenecientes a la población. La principal ventaja de aplicar esta técnica para resolver el problema es que permite encontrar soluciones a problemas de gran tamaño en menor tiempo que los métodos exactos. Sin embargo, la solución obtenida no tiene por que ser la óptima y en ocasiones tampoco es factible por lo que en un principio se descartó su uso.

4.1.2. Ramificación y Poda

Los algoritmos de ramificación y poda se pueden entender como un árbol de decisión en los que tras realizar una determinada decisión hay un conjunto de posibles decisiones. De esta forma, la solución se conforma como un conjunto de decisiones ya tomadas que consiguen un cierto objetivo. El mayor problema de este algoritmo es la escalabilidad dado que cuando se trata con problemas de gran tamaño el conjunto de posible decisiones que se puede tomar en un determinado nodo puede ser inmenso, sobretodo en los nodos de mayor nivel del árbol. Es por eso que se hace necesario una buena estrategia de poda que haga una exploración eficiente del espacio de soluciones. A diferencia de los algoritmos evolutivos este método si es capaz de encontrar la solución óptima.

La razón por la que esta técnica es descartada es la dificultad posterior de añadir nuevas restricciones al problema o modificar las ya creadas. Es por eso que se plantea utilizar un software comercial específico para el modelado de problemas de optimización.

4.1.3. CPLEX

El uso de este software comercial para resolver el problema proporciona una gran cantidad de beneficios dada la naturaleza de la problemática presente. CPLEX permite el uso del lenguaje OPL, un lenguaje de programación con restricciones. Este modela:

- Programación lineal, entera y mixta
- Programación cuadrática y problemas convexos cuadráticamente limitados.
- Programación con restricciones.
- Modelado de planificación (Scheduling)
- Conexión con bases de datos relacionales.

A modo general, el esquema del algoritmo resolutor de OPL es el siguiente:

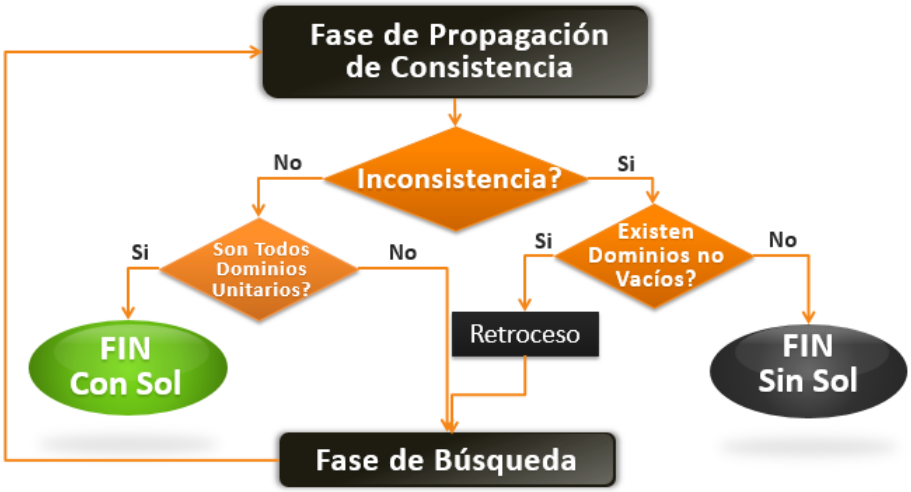


Figura 4.1: *Esquema algoritmo resolutor CPLEX.*

El modo en el que interactúan los componentes de CPLEX se puede ver representado esquemáticamente en la siguiente imagen.

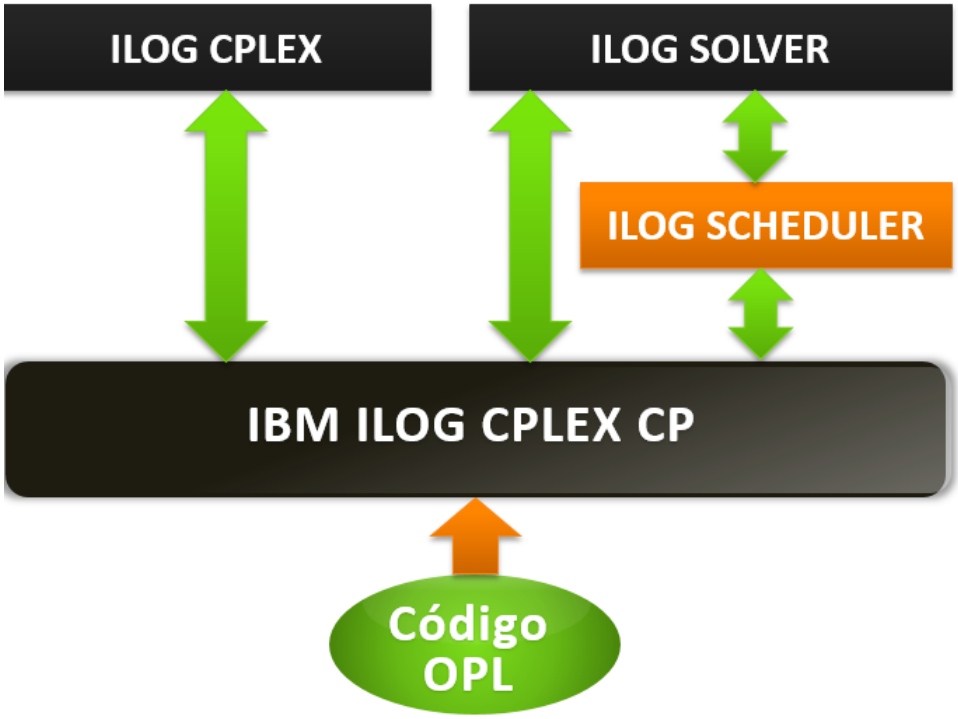


Figura 4.2: *Esquema de componentes CPLEX.*

El principal objetivo del uso de OPL es que el programador sea capaz de modelar los problemas de optimización que se encuentran en entornos industriales. Las principales razones por las que se decidió utilizar esta herramienta para el trabajo son las siguientes:

- El lenguaje OPL es un lenguaje fácil de aprender a la vez que potente y bien documentado.
- Su representación es muy cercana a la formulación matemática del problema a resolver. Esto simplifica mucho el proceso de implementación de un modelo matemático.
- Se reduce el tiempo de desarrollo sin sacrificar eficiencia.
- Compatibilidad de las bibliotecas de componentes.
- Agilidad: CPLEX es compatible con una gran variedad de sistemas informáticos.

- Permite obtener la solución óptima de forma mucho más eficaz que cualquier algoritmo debido a la optimización de los procedimientos de sus bibliotecas y las técnicas que usan.

Capítulo 5

Modelo Matemático e implementación en CPLEX

En este capítulo se describe el modelo matemático que se ha planteado para resolver el problema planteado en el capítulo 2. Para representar el problema se ha utilizado una estructura de datos en forma de grafo. A continuación se detalla el método de construcción del grafo así como los distintos tipos de nodo que existen en él. Tras esto se pasa a detallar la notación utilizada para referirse a los datos. Y por último la modelización de la restricciones así como la función objetivo[8 10 12].

5.1. Notación de los datos

En esta sección se presenta, en forma de tabla, la notación utilizada para referirse a cada uno de los datos utilizados para formular el problema.

Notación	Dato
t_f^i	instante en el que finaliza el servicio i.
t_i^i	instante en el que comienza el servicio i.
$t_{S_iS_j}$	tiempo de traslado entre el punto en el que finaliza el servicio S_i y el punto en el que comienza el servicio S_j .
$t_{S_iD_c}$	tiempo de traslado entre el punto en el que finaliza el servicio S_i y el punto en el que se realiza el descanso c.
in_k	instante en el que comienza la jornada del conductor k
ind_c	instante en el que comienza el descanso c
out_k	instante en el que acaba la jornada del conductor k.
S	conjunto de servicios
O^k	nodo origen del conductor k
F^k	nodo final del conductor k
D_c	nodo del descanso c
n	número de servicios
m	número de conductores
r	número de posibles horarios de descanso
DC_k	conjunto de nodos de descanso que pueden ser usados por el conductor k
P_i	número de pasajeros esperados para el servicio i
X_{ij}^k	1 si el conductor k realiza el servicio j después del i , 0 en otro caso
SI	subconjunto de servicios de S que no pueden ser suspendidos.

5.2. Nodos

Existen varios tipos de nodos que conforman el grafo del problema. La clasificación de estos según su tipología es la siguiente:

- **Nodos origen** : Hacen referencia al punto de origen de un conductor cualquiera. Existen tantos nodos origen como conductores disponibles.
- **Nodos servicio** : Se refieren a cada uno de los servicios que se pretenden planificar.
- **Nodos fin** : Representan el final de la jornada de conductor. El número de nodos fin es igual al número de nodos origen.

- **Nodos Consumo** : Representan los distintos horarios de descanso que puede tomar un conductor.

5.3. Aristas

La existencia de aristas en el grafo esta condicionada a las restricciones temporales establecidas por los horarios de los conductores, la hora de salida de los servicios y la duración de estos.

De esta forma, en el caso de los servicios, para que exista una arco que conecte el nodo i con el nodo j debe de cumplirse que la hora de fin del servicio i más el tiempo de traslado desde el punto de fin del servicio i a el punto de comienzo del servicio j sea menor que la hora de comienzo del servicio j. La formalización de esta condición es la siguiente:

$$\exists(S_i, S_j) \leftrightarrow t_f^i + t_{S_i S_j} \leq t_i^j$$

Para los nodos orígenes, existirá un arco que lo conecte con un servicio i si el instante en el que comienza el servicio es mayor o igual que el instante en el que comienza la jornada el conductor k asociado al nodo origen.

$$\exists(O^k, S_i) \leftrightarrow t_i^i \leq in_k$$

Para el caso de los nodos fin, se determina un arco que conecta un servicio i con un nodo de fin del conductor k en el caso de que el instante de finalización del servicio i sea menor que la hora de salida del conductor k.

$$\exists(S_i, F^k) \leftrightarrow t_f^i \leq out_k$$

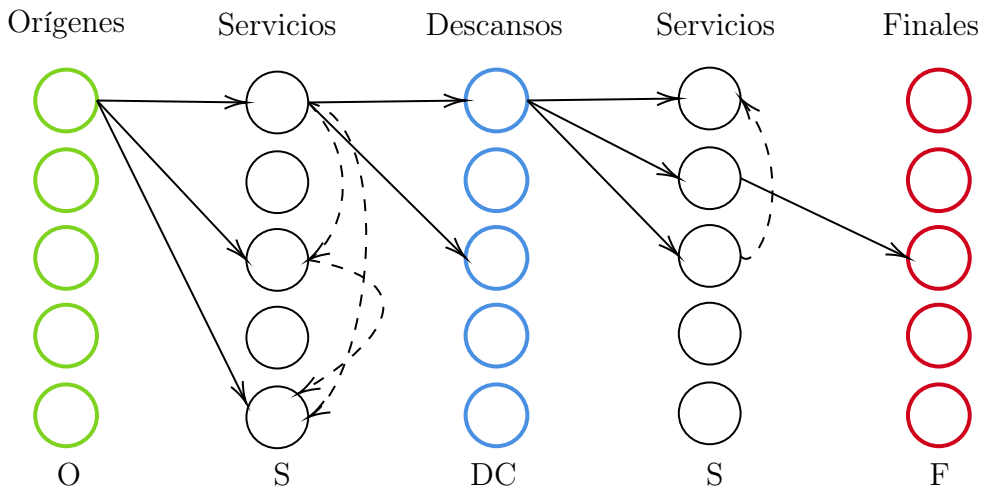
Los nodos consumo solo pueden ser conectados mediante un arco con servicios. De esta forma, para que existe un arco desde un servicio i a un nodo consumo c. El instante en el que finaliza el servicio más el tiempo de traslado a el punto de descanso debe de ser menor o igual al tiempo de comienzo del descanso estipulado por el nodo de consumo c.

$$\exists(S_i, D_c) \leftrightarrow t_f^i + t_{S_i D_c} \leq ind_c$$

De la misma forma, para que exista un arco entre un nodo de descanso c y un servicio i, el instante en el que comienza el descanso más la duración del mismo (25 min) más el tiempo de traslado al punto de comienzo del servicio i debe de ser menor que la hora de comienzo del mismo.

$$\exists(D_c, S_i) \leftrightarrow ind_c + 25 + t_{D_c S_i} \leq t_i^i$$

A continuación se presenta un esbozo de lo que es la estructura del grafo final. De esta forma, se puede comprender como ciertas restricciones del problema están implícitas en la construcción del grafo. La metodología utilizada para construir el grafo imposibilita la aparición de ciclos en este. Este hecho se verá reflejado posteriormente en el modelo, ya que no será necesario añadir las restricciones de eliminación de ciclos o subtours.



5.4. Función Objetivo

La función objetivo trata de maximizar el número de pasajeros recogidos o lo que es lo mismo, minimizar el número de pasajeros sin recoger.

$$max\left(\sum_{k=1}^{m}\sum_{i=1}^{r} P_i * \sum_{j\in\delta_i^+} X_{ij}^k\right)$$

5.5. Restricciones

Dado que la propia topología del grafo usado para representar el problema elimina los ciclos, no es necesario contemplar esta imposición como una restricción del modelo. Esto se traduce posteriormente en un tiempo de computo menor.

- Todos los nodos que representen un servicio insuspendible deben de ser realizados.

$$\sum_{k=1}^{m} \sum_{j \in \delta_i^+} X_{ij}^k = 1 \ \forall i \in SI$$

- Todos los conductores deben de salir de su nodo origen asignado.

$$\sum_{j \in \delta_i^+}^{i \in O_k} X_{ij}^k = 1 \ \forall k \in 1...m$$

- Todos los conductores deben de acabar en su nodo fin asignado.

$$\sum_{i \in \delta_j^-}^{j \in F_k} X_{ij}^k = 1 \ \forall k \in 1...m$$

- Cada conductor debe de realizar uno de los descansos posibles para el.

$$\sum_{i \in DC_k} \sum_{j \in \delta_i^+} X_{ij}^k = 1 \ \forall k \in 1...m$$

- El número de arcos que inciden en un nodo menos los que salen del mismo debe de estar condicionado en función de la tipología del nodo.

$$\sum_{k=1}^{m} \left(\sum_{j \in \delta_i^+} X_{ij}^k - \sum_{j \in \delta_i^-} X_{ji}^k \right) = 1 \ \forall i \in O$$

$$\sum_{k=1}^{m} \left(\sum_{j \in \delta_i^+} X_{ij}^k - \sum_{j \in \delta_i^-} X_{ji}^k \right) = -1 \ \forall i \in F$$

- El arco que entra en un nodo y sale del mismo debe de estar asociado al mismo conductor.

$$\sum_{j \in \delta_i^+} X_{ij}^k - \sum_{j \in \delta_i^-} X_{ji}^k = 0 \ \forall i \in V - \{O^k, F^k\} \wedge \forall k \in 1..m$$

- Un servicio puede ser realizado una sola vez o en caso de que no se le asigne un conductor no realizarse.

$$\sum_{k=1}^{m} \sum_{j \in \delta_i^-} X_{ji}^k \leq 1 \; \forall i \in V - \{O, F\}$$

- El valor de la variable de decisión es binario. De esta forma, se restringen las soluciones fraccionarias.

$$X_{ij}^k \in \{0,\ 1\}$$

5.6. Implementación en CPLEX

Una vez definido el modelo matemático para el problema se pasa a definir este modelo en CPLEX[9] utilizando el lenguaje de modelado OPL. En primer lugar, debido a que los datos utilizados para definir el problema son extraído de las bases de datos de la empresa TITSA como ficheros de tipo CSV, se hace necesario realizar un tratamiento de estos datos para darle el formato concreto que permita cargarlos en CPLEX. Es por esto que a continuación se presenta el origen de los datos el tratamiento de estos así como el resultado final. Tras esto se pasa a mostrar la especificación de cada una de las restricciones en CPLEX.

5.6.1. Tratamiento de los datos

El origen de los datos que se utilizan es como ya se mencionó las bases de datos de la empresa TITSA. Existen distintas vías de obtener los datos y no todas garantizan que el dato obtenido sea fiable al cien por cien. El hecho de contar con datos no reales puede ocasionar que posteriormente el problema que se quiere resolver no tenga solución o la solución obtenida no sea real. Es por eso que es de vital importancia contar con datos fiables.

Formato de datos en el origen

Tal y como se comenta anteriormente existen distintas vías de donde extraer los datos. En un principio se decidió extraer la planificación de las bases de datos en la que se recogía

los viajes que habían sido realizados un determinado día. El problema de utilizar estos datos es que había ocasiones en las que los vehículos perdían la conexión a internet y por lo tanto no aparecían reflejados sus viajes en las bases de datos. Es por esto, que se decidió descartar esta fuente y se paso a buscar otra forma de extraer los datos. Finalmente, los datos fueron obtenidos de la aplicación GPR, una herramienta desarrollada en TITSA que se encarga de realizar una planificación en función de los recursos disponibles. Observando la salida de esta planificación se creo un fichero CSV con la siguiente estructura.

```
COD_TURNO;COD_LINEA;ORIGEN;DESTINO;INICIO;FIN
901011;901;9200;9450;6:00;6:18
901011;906;9181;9453;6:35;7:05
901011;901;9200;9450;7:15;7:33
901011;901;9181;9200;7:45;8:09
901011;901;9200;9450;8:20;8:38
901011;901;9181;9200;9:25;9:49
901011;901;9200;9450;10:00;10:18
901011;901;9181;9200;10:55;11:19
```

Figura 5.1: *Estructura fichero CSV con datos de origen.*

El valor de la primera columna hace referencia al conductor asociado a ese viaje. De esta forma se puede conocer cada uno de los viajes que ha sido planificado para un determinado conductor. Además se recogen otros datos referidos al viaje como son la línea a la que pertenece, el código de la parada de origen, el código de la parada de destino y la hora de inicio y fin del viaje. Con esto lo único que falta es conocer el número de pasajeros que tendrá cada servicio. Este dato puede ser obtenido por dos vías, en primer lugar se puede utilizar el historial de transacciones para conocer el número de pasajeros que se subió en el vehículo que realizó el viaje, entendiendo el problema como un escenario que ya se dio en el pasado y se repetirá en el futuro. Por otro lado, se puede utilizar el perceptrón multicapa desarrollado para estimar el número de pasajeros por hora que tendrá cada línea. De esta forma bastaría con distribuir proporcionalmente los pasajeros por hora y linea entre los viajes de una misma línea en una hora. El resultado final al añadir el dato de los pasajeros

por viaje sería un fichero CSV con la misma estructura que el anterior pero con una nueva columna final en la que se recoja este dato.

Formato de los datos ya tratados

Para poder utilizar los datos en CPLEX, estos deben de seguir una estructura igual a la mostrada en la siguiente imagen de ejemplo.

```
Aristas = {
< 0, 1>,< 0, 2>,< 0, 3>,
...
};
DescansosConductor = {
<0, 83>
<1, 89>
...
};
FinalesConductores = #[
0 : 79
1 : 80
...
]#;
OrigenesConductores = #[
0 : 75
1 : 76
...
]#;
n = 75;
m = 4;
r = 24;
ServiciosObligados = {35};
NodosOrigenes = {75 76 ... };
NodosFinales = {79 80 ...};
NodosDescanso = {83 84 ... };
NodosServicios = { 0 1 ...};
pred = {<0> <1> <2> ...};
P_i = #[
<0> : 38
<1> : 46
<2> : 38
... ]#;
```

Figura 5.2: *Estructura del fichero con los datos para CPLEX.*

En el fichero se recogen todos los datos que construyen el grafo que modela el problema de optimización. En primer lugar se establecen las distintas aristas que conectan los nodos

del problema. Tras esto se establece que nodos pueden ser utilizados para un determinado conductor para descansar. A continuación se establece el nodo de origen y de fin de cada conductor y se le asigna un valor a las variables n, m y r. Por último se especifica la tipología de cada uno de los nodos y la estimación del número de pasajeros asociada a cada uno de ellos. Como es de esperar, solo aquellos nodos que sean de servicio tendrán una estimación de pasajeros igual o superior a cero, teniendo el resto un valor igual a cero.

Para poder realizar esta transformación de formato de los datos se ha desarrollado un programa en Python[2] el cual utiliza el fichero de origen para generar un fichero final en formato .dat el cual contiene ya toda la información en el formato correcto. Este programa se puede encontrar como apéndice de esta memoria B.

5.6.2. Recogida de los datos y definición de restricciones en CPLEX

Una vez se dispone del fichero con los datos en el formato correcto solo falta definir un fichero .mod en cplex donde se recojan estos datos y se definan las restricciones que debe de cumplir la solución al problema. Para recoger los datos desde CPLEX simplemente es necesario enlazar el fichero al proyecto y utilizar este código en el fichero del modelo.

```
{int} NodosServicios = ...;
int n = ...;
{int} NodosOrigenes = ...;
{int} NodosFinales = ...;
int m = ...;
{int} NodosDescanso = ...;
int r = ...;
int totalNodos = n + 2*m + r;
{int} ServiciosObligados = ...;

{int} Nodos = NodosServicios union NodosOrigenes union NodosFinales union
    NodosDescanso;

tuple arista{
    int origen;
    int destino;
}
{arista} Aristas = ...;

tuple descCond{
    int conductor;
    int nodoDescanso;
```

```
}
{descCond} DescansosConductor = ...;

tuple pasajeros {
    int nodoServicio;
}

{pasajeros} pred = ...;
int P_i[pred] = ...;
int OrigenesConductores[0..(m-1)] = ...;
int FinalesConductores[0..(m-1)] = ...;
```

Listing 5.1: *Definición y recogida de datos en Cplex*

5.6.3. Variables de decisión y función objetivo

Llegados a este punto, una vez ejecutado el programa los datos ya están cargados en memoria por lo que se debe definir en primer lugar las variables de decisión y luego la función objetivo.

```
# SerCon[i][j][k] 1 si el conductor k va al nodo j despúes del i, 0 en
    otro caso
dvar int+ SerCon[0..(totalNodos-1)][0..(totalNodos-1)][0..(m-1)] in 0..1;

maximize
    sum(k in 0..(m-1))sum(i in Nodos)sum(<i,j> in Aristas) P_i[<i>] *
    SerCon[i][j][k];
```

Listing 5.2: *Definición de variables de decisión y función objetivo*

Por último es establecen las restricciones del modelo.

```
subject to {
    #Cada origen debe de ser asignado a su conductor
  forall(k in 0..(m-1))
    sum(<i,j> in Aristas : i == OrigenesConductores[k]) SerCon[i,j,k] ==
    1;

  #Cada final debe de ser asignado a su conductor
  forall(k in 0..(m-1))
    sum(<i,j> in Aristas : j == FinalesConductores[k]) SerCon[i,j,k] == 1;



  #Cada conductor solo puede realizar un único descanso de los que tiene
    como posible
  forall(k in 0..(m-1))
      sum(<k,i> in DescansosConductor)sum(<i,j> in Aristas) SerCon[i,j,k]
    == 1;
```

```
15
16
17      #Los servicios pueden ser suspendidos o realizados una única vez
18      forall (i in NodosServicios)
19          sum(k in 0..(m-1))sum(<j,i> in Aristas ) SerCon[j][i][k] <= 1;
20
21      #En los nodos origenes no puede entrar ninguna arista, y solo puede
        salir una
22      forall (i in NodosOrigenes)
23          sum(k in 0..(m-1))((sum(<i,j> in Aristas ) SerCon[i][j][k])
24          - (sum(<j,i> in Aristas) SerCon[j][i][k])) == 1;
25
26      #Los servicios pertenecientes al conjunto ServiciosObligados tienen
        que ser realizados
27      forall (i in ServiciosObligados)
28          sum(k in 0..(m-1))sum( <i,j> in Aristas) SerCon[i][j][k] == 1;
29
30      #El número de aristas que entran a un nodo servicio o descanso  menos
        las que salen debe de ser el mismo
31      forall (i in (NodosServicios union NodosDescanso), k in 0..(m-1))
32          ((sum(<i,j> in Aristas) SerCon[i][j][k]) - (sum(<j,i> in Aristas)
        SerCon[j][i][k])) == 0;
33
34      #En los nodos finales solo puede entrar una arista y no puede salir
        ninguna.
35      forall (i in NodosFinales)
36          sum(k in 0 ..(m-1))((sum(<i,j> in Aristas) SerCon[i][j][k]) - (sum
        (<j,i> in Aristas) SerCon[j][i][k])) == -1;
37  }
```

Listing 5.3: *Implementación de las restricciones*

En la siguiente imagen se puede observar las estructuras de datos definidas en CPLEX para representar el modelo matemático planteado.

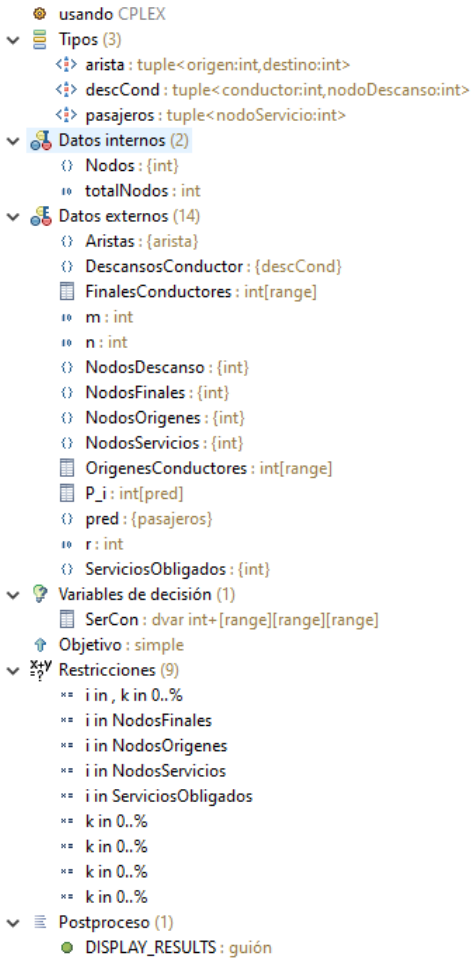


Figura 5.3: *Estructuras de datos utilizadas en CPLEX.*

5.6.4. Extracción de la solución

Una vez el proceso de optimización concluye, se debe extraer de la variable de decisión definida como *SerCon* la asignación de servicios para cada conductor. Para ello simplemente es necesario recorrer los tres índices de la variable y ver si el valor de esta para una determinada asignación (i,j,k) es igual a uno. Para ello se utiliza el siguiente código OPL que se encarga de escribir en un fichero de tipo CSV la solución obtenida.

```
1
2 execute DISPLAY_RESULTS{
3     var f13 = new IloOplOutputFile(#RUTA#);
4     f13.writeln("Conductor;","j;","i");
```

```
    for(var k = 0; k < m; k++){
        for(var j in Nodos){
            for(var i in Nodos){
                if(SerCon[i][j][k] == 1){
                    f13.writeln(k, ";", j, ";", i );
                }
            }
        }
    }
    f13.close();
}
```

Listing 5.4: *Script de extracción de la solución*

Una vez la solución esta recogida en el fichero CSV se lanza un último script Python que se encarga de ordenar para cada conductor los servicios que se le han asignado. Esto permite conocer de forma cronologíca el trabajo que realizará cada conductor en la jornada. El script Python utilizado se encuentra detallado en el apéndice C, además de ordenar la solución el programa se encarga de comprobar que la solución obtenida es correcta. Finalmente, se puede visualizar la solución tal y como se ve en la siguiente imagen.

Conductor	J	I
0	11	57
0	12	11
0	36	12
0	15	36
0	16	15
0	67	16
0	17	67
0	18	17
0	19	18
0	20	19
0	21	20
0	22	21
0	61	22

Conductor	J	I
1	1	58
1	2	1
1	3	2
1	4	3
1	49	4
1	71	49
1	6	71
1	29	6
1	30	29
1	31	30
1	32	31
1	33	32
1	62	33

Conductor	J	I
2	34	59
2	35	34
2	13	35
2	14	13
2	37	14
2	38	37
2	79	38
2	40	79
2	41	40
2	42	41
2	43	42
2	44	43
2	45	44
2	63	45

Conductor	J	I
3	46	60
3	24	46
3	25	24
3	26	25
3	27	26
3	50	27
3	83	50
3	51	83
3	52	51
3	53	52
3	54	53
3	55	54
3	56	55
3	64	56

Figura 5.4: *Visualización de una solución.*

En este caso, se trata de una solución para un problema con cuatro conductores. Las casillas marcadas como azul simbolizan el nodo origen del conductor, las rojas representan el nodo final del conductor y finalmente las marcadas en azul representan el descanso del conductor. Por último, para poder desgranar la solución es necesario conocer a que servicio se refiere cada nodo. Para ello se utiliza un fichero de traducción que fue generado durante la construcción del grafo. Este fichero de traducción tiene una estructura similar a la que aparece en la siguiente imagen:

```
CONDUCTORES --> CÓDIGO (K)
901012 --> 0
901021 --> 1
901022 --> 2
901031 --> 3
IdServicio(j o i) --> caracterisitcas
0 ---> linea: 901, H.Inicio: 6:00, Origen: 9200, Destino: 9450
1 ---> linea: 906, H.Inicio: 6:35, Origen: 9181, Destino: 9453
2 ---> linea: 901, H.Inicio: 7:15, Origen: 9200, Destino: 9450
3 ---> linea: 901, H.Inicio: 7:45, Origen: 9181, Destino: 9200
4 ---> linea: 901, H.Inicio: 8:20, Origen: 9200, Destino: 9450
5 ---> linea: 901, H.Inicio: 9:25, Origen: 9181, Destino: 9200
6 ---> linea: 901, H.Inicio: 10:00, Origen: 9200, Destino: 9450
7 ---> linea: 901, H.Inicio: 10:55, Origen: 9181, Destino: 9200
8 ---> linea: 901, H.Inicio: 11:25, Origen: 9200, Destino: 9450
9 ---> linea: 901, H.Inicio: 12:00, Origen: 9181, Destino: 9200
                                  ...
                                  ...
                                  ...
Nº de pasajeros Recogido --> 2202
Conductores de Baja --> 901011
```

Figura 5.5: *Fichero de traducción de una solución.*

Tal y como se puede ver en el fichero se detalla el índice asignado a cada conductor por el programa así como el índice de cada servicio. Además, para el caso de los servicios, se conoce la línea a la que pertenece, la hora de inicio y los puntos de salida y llegada del viaje. Finalmente, aparece el número de pasajeros que recoge la planificación antes de ser optimizada y el identificador del conductor de baja.

Capítulo 6

Evaluación

En este sección de la memoria se presentan una serie de experimentos realizados con la herramienta ya desarrollada. En primer lugar, se presentan cada uno de las instancias del problema utilizadas en la experimentación. Tras esto se muestran los resultados obtenidos con la herramienta así como el tiempo de ejecución y la mejora que ha supuesto la solución obtenida. Además se realiza una breve discusión acerca de los resultados obtenidos para cada experimento. Finalmente, se describe el entorno en el que fueron desarrollados los experimentos.

6.1. Instancias del problema

Para realizar la experimentación se han definido cuatro escenarios de distinta dimensión entre los que varía el número de viajes y el número de conductores disponibles. Los escenarios definidos son extraídos de datos reales de la empresa TITSA por lo que se puede garantizar la veracidad de las soluciones. En la tabla que aparece a continuación se muestran los detalles de cada una de las instancias del problema utilizadas para la experimentación.

En la tabla anterior se entiende que la columna máximo número de pasajeros establece el número de pasajeros que se recogería en el caso de que no se suspendiese ningún servicio. Por otra parte, la columna que esta justo a su derecha, número de pasajeros recogidos, expresa el número de pasajeros que se recogería si faltan los conductores indicados y no se realiza ninguna cambio, es decir los servicios suspendidos son los que estos tenían asignados.

Instancia	Nº viajes	Nº cond.	Nº cond. baja	Nº cond. disp.	Max. Nº Pasajeros	Nº Pasajeros recogidos	% Pasajeros recogidos
I	*57*	*5*	*1*	*4*	*2749*	*2202*	*80.10 %*
II	*105*	*9*	*2*	*7*	*3992*	*2956*	*74.04 %*
III	*202*	*18*	*4*	*14*	*6244*	*4326*	*69.28 %*
IV	*406*	*39*	*5*	*34*	*45181*	*37489*	*82.97 %*

6.2. Resultados obtenidos

A continuación se presentan los resultados que fueron obtenidos por la herramienta para cada instancia del problema previamente descrita. Además de una breve discusión de los resultados logrados, se detalla el tiempo de ejecución que fue necesario para que CPLEX pudiese encontrar la solución óptima del problema que maximiza el número de pasajeros recogidos.

6.2.1. Instancia I

Los resultados obtenidos para la instancia I se muestran en la siguiente tabla.

Nº viajes realizados	Nº pasajeros recogidos	% pasajeros recogidos	% pasajeros antes de optimizar	Tiempo ejecución
46	2333	84.86 %	80.10 %	1.38"

Tal y como se puede apreciar los resultados obtenidos no muestran un gran aumento en el porcentaje de pasajeros recogidos. Esto es debido a que el conductor de baja tenía un horario de entrada muy distinto al de los conductores disponibles de forma que las horas de su jornada que se solapasen con las de los conductores disponibles eran pocas. Esto implica que algunos de los servicios que habían sido asignados al conductor de baja no van a poder ser realizados nunca y se da la coincidencia que la predicción de pasajeros para alguno de ellos era alta.

6.2.2. Instancia II

En la tabla que se muestra seguidamente se pueden observar los resultados obtenidos para esta instancia del problema.

Nº viajes realizados	**Nº pasajeros recogidos**	**% pasajeros recogidos**	**% pasajeros antes de optimizar**	**Tiempo ejecución**
88	3662	91.73 %	74.04 %	5.45"

De los resultados obtenidos en la tabla se puede apreciar como en este caso el porcentaje de pasajeros recogidos con la nueva planificación ha aumentado notoriamente pasando de un 74.04 % a un 91.73 %. El hecho de que el porcentaje de pasajeros recogidos sea tan alto a la vez que el número de servicios realizados, 88 de 105, sea en proporción inferior, 83.80 % de servicios realizados, es un buen indicador de que la red de transporte puede ser optimizada de forma que con un menor número de conductores se puede lograr recoger a una gran cantidad de pasajeros. Entonces, tal y como se puede deducir la herramienta desarrollada no tiene un único uso para reorganizar los servicios sino que además permite estudiar distinto casos teóricos que pueden ser de interés para la compañía TITSA para tener cierta información analítica de sus redes de transporte.

Finalmente, comentar como a pesar de haber doblado el número de viajes de la instancia I, el tiempo de ejecución sigue siendo considerablemente bajo y por lo tanto asumible. Esto implica que para instancias de esta índole no necesariamente se deba de conocer la falta de un conductor con un día de antelación ya que aunque se informe el mismo día de su falta la reorganización de los servicios no supone apenas un coste temporal. Esto último implica que la herramienta puede ser usada en un mayor número de casos y problemáticas.

6.2.3. Instancia III

Los resultados obtenidos para la instancia III se muestran a continuación en forma de tabla.

Nº viajes realizados	**Nº pasajeros recogidos**	**% pasajeros recogidos**	**% pasajeros antes de optimizar**	**Tiempo ejecución**
167	5539	88.71 %	69.28 %	1' 10"

Los resultados obtenidos para esta instancia resultan reveladores dado que a pesar de faltar una gran porción de los conductores, 4 de 18, y el número de viajes realizados sea tan bajo proporcionalmente, 82.67 %, el porcentaje de pasajeros recogidos ha aumentado notablemente pasando de un 69.28 % a un 88.71 %. En cuanto al tiempo de ejecución, se puede apreciar como en este caso si que se ha necesitado algo más de tiempo para encontrar la solución óptima. Sin embargo, sigue siendo asumible invertir 1' y 10"para encontrar la solución óptima, por lo que se deduce que problemas con esta dimensión pueden ser resueltos el mismo día de la falta de un conductor.

Llegados a este punto se puede apreciar como el comportamiento en líneas generales de la herramienta es altamente productivo dado que con un coste temporal ínfimo hasta el momento, se puede conseguir una gran mejora.

6.2.4. Instancia IV

Por último los resultados obtenidos para instancia IV se expresan en la siguiente tabla.

Nº viajes realizados	**Nº pasajeros recogidos**	**% pasajeros recogidos**	**% pasajeros antes de optimizar**	**Tiempo ejecución**
370	44264	98.77 %	82.97 %	45' 1"

De los resultado obtenidos se puede concluir como el hecho de que la proporción de conductores que faltaron sea menor posibilita que el conjunto de soluciones posibles sea mucho mayor y por lo tanto existe una mayor posibilidad de que el número de pasajeros recogidos sea mayor. En este caso se ha pasado de tener un 82.97 % a un 98.77 %, pero a diferencia de las instancias anteriores el tiempo necesario para encontrar la solución óptima

si ha aumentado considerablemente superando los 45'. Por lo tanto, en el caso de que surjan problemas de esta dimensión, estos deberán ser identificados con la suficiente antelación para lanzar una ejecución.

Cabe destacar que por normal general en la realidad no se darán problemas de esta dimensión dado que el número de viajes a replanificar no será tan grande. Por otra parte, si lo que se pretende es obtener información analítica de la red de servicios, que el tiempo de ejecución sea bajo no es algo primordial.

6.2.5. Entorno de experimentación

Como se ha podido apreciar el tiempo de ejecución varía en función del tamaño del problema que se resuelve. Además del tamaño, las características del ordenador que ejecuta el programa influyen considerablemente en el tiempo de ejecución. De esta manera, es importante conocer las características del ordenador en el que fueron realizados los experimentos. En este caso se trata de un ordenador con el sistema operativo Windows 10 con las siguientes características:

Procesador:	Intel(R) Core(TM) i5-6600 CPU @ 3.30GHz 3.31 GHz
Memoria instalada (RAM):	16,0 GB (15,9 GB utilizable)
Tipo de sistema:	Sistema operativo de 64 bits, procesador x64

Figura 6.1: *Entorno de experimentación.*

Capítulo 7

Conclusiones y líneas futuras

7.1. Conclusiones

El sector del transporte se caracteriza por un elevado consumo de combustibles fósiles y un gran impacto medioambiental, donde el transporte terrestre es el que concentra el mayor porcentaje de consumo (un 80 % del consumo final del sector (IDAE)). Además, dentro del transporte por carretera, es la movilidad en coche la que produce la mayor parte de los efectos negativos derivados del transporte, tales como congestión, ruido, riesgo de accidentes, contaminación, etc. De ahí que se hace necesario la búsqueda de alternativas de transporte que contribuyan a la sostenibilidad del sector. Es por esto, que el transporte público debe de ser una alternativa competitiva y cómoda para el usuario garantizando un servicio de calidad y fiable.

Actualmente, las nuevas tecnologías ofrecen una gran oportunidad para estudiar y mejorar el servicio prestado por un empresa. Es en este contexto, en el que la investigación en formas alternativas de mejorar el transporte tiene una gran cabida. El uso del transporte público es una clara alternativa para solucionar los problemas relacionados con las pautas de movilidad actuales y es aquí donde se centra la línea de investigación de este trabajo. El transporte público será una alternativa mejor o peor en función de la calidad del servicio que ofrece por lo que es de gran interés mejorarlo en la mayor medida posible.

En este trabajo se trata de contribuir a la empresa de transporte público TITSA mediante

el desarrollo de una herramienta capaz de optimizar la recogida de pasajeros en caso de falta de recursos. Con la finalización del trabajo presente, se consigue una planificación de la jornada mucho más eficiente que la que se llevaba a cabo anteriormente. Esto incide directamente en la calidad del servicio prestado, dado que además de contar con la mejor solución posible en caso de que surja un problema, la aplicación creada permite realizar estudios acerca del uso de recursos y permite definir protocolos de actuación.

Para conseguir optimizar la planificación de una jornada se ha diseñado un modelo matemático en el que se representa mediante un grafo acíclico el espacio de soluciones del problema. Este planteamiento presenta ventajas a la hora de incorporar nuevas restricciones al problema y a su vez permite utilizar distintas tecnologías que optimicen el modelo. Además, se ha hecho una fase de experimentación sobre la herramienta con un conjunto de diferentes casos de estudios en los que se ha comprobado que los resultados aportados por la misma en un corto periodo de tiempo son bastante buenos, en términos de captación de usuarios. Por otra parte, como herramienta necesaria para construir el problema se desarrollo un perceptrón multicapa capaz de predecir el número de pasajeros que tendrá un determinado día una línea concreta. El uso de esta herramienta, implementada en keras, no es solo útil para la construcción del problema ya que permite conocer datos de gran interés para la empresa que pueden ser utilizados en otros casos de estudio.

Por último, cabe remarcar la aplicabilidad directa del trabajo que ha sido desarrollado. Durante el proceso de implementación así como el de evaluación, se han utilizado datos reales de la empresa TITSA que han permitido conocer con certeza la calidad de las soluciones obtenidas y la utilidad de las mismas.

7.2. Líneas futuras

Una vez la herramienta ha sido realizada, se definen una serie de líneas de actuación futuras que pretenden mejorar la aplicación, instaurar su uso en la metodología de trabajo de la empresa, realizar informes y demás. A continuación se listan las distintas tareas que

están planificadas para realizar en el futuro:

- Crear un cuadro de mando con POWER BI en el que se pueda apreciar una comparación clara entre el número de pasajeros recogidos cuando se realizaba una replanificación con el método tradicional y el número de pasajeros que se hubiera recogido si se hubiese utilizado la aplicación desarrollada. El método tradicional de reorganización es llevado a cabo por el jefe del área afectada que simplemente asigna bajo su criterio a los distintos conductores.

- Implementar el modelo matemático en las librerías de CPLEX para Python para así poder automatizar en los servidores de la empresa un proceso que lance cada vez que haya una falta de conductor el programa creado.

- Realizar estudios en los que se analice como influiría hipotéticamente la falta de un conductor las distintas zonas de la red de transporte. De esta forma, cuando surge un problema se puede conocer de antemano la gravedad que puede tener.

- Establecer protocolos de actuación en caso de problemática. Si se tienen estudiados todos los posibles casos de problemas que puedan surgir, la solución puede ser inmediata.

Bibliografía

[1] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.

[2] Joey Bernard. Python data analysis with pandas. In *Python Recipes Handbook*, pages 37–48. Springer, 2016.

[3] Mónica Bocco, Enrique Willington, Mónica Arias, et al. Comparison of regression and neural networks models to estimate solar radiation. *Chilean Journal of Agricultural Research*, 70(3):428–435, 2010.

[4] Juan Ricardo Rugerio Breton. Comparacion de metodos heuristicos para el problema de coloreado de grafos. 2018.

[5] Yeicy Bermúdez Colina. Aplicaciones de programación lineal, entera y mixta. *Ingeniería Industrial. Actualidad y Nuevas Tendencias*, 2(7):85–104, 2011.

[6] Rafael Martí Cunquero. Algoritmos heurísticos en optimización combinatoria. *Valencia: Universodad de Valencia. Retrieved*, 11(01):2012, 2003.

[7] Carmen Esclapés. *Asignación de conductores a jornadas de trabajo en empresas de transporte colectivo*. PhD thesis, PhD thesis, Universitat Politecnica de Catalunya, Spain, 2000.

[8] Ali Haghani, Huijun Hu, and Qiang Tian. An optimization model for real-time emergency vehicle dispatching and routing. In *82nd annual meeting of the Transportation Research Board, Washington, DC*. Citeseer, 2003.

[9] IBM. Ibm ilog cplex optimization studio opl language user's manual. 2016.

[10] Juan A Mesa, Francisco A Ortega, and Miguel A Pozo. A geometric model for an effective rescheduling after reducing service in public transportation systems. *Computers & Operations Research*, 40(3):737–746, 2013.

[11] Evelien Van der Hurk, Leo G Kroon, Gábor Maróti, and Peter Vervest. Dynamic forecast model of time dependent passenger flows for disruption management. In *12th conference on advanced systems for public transport in Santiago, Santiago, Chile*, pages 23–27, 2012.

[12] H Paul Williams. *Model building in mathematical programming*. John Wiley & Sons, 2013.

Apéndice A

Código C++ tratamiento de datos

A.1. Main.cpp

```
#include <iomanip>
#include <cstdio>
#include <string.h>
#include <iostream>
#include <fstream>
#include <algorithm>
#include <map>
#include "entrada.hpp"
using namespace std;
int main (int argc, char **argv){
    ifstream fich (argv[1]);
    map<entrada,int> lineasFinales;
    if(fich.is_open()){
        string line;
        char * pch;
        getline(fich,line);
        while ( getline (fich,line)){
```

```
char *y = new char[line.length() + 1]; // or
strcpy(y, line.c_str());
pch = strtok (y,";");
int dia, mes, anio, hora, linea;
const char* charstring = pch;
string fecha(charstring);
if(fecha[1] == '/'){
    dia = stoi(fecha.substr(0,1));
    mes = stoi(fecha.substr(2,2));
    anio = stoi(fecha.substr(5,4));
}
else{
    dia = stoi(fecha.substr(0,2));
    mes = stoi(fecha.substr(3,2));
    anio = stoi(fecha.substr(6,4));
}
int j = fecha.find(" ") + 1;
hora = stoi(fecha.substr(j ,2));
pch = strtok (NULL, ";");
linea = stoi(pch);
pch = strtok (NULL, ";");
pch = strtok (NULL, ";");
pch = strtok (NULL, ";");
pch = strtok (NULL, ";");
entrada x(dia, mes, anio, hora, linea);
pair<map<entrada, int>::iterator,bool> salida;
salida = lineasFinales.insert(make_pair(x,1));
```

```
            if(salida.second == false){
                salida.first->second = salida.first->second + 1;
            }
        }
        fich.close();
        ofstream fs("ficheroParseado.csv");
        fs << "dia;mes;año;hora;linea;numUsuarios\n";

        for(map<entrada, int>::iterator it = lineasFinales.begin();
         it != lineasFinales.end(); ++it){
            fs  <<it->first.toString() << it->second << "\n";
        }
        fs.close();
    }
    else{
        cout << "No se pudo abrir/encontrar el fichero" << endl;
    }
}
```

A.2. Entrada.cpp

```
#include "entrada.hpp"
#include <iostream>
#include <cstdio>
using namespace std;
string entrada::toString(void) const{
string out ="";
```

```
    out += to_string(dia) + ";" + to_string(mes) + ";" + to_string(anio) + ";"
     + to_string(hora) + ";" + to_string(linea) + ";";
    return out;
}
entrada::entrada(void):
dia(0),
mes(0),
anio(0),
hora(0),
linea(0),
usuarios(0)
{}
entrada::entrada(int dia, int mes, int anio, int hora, int linea){
this->dia = dia;
this->mes = mes;
this->hora = hora;
this->anio = anio;
this->linea = linea;
this->usuarios = 0;
}
entrada::~entrada(void){
dia = 0;
mes = 0;
anio = 0;
hora = 0;
linea = 0;
usuarios = 0;
```

```
}
int& entrada::getDia(void){
return dia;
}
int& entrada::getMes(void){
return mes;
}
int& entrada::getHora(void){
return hora;
}
int& entrada::getLinea(void){
return linea;
}
int& entrada::getSetUsuarios(void){
return usuarios;
}
int entrada::getDia(void) const{
return dia;
}
int entrada::getMes(void)const{
return mes;
}
int entrada::getHora(void)const{
return hora;
}
int entrada::getLinea(void)const{
return linea;
```

```
}
bool entrada::operator< (const entrada & entry) const
{
if((this->dia < entry.getDia()) || (this->mes < entry.getMes()) ||
    (this->hora < entry.getHora()) || this->linea < entry.getLinea()){
return true;
}
return false;
}
```

A.3. Entrada.hpp

```
#pragma once
#include <iostream>
#include <cstdio>
using namespace std;
class entrada{
private:
int dia,mes,anio,hora,linea, usuarios;
public:
entrada(void);
entrada(int dia, int mes, int anio, int hora, int linea);
~entrada(void);
int& getDia(void);
int& getHora(void);
int& getMes(void);
int& getLinea(void);
int& getSetUsuarios(void);
```

```
bool operator< (const entrada & entry)const;
int getDia(void)const;
int getHora(void)const;
int getMes(void)const;
int getLinea(void)const;
string toString(void)const;

};
```

Apéndice B

Programa de formateo de datos

```
import sys
import requests
import lxml.html as lh
import pandas as pd
import os
import errno

def buscarTiempoViaje(dfTiempo, idParada1, idParada2):
    if idParada1 ==  idParada2 :
        return 0
    encontrado = False
    tiempo = 0
    for idx, row in dfTiempo[dfTiempo["stop_id_1"] == idParada1].iterrows():
        if row["stop_id_2"] == idParada2 :
            return int(row["Distance"])
    for idx, row in dfTiempo[dfTiempo["stop_id_1"] == idParada2].iterrows():
        if row["stop_id_2"] == idParada1 :
            return int(row["Distance"])
```

```
def CalcularHorarioConductores(df,conductoresBaja):
    horario_ = {}
    for i in range(0, df.shape[0]) :
        cod_turno = str(df.iloc[i]["COD_TURNO"])
        inicio = str(df.iloc[i]["INICIO"])
        fin = str(df.iloc[i]["FIN"])
        destino = int(df.iloc[i]["DESTINO"])
        origen = int(df.iloc[i]["ORIGEN"])
        inicio = int(inicio.split(":")[0]) * 60 + int(inicio.split(":")[1])
        fin = int(fin.split(":")[0]) * 60 + int(fin.split(":")[1])
        if cod_turno in horario_:
            if horario_[cod_turno]["in"] > inicio :
                horario_[cod_turno]["in"] = inicio
                horario_[cod_turno]["ptoIni"] = origen
            if horario_[cod_turno]["out"] < fin :
                horario_[cod_turno]["out"] = fin
                horario_[cod_turno]["ptoFin"] = destino
        elif cod_turno not in conductoresBaja :
            horario_[cod_turno] = {"in" : inicio, "out": fin, "ptoIni" : origen,
             "ptoFin" : destino}
    return horario_


if __name__ == "__main__":
```

```
df = pd.read_csv(sys.argv[2], sep = ";")
dfTiempo = pd.read_csv(sys.argv[3], sep = ";")



Aristas = "Aristas = {\n"
DescansosConductor = "DescansosConductor = {\n"
pred = "pred = { \n"
P = "P_i = #[\n"
dfBackTrack = df



try:
    serviciosSuspendidos = sys.argv[1].split(',')
except:
    print("Añadir como argumento el número de los empleados
     que han faltado como una lista de numeros separados por comas")
    exit(-1)

enter = 0
servicios = []
conductores = []
conductoresBaja = serviciosSuspendidos
origenes = []
finales = []
descansos = []
descansosInicio = []
frecuenciaDescansos = 20
```

```
numDescansos = 6
PasajerosRecogidos = 0
DocumentoTraduccion = "CONDUCTORES --> CÓDIGO (K) \n"



for i in range(0, df.shape[0]) :
    cod_turno = str(df.iloc[i]["COD_TURNO"])
    pasajeros = str(df.iloc[i]["NumPasajeros"])
    P += "<" + str(i) + "> : " + pasajeros + "\n"
    if cod_turno not in conductoresBaja :
        PasajerosRecogidos += int(pasajeros)
    #if i not in servicios:
    servicios.append(i)
    if cod_turno not in conductores and cod_turno not in conductoresBaja:
        conductores.append(cod_turno)



horario = CalcularHorarioConductores(df,conductoresBaja)

indexConductor = df.shape[0]
chofer = 0
j = df.shape[0] + (2 * len(conductores))
for key, value in horario.items():
    DocumentoTraduccion += str(key) +  " --> " +  str(chofer) + "\n"
    origenes.append(indexConductor + chofer)
    finales.append(indexConductor + len(horario) + chofer)
    chofer += 1
```

```
    hInicio = value["in"] + 3 * 60
    for i in range(0, numDescansos) :
        descansosInicio.append(hInicio + (i * frecuenciaDescansos))
        DescansosConductor += "<" + str(chofer - 1) + ", " + str(j) + ">\n"
        descansos.append(j)
        j += 1

NodosServicios = "NodosServicios = {"
retorno  = 0
DocumentoTraduccion += "IdServicio(j o i) --> caracterisitcas\n"
# En este bucle se construyen las aristas entre los servicios
for i in range(0, df.shape[0]) :
    NodosServicios += " " + str(i)
    pred += "<" + str(i) + "> "
    retorno += 1
    if retorno == 15:
        NodosServicios += "\n"
        pred += "\n"
        retorno = 0
    cod_turno = str(df.iloc[i]["COD_TURNO"])
    cod_linea = str(df.iloc[i]["COD_LINEA"])
    destino = str(df.iloc[i]["DESTINO"])
    origen = str(df.iloc[i]["ORIGEN"])
    inicio = str(df.iloc[i]["INICIO"])
    fin = str(df.iloc[i]["FIN"])

    DocumentoTraduccion += str(i) + " ---> linea: " + cod_linea +
```

```
 ", H.Inicio: " + inicio + ", Origen: " + origen +
                                                ", Destino: " + destino + "\n"

inicio_h = int(inicio.split(":")[0])
inicio_m = int(inicio.split(":")[1])
inicio = inicio_h * 60 + inicio_m

fin_h = int(fin.split(":")[0])
fin_m = int(fin.split(":")[1])
fin = fin_h * 60 + fin_m



print(i + 1, " de ", df.shape[0])
# servicio - servicio
for j in range(i + 1, df.shape[0]):
    destino_j = str(df.iloc[j]["DESTINO"])
    origen_j = str(df.iloc[j]["ORIGEN"])
    inicio_j = str(df.iloc[j]["INICIO"])
    fin_j = str(df.iloc[j]["FIN"])

    inicio_h = int(inicio_j.split(":")[0])
    inicio_m = int(inicio_j.split(":")[1])
    inicio_j = inicio_h * 60 + inicio_m

    fin_h = int(fin_j.split(":")[0])
    fin_m = int(fin_j.split(":")[1])
    fin_j = fin_h * 60 + fin_m
```

```
    tiempoTraslado = buscarTiempoViaje(dfTiempo, int(destino), int(origen_j))
    tiempoTraslado_j = buscarTiempoViaje(dfTiempo, int(destino_j), int(origen))
    #print(destino, " --- ", origen_j)
    if (fin + tiempoTraslado) <= inicio_j :
        Aristas += "< " + str(i) + ", " + str(j) + ">,"
        enter += 1
        if enter == 3 :
            Aristas += "\n"
            enter = 0

    elif (fin_j + tiempoTraslado_j) <= inicio :
        Aristas += "< " + str(j) + ", " + str(i) + ">,"
        enter += 1
        if enter == 3 :
            Aristas += "\n"
            enter = 0

#servicio - descanso  y descanso - servicio
for j in range(0, len(descansosInicio)):
    tiempoTraslado = buscarTiempoViaje(dfTiempo, int(destino) , 9462)
    if ( fin + tiempoTraslado) <= descansosInicio[j] :
        Aristas += "< " + str(i) + ", " + str(df.shape[0] +
         (2 * len(conductores)) + j) + ">,"
        enter += 1
        if enter == 3:
            Aristas += "\n"
```

```
            enter = 0
    tiempoTraslado = buscarTiempoViaje(dfTiempo, 9462, int(origen))
    if ( descansosInicio[j] + 25 + tiempoTraslado) <= inicio :
        Aristas += "< " + str(df.shape[0] + (2 * len(conductores)) + j) +
         ", " + str(i) + ">,"
        enter += 1
        if enter == 3:
            Aristas += "\n"
            enter = 0

#origen-servicio y servicio-fin
chofer = 0
for key, value in horario.items():
    tiempoTraslado = buscarTiempoViaje(dfTiempo, value["ptoIni"], int(origen))
    if (value["in"] + tiempoTraslado) <= inicio :
        Aristas += "< " + str(indexConductor + chofer) + ", " + str(i) + ">,"
        enter += 1
        if enter == 3:
            Aristas += "\n"
            enter = 0

    tiempoTraslado = buscarTiempoViaje(dfTiempo, int(destino), value["ptoFin"])
    if (fin + tiempoTraslado) <= value["out"] :
        Aristas += "< " + str(i) + ", " + str(indexConductor
         + len(conductores) + chofer) + ">,"
        enter += 1
        if enter == 3:
```

```
                Aristas += "\n"
                enter = 0
        chofer += 1



WriteTxtFile = open("FicherosJornadasCPLEX\\"
 + str(sys.argv[2].split("\\")[1][:-4]) + ".dat", "w")
WriteTxtFile2 = open("FicherosJornadasCPLEX\\"
 + str(sys.argv[2].split("\\")[1][:-4]) + "_TRADUCCION.txt", "w")
WriteTxtFile3 = open("FicherosJornadasCPLEX\\OrigenesConductores_"
 + str(sys.argv[2].split("\\")[1][:-4]) + ".json", "w")

for i in range(df.shape[0], df.shape[0] + len(descansos) + 2 * len(origenes)):
    pred += "<" + str(i) + "> "
    retorno += 1
    if retorno == 15:
        pred += "\n"
        retorno = 0
    P += "<" + str(i) + "> : " + str(0) + "\n"



Aristas += "};"
DescansosConductor += "};"
pred += "};"
P += "]#;"
```

```
OrigenesConductores = "OrigenesConductores = #[\n"
origCondJSON = "{\n"
FinalesConductores = "FinalesConductores = #[\n"
WriteTxtFile.write (Aristas + "\n" + pred + "\n" + P + "\n"
 + DescansosConductor + "\n")

index = 0
enter = 0
cadena = "NodosOrigenes = {"
for element in origenes :
    cadena += str(element) + " "
    OrigenesConductores += str(index) + " : " + str(element) + "\n"
    origCondJSON += "\t \"" + str(index) + "\" : " + str(element)
    if index != (len(origenes) - 1) :
        origCondJSON += ",\n"
    index += 1
    enter += 1
    if enter ==  15 :
        cadena += "\n"
        enter = 0
cadena += "};\n"
WriteTxtFile.write (cadena)
origCondJSON += "\n}"
WriteTxtFile3.write(origCondJSON)
WriteTxtFile3.close()
```

```
enter = 0
index = 0
cadena = "NodosFinales = {"
for element in finales :
    cadena += str(element) + " "
    FinalesConductores += str(index) + " : " + str(element) + "\n"
    index += 1
    enter += 1
    if enter ==  15 :
        cadena += "\n"
        enter = 0
cadena += "};\n"
WriteTxtFile.write (cadena)



enter = 0
cadena = "NodosDescanso = {"
for element in descansos :
    cadena += str(element) + " "
    enter += 1
    if enter ==  15 :
        cadena += "\n"
        enter = 0
cadena += "};\n"
WriteTxtFile.write (cadena)
```

```
WriteTxtFile.write(NodosServicios + "};\n")

WriteTxtFile.write("n = " + str(df.shape[0]) + ";\n")

WriteTxtFile.write("m = " + str(len(origenes)) + ";\n")

WriteTxtFile.write("r = " + str(len(descansos)) + ";\n")

WriteTxtFile.write(OrigenesConductores + "]#;\n")

WriteTxtFile.write(FinalesConductores + "]#;")

WriteTxtFile2.write(DocumentoTraduccion + "\n")
WriteTxtFile2.write("N° de pasajeros Recogido --> "
 + str(PasajerosRecogidos) + "\nConductores de Baja --> ")

for element in conductoresBaja :
    WriteTxtFile2.write(str(element) + " ")

WriteTxtFile.close()
WriteTxtFile2.close()
```

Apéndice C

Programa de ordenado de soluciones

```
    import sys
import requests
import lxml.html as lh
import pandas as pd
import os
import errno
import json



if __name__ == "__main__":
    df = pd.read_csv("solution.csv", sep = ";")
    with open(sys.argv[1], 'r') as json_file:
        origCond = json.load(json_file)
    conductores = df.Conductor.unique()
    col = []
    col.append(("Conductor",[]))
    col.append(("J",[]))
    col.append(("I",[]))
```

```
for i in conductores :
    first  = True
    buscar = 0
    tam = df.loc[df['Conductor'] == i].shape[0]
    for r in range(0,tam):
        if first:
            first = False
            #print(i, " ", origCond[str(i)])
            aux = df.loc[(df['Conductor'] == i) & (df['i'] == origCond[str(i)])]
            col[0][1].append(i)
            buscar = aux.iloc[0].values[1]
            col[1][1].append(int(buscar))
            col[2][1].append(int(aux.iloc[0].values[2]))
            #print(aux.values)
        else :
            aux = df.loc[(df['Conductor'] == i) & (df['i'] == buscar)]
            #print(aux.values)
            col[0][1].append(i)
            buscar = aux.iloc[0].values[1]
            col[1][1].append(int(buscar))
            col[2][1].append(int(aux.iloc[0].values[2]))

Dict = {title:column for (title,column) in col}
finalDF = pd.DataFrame(Dict)
if df.shape[0] == finalDF.shape[0] :
    print()
    print("--- LA SOLUCIÓN FUE ORDENADA CORRECTAMENTE ---")
```

```
else :
    print()
    print("--- LA SOLUCIÓN ES ERRONEA ---")
finalDF.to_csv( "SolucionesOrdenadas\\solOrdenada_Urbano_"
+ str(sys.argv[1].split("\\")[1].split("_")[2][:-5])
+ ".csv", sep=';', index = False)
```