Sección de Matemáticas
Universidad de La Laguna

Saray Pérez Delgado

Sistema inteligente para buscar la eficiencia de los puntos de acceso (paradas) a la red de transporte en Tenerife

Smart system to find the efficiency of the access points (bus stops) to the transport network in Tenerife

Trabajo Fin de Grado
Grado en Matemáticas
La Laguna, Marzo de 2022

Dirigido por
Ginés León Rodríguez
Antonio Alberto Sedeño Noda

Ginés León Rodríguez
Dpto. de Big Data y Data Science
TITSA
38111 Sta. María del Mar, Tenerife

Antonio Alberto Sedeño Noda
Dpto. de Matemáticas, Estadística e Investigación Operativa.
Universidad de La Laguna
38200 La Laguna, Tenerife

Agradecimientos

A mis padres. A mis amigos. A mis tutores.

Saray Pérez Delgado
La Laguna, 10 de marzo de 2022

Resumen · Abstract

Resumen

En la isla de Tenerife existen puntos de acceso o paradas de guaguas insatisfactorias desde la perspectiva de la empresa de Transporte Interurbano de Tenerife, TITSA. En este proyecto de fin de grado intentaremos modelizar y solucionar este problema a partir de un caso práctico, así como utilizar un sistema inteligente creado mediante algoritmos para estudiar los datos de las líneas susceptibles a cambiar sus recorridos. El propósito es optimizar el tiempo de viaje del servicio modificado.

Palabras clave: *Puntos de acceso– Optimización– Sistema inteligente – Algoritmos – Grafo dirigido.*

Abstract

On Tenerife island there are unsatisfactory access points or bus stops from the perspective of the Tenerife Interurban Transport company, TITSA. In this final degree project we will try to model and solve this problem from a practical case, as well as how to use an intelligent system to study the data of the lines susceptible to change their routes. The purpose is to optimize the travel time of the modified service.

Keywords: *Access points –Optimization – Intelligent system – Algorithm - Directed graph.*

Contenido

Agradecimientos III

Resumen/Abstract V

Introducción IX

1. Fundamentos teóricos y tratamiento de datos 1
1.1. Artificial Intelligence & Machine learning 1
1.1.1. ¿Qué se requiere para crear buenos sistemas de machine learning? 2
1.1.2. ¿Quién lo utiliza? 2
1.2. TITSA: tratamientos de datos 3
1.2.1. Modelos predictivos y redes neuronales 3
1.2.2. Visualización de datos 4
1.3. Programación lineal 6
1.3.1. Método del simplex 7
1.3.2. Entorno de programación 8

2. Presentación del problema general y objeto de estudio 13
2.1. Representación del problema 13
2.1.1. Caso particular 14

3. Resolución del problema particular 17
3.1. Datos 17
3.1.1. Modelización de los datos 18
3.2. Grafo del problema 22
3.3. Modelo del problema 23
3.4. Algoritmos 25
3.4.1. Resolución del problema $P_{\bar{T}}(Y,T)$ 25
3.5. Resolución con Python 29
3.5.1. Creación del grafo - Fase 1 30

3.5.2. Creación del grafo - Fase 2 32

A. Apéndice 39
A.1. Preproceso 39
A.2. Creación del grafo - 1ª FASE 40
A.3. Creación de las listas para las búsquedas dicotómicas 42
A.4. Creación del grafo - 2ª FASE 42
A.5. Script de correspondencia de línea y trayecto con la capacidad óptima 46

Bibliografía 47

Lista de símbolos y abreviaciones 49

Poster 51

Introducción

En este trabajo de fin de grado mostraremos como la aplicación de técnicas de inteligencia artificial junto con la programación matemática ayudan a tomar mejores decisiones en el desempeño de una empresa.

Nuestro objetivo es mejorar la eficiencia de determinados puntos de acceso a la red de Transporte Interurbano de Tenerife. Para ello, este proyecto constará de tres capítulos.

En primer lugar, veremos cómo TITSA se nutre del Machine Learning y de la AI (Artificial Intelligence) para adentrarse en el mundo del Big Data & Data Science. Explicaremos también como hemos tratado y limpiado los datos que usaremos con posterioridad en este proyecto. Para seguir con una pequeña introducción al mundo de la programación lineal.

En el segundo capítulo nos centraremos en un problema real que presenta la empresa con algunas paradas de la red de transporte. Además, identificaremos varias variantes del problema y sus respectivas soluciones posibles. Haremos hincapié en un caso particular del problema.

Por último, en el tercer capítulo modelizaremos y resolveremos dicho caso particular presentado anteriormente. Para ello, analizaremos el tratamiento de los datos utilizados.

El trabajo finalizará con las conclusiones y las propuestas de futuros proyectos para extender este tipo de estudio.

1

Fundamentos teóricos y tratamiento de datos

La empresa de Transportes Interurbanos de Tenerife, más conocida como TITSA, a finales de 2017 empieza a dar los primeros pasos para implantar en su estructura el departamento Big Data y Data Science, a medida que va demostrando que, aunque los sistemas siempre recogerán anomalías en los datos, existen técnicas, procesos y herramientas que permiten limpiarlo, depurarlo, y darle una fiabilidad nunca vista antes.

Si el mundo de hoy representa ya una explosión de datos y el poder de la AI (Artificial Intelligence), los próximos años mostrarán que la era actual es el primer paso de una transformación masiva. TITSA debe estar posicionada para este momento, incluso si esa visión se extiende más allá de la tenencia de los líderes actuales.

Es por ello que, analizando los problemas actuales que presenta la empresa, se requiere un sistema inteligente que detecte cambios de flujo entre las diferentes zonas de transporte de la isla, así como la relación de los diferentes puntos de acceso a la red y los servicios que se ofrecen entre ellos.

1.1. Artificial Intelligence & Machine learning

En términos informáticos nos referimos a la inteligencia artificial como aquella que es expresada por las máquinas, los procesadores y los softwares, es decir, en referencia a la inteligencia natural, lo equivalente al cuerpo, el cerebro y la mente, respectivamente. Dentro de la inteligencia artificial nos encontramos con una de sus ramas, el machine learning o en español, el aprendizaje automatizado. Este es un método de análisis de datos que automatiza la construcción de modelos analíticos. Está basado en la idea de que los sistemas pueden aprender de los datos e identificar ciertos patrones en ellos.

1.1.1. ¿Qué se requiere para crear buenos sistemas de machine learning?

Con el machine learning podemos construir modelos de forma rápida y automática que nos permitan analizar mayores y más complejas cantidades de datos, así como producir resultados más rápidos y precisos. Para ello requerimos de:

1. Recursos de preparación de datos:
   En esta fase es muy importante extraer los datos, tratarlos, limpiarlos y dejarlos listos para empezar a trabajar. De este modo, si tenemos una pequeña cantidad de datos nos será suficiente con una hoja de cálculo (Excel, Google Spreadsheets...) mientras que, si nos encontramos con una gran cantidad de estos, tendremos que hacer uso de programas como R o Python.
2. Algoritmos: básicos y avanzados:
   Se utilizan algoritmos programados con el fin de recibir y analizar los datos de entrada para predecir los datos de salida. A medida que se introducen nuevos datos, estos algoritmos van aprendiendo a optimizar sus operaciones y mejorar su rendimiento, es decir, van obteniendo "inteligencia".
3. Automatización y procesos iterativos:
   Conseguir un sistema que, de forma automática, nos permita reconocer dichos patrones y pueda llevar a cabo todos estos procesos de manera eficiente y robusta, teniendo en cuenta tiempo de ejecución y restricciones computacionales.
4. Escalabilidad:
   El machine learning aborda dos áreas diferentes de escalabilidad. El primero es el entrenamiento de un modelo de un gran conjunto de datos. El segundo es poner en uso el modelo entrenado para que pueda extenderse o escalarse para satisfacer las necesidades de las aplicaciones que lo utilizan.

1.1.2. ¿Quién lo utiliza?

1. Servicios financieros:
   Algunas empresas de la industria financiera utilizan el machine learning o el aprendizaje automático para dos propósitos principales: prevenir el fraude e identificar información importante en los datos. Esta información o Insights pueden identificar oportunidades de inversión o ayudar a los inversores a comprender cuándo comprar o vender. Del mismo modo, también puede identificar a los clientes con características de alto riesgo o, por medio de vigilancia automática, detectar señales de fraude.
2. Gobierno:
   Tanto la seguridad como los servicios públicos tienen numerosas fuentes de

datos de las que se podría extraer información valiosa. El aprendizaje automático en estos casos puede detectar de nuevo posibles fraudes así como minimizar el robo de identidad o identificar formas de incrementar el capital y/o ahorrar dinero.

3. Marketing y ventas:
   Las páginas web que recomiendan contenido parecido al buscado con anterioridad se lucran del machine learning para analizar el historial de búsqueda o compra. El futuro del comercio online está en manos de esta capacidad de capturar datos, analizarlos y utilizarlos de manera personalizada.
4. Petróleo y gas:
   El uso de la inteligencia artificial en este sector es cada vez mayor y esto se debe a que se puede sacar partido de ella para la predicción de fallos de sensores o para optimizar su distribución.
5. Atención a la salud:
   Gracias a esta tecnología se pueden estudiar y evaluar los datos de un paciente a tiempo real. Asimismo, se pueden identificar tendencias para desarrollar diagnósticos avanzados y mejorados.
6. Redes de transporte:
   De nuevo, es importante para este sector analizar patrones y tendencias de los usuarios para así hacer las rutas más eficientes e incrementar la rentabilidad.

1.2. TITSA: tratamientos de datos

Cada vez más, y de forma más incipiente, las empresas reconocen que el valor de los datos representan una fuente estratégica de ideas e información para la mayoría de sus líneas de negocios.

En un mundo cada vez más interconectado y con más datos, la gestión de la información se vuelve cada vez más crítica. La empresa debe asegurar no sólo la calidad e integridad de los datos, sino también que los mismos produzcan valor a los distintos niveles ejecutivos.

1.2.1. Modelos predictivos y redes neuronales

TITSA es una empresa que se nutre de unos algoritmos, denominados **modelos predictivos**, que le permiten predecir el comportamiento de sus usuarios y por ende la situación en la que se encontrará la empresa a largo plazo.

Se denomina modelo predictivo al proceso de utilizar el análisis de datos para realizar predicciones teniendo siempre en cuenta el dato histórico, es decir, es muy importante conocer lo ocurrido en años anteriores para así poder comparar los resultados. De la misma manera que es importante el pasado, es muy necesario conocer el presente y saber los datos de lo que está pasando en todo momento con el fin de mejorarlos.

Si estos procesos se construyen adecuadamente junto con técnicas matemáticas, TITSA podrá obetener datos del tipo: cuántos clientes tendrán a final de año, cuántas ventas tendrán al terminar el año o cuántos clientes se pueden perder.

Para cada predicción que se realice en la empresa se tienen en cuenta una gran variedad de datos con el objetivo de tener el menor error posible, entre ellos se encuentran los datos del usuario, como edad y sexo, el día de la semana, la fecha, la hora, la parada de subida y de bajada de cada usuario, el lugar de residencia, el soporte de la tarjeta, la meteorología de la zona, el tipo de día (laboral, festivo,...), los datos proporcionados por Metropolitano de Tenerife y en la situación actual se suman también todos los datos que tengan relación con el COVID-19.

Algunas de las técnicas matemáticas más usadas para complementar estos procesos son los modelos Arima y Sarima, que son modelos dinámicos de series temporales, el modelo lineal, que consta de una variable predictora y otra que actuará como respuesta, y el modelo no lineal, que pretende obtener los valores de los parámetros asociados con la mejor "curva de ajuste".

Dentro de los modelos predictivos nos encontramos con las **redes neuronales**, basadas en el comportamiento del cerebro humano, es decir, persigue conectar un conjunto de unidades que se transmitan señales entre sí.

El objetivo de este algoritmo es entender datos del mundo real (imágenes, texto, voz, etc.), procesarlos y clasificarlos o etiquetarlos. Cierto es, que este es un modelo que necesita un entrenamiento duro para poder realizar todas esas acciones en un futuro sin ayuda alguna.

Al igual que los modelos predictivos, en las redes neuronales es muy importante el dato histórico pero es aún más importante saber lo que ocurre en el día a día. De esta manera se corrige con la realidad de una forma rápida y eficiente.

1.2.2. Visualización de datos

TITSA hace uso de la plataforma **Power BI** para hacer llegar a todos los empleados de la empresa un resumen semanal de la posición en la que se encuentran a nivel empresarial.

Power BI es una colección de servicios software, aplicaciones y conectores que funcionan conjuntamente para convertir orígenes de datos sin relación entre sí en información coherente, interactiva y atractiva visualmente. Tanto si se trata de una sencilla hoja de cálculo de Excel como de una colección de almacenes de datos híbridos locales o basados en la nube, Power BI le permite conectar fácilmente los orígenes de datos, visualizar (o descubrir) lo más importante y compartirlo con quien quiera.

Power BI consta de una aplicación de escritorio de Windows denominada Power BI Desktop, un servicio SaaS (software como servicio) en línea denomi-

nado servicio Power BI, y aplicaciones móviles de Power BI disponibles para teléfonos y tabletas Windows, así como para dispositivos iOS y Android.

Estos tres elementos, Desktop, el servicio y el destinado a dispositivos móviles, están diseñados para permitir a los usuarios crear, compartir y utilizar información empresarial de la forma que les resulte más eficaz para su rol.

El flujo de trabajo habitual de TITSA en Power BI comienza en Power BI Desktop, donde se crea un informe. Luego, ese informe se publica en el servicio Power BI y despueés se comparte para que los usuarios de las aplicaciones de Power BI Mobile puedan usar la información (figura 1.1).

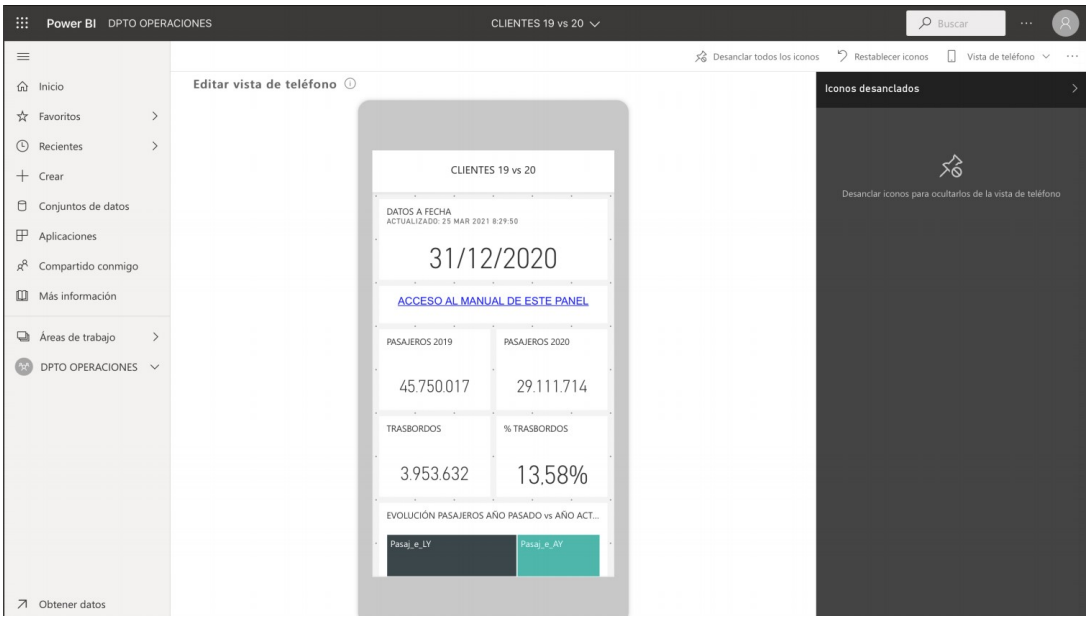


Figura 1.1. Visualización móvil en el servicio en línea Power BI.

Tratamiento de los datos

Aunque anteriormente hemos mencionado el flujo habitual de Power BI, nosotros por el momento hemos dedicado el total de nuestro tiempo a trabajar con la aplicación de escritorio de Windows, Power BI Desktop.

Power BI tiene unos bloques de creación básicos, entre ellos, las visualizaciones, estas pueden ser sencillas, como un único número que representa un aspecto significativo, o visualmente complejas, como un mapa de colores degradados que muestra la opinión con respecto a un determinado problema. La finalidad de un objeto visual es presentar los datos de una manera que ofrezca contexto e información detallada, lo que probablemente resultaría difícil de discernir en una tabla sin formato de números o texto.

Nuestro conjunto de datos está basado en una combinación de tablas de Excel. Cuando se obtienen estos datos, muchas veces no presentan el formato correcto o no están limpios como se desea. Por este motivo, hemos tenido que limpiar o transformar los datos.

Power BI Desktop (figura 1.2) es una herramienta que nos permite conectarnos a los datos y modelarlos para así poder visualizarlos de distintas formas.

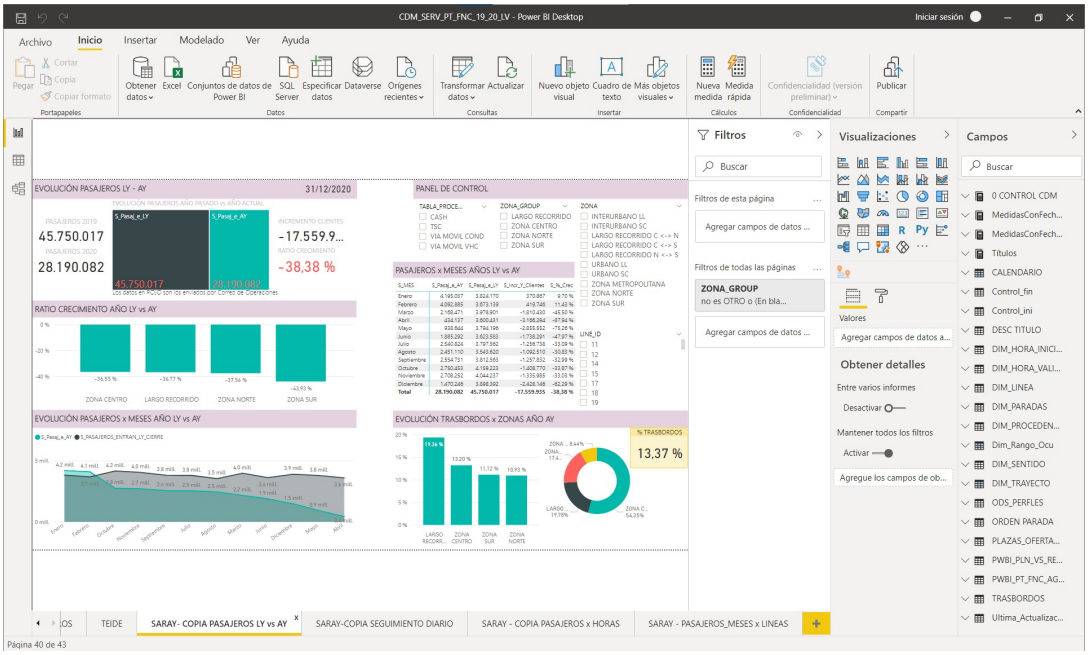


Figura 1.2. Panel de visualizaciones en Power BI desktop.

1.3. Programación lineal

En respuesta a qué es programación lineal (PL) se puede aludir que este es un método matemático para reducir costos o, maximizar ganancias en diferentes áreas de una organización o empresa.

En PL se someten las diferentes variables a una serie de restricciones expresadas a través de un sistema de ecuaciones lineales o desigualdades. La búsqueda de la mejor decisión es guiada a través de un función objetivo que valora cada posible alternativa en relación al criterio de optimización.

Debemos tener en cuenta que la PL está formada por los siguientes elementos fundamentales:

1. Soluciones factibles
   Son aquellas que verifican todas las restricciones del problema.

2. La región factible.
   Es el conjunto de soluciones factibles. En el caso de no encontrarse ninguna, el problema sería infactible.
3. La función objetivo
   Es la función a optimizar, ya sea minimizando o maximizando su resultado.
4. Solución óptima.
   Es una solución factible que da el valor más favorable de la función objetivo.

Programación lineal entera

Los modelos de programación entera son una extensión de los modelos lineales en los que algunas variables toman valores enteros. Este tipo de modelos permite representar sistemas mucho más complejos. A cambio, la resolución de los mismos se complica excesivamente, haciendo así que, problemas con pocas variables pueden ser casi imposibles de resolver.

Atendiendo al tipo de variables, los problemas de PLE se pueden diferenciar entre:

1. Enteros puros: Son aquellos en que todas las variables únicamente pueden tomar valores enteros.
2. Mixtos: Son aquellos en los que hay al mismo tiempo variables continuas y variables que solo pueden tomar valores enteros.
3. Binarios: Las variables sólo pueden tomar los valores cero o uno.

1.3.1. Método del simplex

Teorema 1.1 (Teorema fundamental de Programación Lineal).
Si un problema de PL tiene región factible no vacía, entonces, si existe el óptimo (ya sea máximo o mínimo) de la función objetivo, se encuentra en un punto extremo (o vértice) de la región factible.
Si una función alcanza el valor óptimo en dos vértices consecutivos de la región factible, entonces alcanza también dicho valor óptimo en todos los puntos del segmento que determinan ambos vértices.

El teorema 1.1 afirma que la solución óptima de un programa lineal planteado en su forma estándar, si existe, corresponde siempre a una de las soluciones básicas factibles.

El método del simplex es un medio analítico para tratar con las soluciones de problemas de PL. Es capaz de resolver modelos más complejos que los que podemos llegar a resolver con el método gráfico ya que no tenemos restricción alguna en la cantidad de variables disponibles.

Si deseamos obtener el punto óptimo de un problema de PL, tenemos que obtener todos los vértices de la región factible y el valor de cada uno de ellos

en la función objetivo. En la práctica, el método del simplex es eficiente debido a que el número de vértices enumerados suele ser en media, a lo sumo, tres veces el número de restricciones (Como nos explican Daniel A. Spielman y Shang-Hua Teng en el siguiente estudio: `Smoothed analysis: an attempt to explain the behavior of algorithms in practice`)

El método del simplex nos ayudará a elaborar un criterio que nos permitirá saber si una solución básica factible es óptima o no.

Nos encontramos ante un método iterativo (se puede comprender esto de una mejor manera observando la figura 1.3) que permite ir mejorando la solución en cada paso. Los pasos a seguir son los siguientes:

a. Suponiendo que ya conocemos una solución básica inicial. Si es óptima, hemos terminado.
b. Si no es óptima, pasamos a otra solución (vértice contiguo) tal que el valor de la función objetivo sea mayor (o menor, dependiendo de que busquemos) que en la anterior. Si de nuevo ocurre que no es óptima, repetiremos el paso.
c. Como tenemos un número finito de vértices significa que, si hay solución óptima, la encontraremos en un número finito de iteraciones del método.

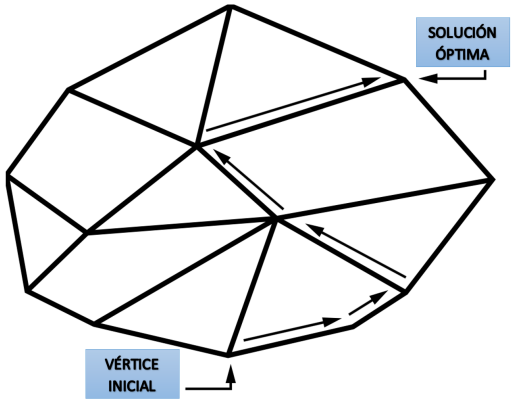


Figura 1.3. Ilustración del Método Simplex.

1.3.2. Entorno de programación

Este estudio se ha programado en lenguaje Python, pues es un lenguaje sencillo de leer y escribir debido a su alta similitud con el lenguaje humano. Además, se trata de un lenguaje multiplataforma de código abierto y, por lo tanto, gratuito, lo que permite desarrollar software sin límites.

Para el desarrollo de nuestro código se han utilizado los siguientes entornos:

- Visual Studio Code (figura 1.4):
  Es el editor de código fuente de Microsoft para Windows, Linux, macOS y Web. Incluye soporte para depuración, resaltado de sintaxis, finalización de código inteligente, fragmentos y refactorización de código.

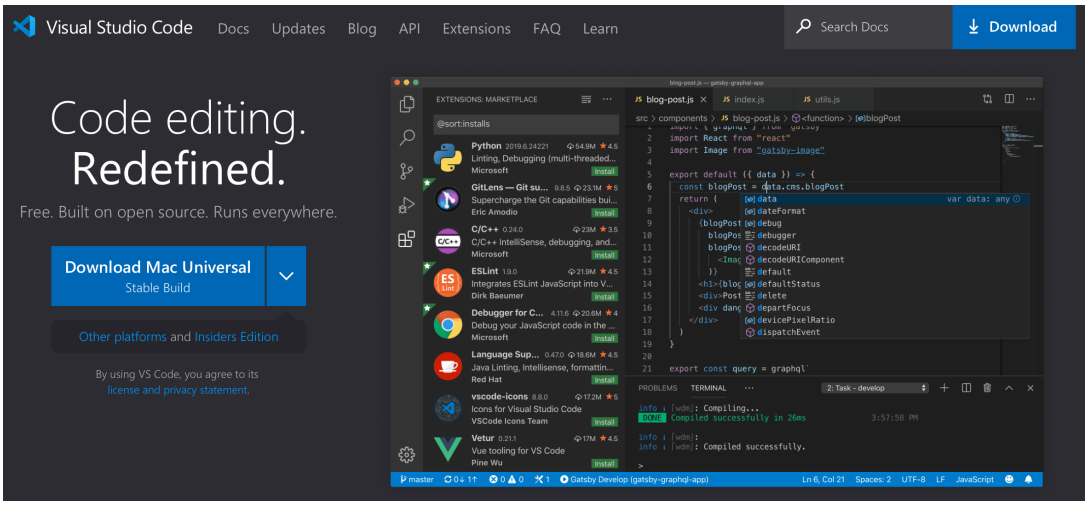


Figura 1.4. Entorno Visual Studio Code.

- Jupyter Notebook (figura 1.5):
  El término "notebook"puede referirse a muchas entidades diferentes, principalmente aplicaciones web Jupyter, servidores web Jupyter Python o formatos de documentos Jupyter. Los documentos de Jupyter Notebook siguen un esquema versionado y contienen una lista ordenada de celdas de entrada/salida, que pueden contener código, texto (usando Markdown), matemáticas, gráficos y texto enriquecido, generalmente terminando con la extensión ".ipynb".

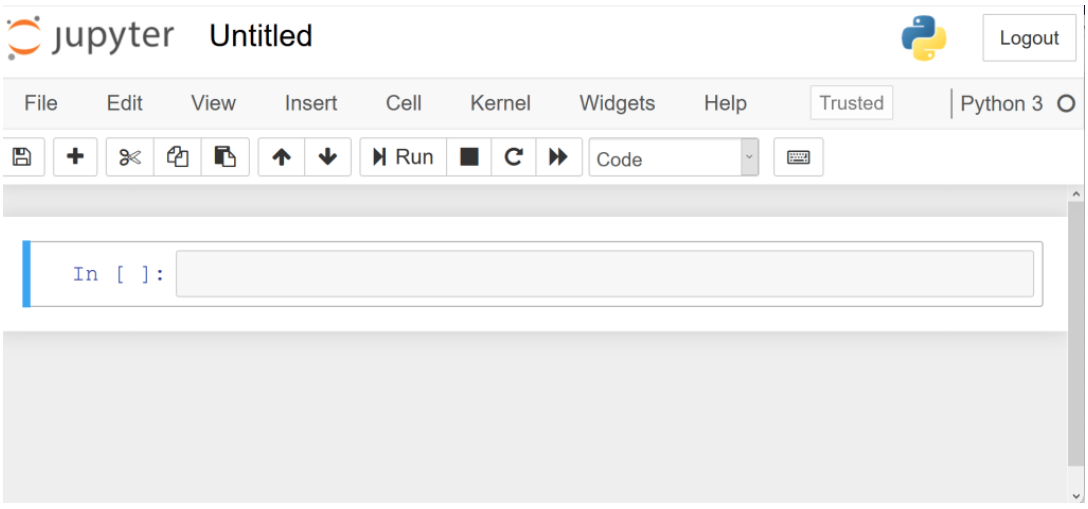


Figura 1.5. Entorno Jupyter Notebook.

Implementación del algoritmo

Para poder implementar el algoritmo en Python nos hemos apoyado en la fundación COIN-OR (del inglés *Computacional Infrastructure for Operations Research*), donde podemos encontrar códigos abiertos al público interesado en la investigación de operaciones. (figura 1.6).

Figura 1.6. COIN-OR Foundation.

Dentro de este recurso nos encontramos con el proyecto Graph, donde se sitúa GiMPy, una librería Python que contiene única y exclusivamente implementaciones Python de diferentes algoritmos de grafos con sus respectivas visualizaciones.

Nos hemos apoyado en el método del algoritmo de flujo máximo (figura 1.7).

coinor.gimpy.examples.maxflow

Small example for demonstrating flow agumenting path (ford-fulkerson) algorithm. This example is taken from youtube lecture series on Advanced Operations Research by Prof. G.Srinivasan. Link to the video is: http://www.youtube.com/watch?v=J0wzih3_5Wo

black: there is no flow on the arc red : the flow equals to the arc capacity green: there is positive flow on the arc, less then capacity.

▼ EXPAND SOURCE CODE

```
    g.add_node(3,pos='"2,4!"')
    g.add_node(2,pos='"2,0!"')
    g.add_node(4,pos='"4,2!"')
    g.add_node(6,pos='"6,4!"')

    g.add_node(5,pos='"6,0!"')
    g.add_node(7,pos='"8,2!"')
    g.add_edge(1, 3, capacity=40, label='40')
    g.add_edge(1, 2, capacity=40, label='40')
    g.add_edge(3, 4, capacity=10, label='10')
    g.add_edge(3, 6, capacity=10, label='10')
    g.add_edge(2, 4, capacity=15, label='15')
    g.add_edge(2, 5, capacity=20, label='20')
    g.add_edge(4, 6, capacity=20, label='20')
    g.add_edge(4, 7, capacity=10, label='10')
    g.add_edge(4, 5, capacity=10, label='10')
    g.add_edge(6, 7, capacity=30, label='30')
    g.add_edge(5, 7, capacity=20, label='20')
    g.set_display_mode('off')

#    g.max_flow_preflowpush(1, 7, algo = 'HighestLabel')
    g.max_flow(1, 7, algo = 'BFS')

    nl = list(int(n) for n in g.get_node_list())
    nl.sort()
    for n in nl:
        for m in nl:
            if g.check_edge(n, m):
                print(n, m, g.get_edge_attr(n, m, 'flow'))
```

Figura 1.7. Parte del ejemplo del algoritmo de flujo máximo.

2

Presentación del problema general y objeto de estudio

Como ya he mencionado en la introducción de este trabajo, el objetivo es mejorar los servicios ofrecidos por TITSA y para ello debemos solucionar determinados inconvenientes que presentan ciertos puntos de acceso o paradas.

Se han detectado incongruencias en diferentes zonas de transporte de la isla. Para resolverlas se requiere de un sistema inteligente que detecte cambios de flujo entre las distintas zonas de transporte de Tenerife y además vea la relación entre los diferentes puntos de acceso a la red y los servicios que se ofrecen entre ellos.

Nuestro primer paso es hacer un estudio a gran escala de todos los problemas que puedan derivar del estudio de los sectores de transporte de la isla con el fin de crear una pequeña guía que contenga una resolución esquematizada que podamos seguir en el futuro en el caso de que nos encontremos ante uno de estos problemas.

Una vez realizado ese primer paso, nos iremos por una de las ramas de dicha guía e intentaremos resolver el problema particular asociado.

2.1. Representación del problema

Imaginemos que tenemos una región sectorizada, dónde para cada sector, conocemos la demanda que recibe así como la que genera hacia otros sectores. Por otro lado, para esa misma región, tenemos una red de nodos (paradas), que son atendidos por una serie de líneas (recorridos de autobuses). Un sector puede tener varios nodos, y para cada uno de ellos conocemos la demanda que genera hacia el resto de los nodos pertenecientes al mismo recorrido; de la misma manera conocemos también la demanda que atrae. Por tanto, se podría tener un sistema que comprobase si se da alguno de los siguientes casos:

1. Caso de un sector A con una demanda significativa de un sector B:
   a) ¿Hay recorridos que pasan por nodos de A y de B?
      1) Sí, y hay pares de nodos (a,b) con alta demanda entre ellos.
         - No se produce ninguna modificación.

   2) Sí, y no hay pares de nodos (a,b) con alta demanda entre ellos.
      - Hay que estudiar si se elimina el nodo y en tal caso estaríamos ante lo que llamaremos **Problema Eliminar Parada** (PEP).
   3) No, y hay pares de nodos (a,b) con alta demanda en ambos, es decir, tenemos un nodo con alta demanda en el sector A, un nodo con alta demanda en el sector B, y ninguna línea o recorrido que los une.
      - Estudiar si se puede atender la demanda modificando una línea existente: **Problema Modificar Recorrido** (PMR).
      - En caso contrario, se justifica la creación de un nuevo recorrido.
   4) No, y no hay pares de nodos (a,b) con alta demanda en ambos, es decir, o bien tenemos un nodo con alta demanda en el sector A y ningún nodo con alta demanda en el sector B, o viceversa, y ninguna línea o recorrido que los una.
      - Problema añadir parada y modificar recorrido (PAP y PMR, respectivamente).
      - En caso contrario, se justifica la creación de un nuevo recorrido.
2. Caso de un sector A que no tiene una demanda significativa de un sector B
   a) ¿Hay recorridos que pasan por nodos de A y de B?
      1) Sí, y hay pares de nodos (a,b) con baja demanda entre ellos.
         - Estudiar si se elimina esa parte del recorrido.

2.1.1. Caso particular

Imaginemos una zona A y B que presentan una alta demanda, y se detecta una parada en la zona B, denominada P_b, con baja demanda, y que no tiene un recorrido que pase por ninguna parada de A, con alta demanda. En cambio, la parada P_b tiene una alta demanda teórica de las paradas de alta demanda del sector A. Por tanto, el problema a resolver es un PMR, es decir, para cada recorrido que pasa por un nodo de alta demanda en A, modificar su recorrido para que pase por el nodo P_b de forma que se cumplan las restricciones siguientes planteadas:

- La capacidad del vehículo que hace el recorrido debe ser capaz de absorber la demanda.
- La penalización en tiempo de cada pasajero debe estar dentro de los rangos permitidos.

Por otro lado, si un recorrido no cumple la condición anterior, es decir, resulta que se queda una parada P_a de alta demanda en el sector A sin conectar con la parada P_b, se genera otro problema a resolver, que podría ser:

1. Buscar algún recorrido de los que pasan por A, y estudiar si este podría pasar por las parada P_a y P_b cumpliendo las restricciones del problema.
2. En caso negativo, habría que estudiar si añadir un recorrido nuevo que uniese la parada P_a con la parada P_b.

Objeto de estudio: Polígono de Güímar

En el polígono de Güímar existe una parada muy cercana a 3 grandes supermercados. Se observa que, del sector del pueblo de Güímar, mucha gente demanda ir a esa parada, pero no existe ningún recorrido entre paradas de alta demanda el pueblo de Güímar y esa parada. Por tanto, estamos en un caso de estudiar si, modificando los recorridos de las líneas que pasan por nodos de alta demanda en el pueblo de Güímar y haciéndolos pasar por esa parada, se cumplen las restricciones del problema.

En los siguientes gráficos podemos apreciar como las paradas que se encuentran actualmente en el Polígono de Güímar no son frecuentadas por muchos usuarios. Teniendo un total de 59 clientes de entrada y salida a ellas.

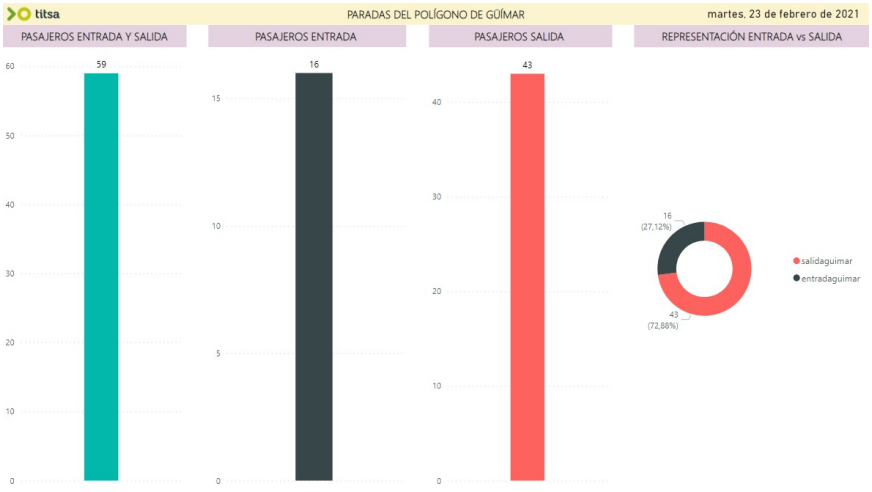


Figura 2.1. Pasajeros de entrada y salida en las paradas del Polígono de Güímar.

Las líneas que estudiaremos son 120, 121, 122, 124, 111 y 711, pues son las que realizan el recorrido que podemos observar en la figura 2.2.

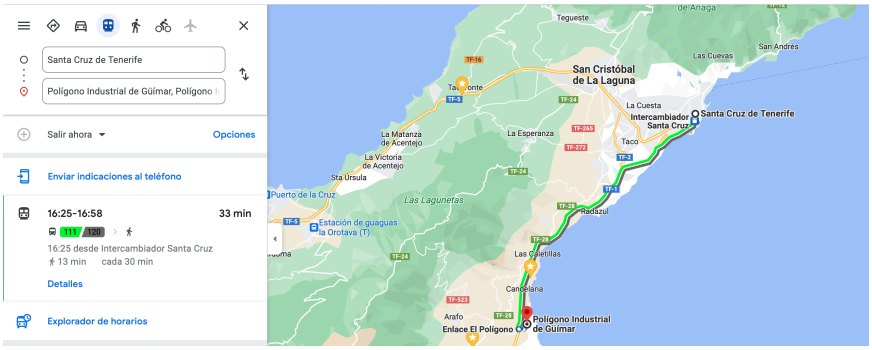


Figura 2.2. Recorrido Santa Cruz - Polígono de Güímar.

Tendremos en cuenta solamente los trayectos desde Santa Cruz hacia el Sur de Tenerife, por lo que se nos quedarían un total de 13 recorridos (como podemos ver en la figura 2.3).

	CÓDIGO LÍNEA	CÓDIGO TRAYECTO	DESCRIPCIÓN TRAYECTO
1	111	11	SANTA CRUZ - AEROPUERTO SUR - ESTACIÓN COSTA ADEJE
2	120	11	SANTA CRUZ - CANDELARIA - GÜIMAR
3	120	21	SANTA CRUZ - PUERTITO DE GÜIMAR - GÜIMAR
4	120	61	GÜIMAR - LAS ARENAS - PUERTITO DE GÜIMAR
5	121	11	SANTA CRUZ - ARAFO - GÜIMAR
6	121	41	SANTA CRUZ - ARAFO - GÜIMAR
7	122	11	SANTA CRUZ-CALETILLAS-CANDELARIA-POLÍGONO GÜIMAR
8	122	21	SANTA CRUZ - CALETILLAS - CANDELARIA
9	124	11	SANTA CRUZ - CANDELARIA - GÜIMAR
10	711	11	INTERCAMBIADOR-AEROPUERTO SUR-ESTACIÓN COSTA ADEJE
11	711	21	INTERCAMBIADOR - AEROPUERTO SUR -COSTA ADEJE
12	711	31	INTERCAMBIADOR - AEROPUERTO SUR - POR CALETILLAS
13	120	41	SANTA CRUZ - CANDELARIA - GÜIMAR

Figura 2.3. Código de líneas, trayecto y descripción del recorrido.

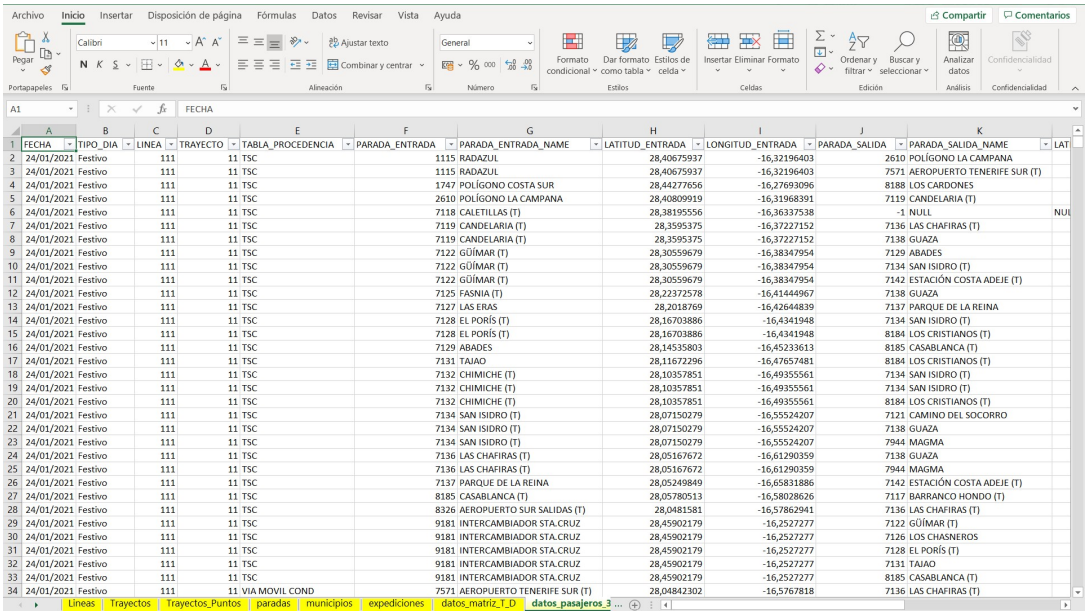
3

Resolución del problema particular

Consideraremos de ahora en adelante que la localización de la parada nueva añadidas se encuentra en el foco de demanda del Polígono de Güímar.

3.1. Datos

TITSA me ha facilitado una gran cantidad de datos en forma de hojas de cálculo de Excel (figura 3.1) . Desde una matriz de tiempo y distancia entre los puntos de acceso de determinadas líneas hasta los datos de los pasajeros recogidos durante el mes de febrero de 2021. Añadiendo también datos estructurales del SAE (Sistema de Ayuda a la Explotación de autobuses) de líneas, puntos de acceso, municipios, trayectos y expediciones planificadas para estas líneas.



FECHA	TIPO_DIA	LINEA	TRAYECTO	TABLA_PROCEDENCIA	PARADA_ENTRADA	PARADA_ENTRADA_NAME	LATITUD_ENTRADA	LONGITUD_ENTRADA	PARADA_SALIDA	PARADA_SALIDA_NAME
24/01/2021	Festivo	111	11	TSC	1115	RADAZUL	28,40675937	-16,32196403	2610	POLÍGONO LA CAMPANA
24/01/2021	Festivo	111	11	TSC	1115	RADAZUL	28,40675937	-16,32196403	7571	AEROPUERTO TENERIFE SUR (T)
24/01/2021	Festivo	111	11	TSC	1747	POLÍGONO COSTA SUR	28,44277656	-16,27693096	8188	LOS CARDONES
24/01/2021	Festivo	111	11	TSC	2610	POLÍGONO LA CAMPANA	28,40809919	-16,31968391	7119	CANDELARIA (T)
24/01/2021	Festivo	111	11	TSC	7118	CALETILLAS (T)	28,38195556	-16,36337538	-1	NULL
24/01/2021	Festivo	111	11	TSC	7119	CANDELARIA (T)	28,3595375	-16,37227152	7136	LAS CHAFIRAS (T)
24/01/2021	Festivo	111	11	TSC	7119	CANDELARIA (T)	28,3595375	-16,37227152	7138	GUAZA
24/01/2021	Festivo	111	11	TSC	7122	GÜÍMAR (T)	28,30559679	-16,38347954	7129	ABADES
24/01/2021	Festivo	111	11	TSC	7122	GÜÍMAR (T)	28,30559679	-16,38347954	7134	SAN ISIDRO (T)
24/01/2021	Festivo	111	11	TSC	7122	GÜÍMAR (T)	28,30559679	-16,38347954	7142	ESTACIÓN COSTA ADEJE (T)
24/01/2021	Festivo	111	11	TSC	7125	FASNIA (T)	28,22372578	-16,41444967	7138	GUAZA
24/01/2021	Festivo	111	11	TSC	7127	LAS ERAS	28,2018769	-16,42644839	7137	PARQUE DE LA REINA
24/01/2021	Festivo	111	11	TSC	7128	EL PORÍS (T)	28,16703886	-16,4341948	7134	SAN ISIDRO (T)
24/01/2021	Festivo	111	11	TSC	7128	EL PORÍS (T)	28,16703886	-16,4341948	8184	LOS CRISTIANOS (T)
24/01/2021	Festivo	111	11	TSC	7129	ABADES	28,14535803	-16,45233613	8185	CASABLANCA (T)
24/01/2021	Festivo	111	11	TSC	7131	TAJAO	28,11672296	-16,47657481	8184	LOS CRISTIANOS (T)
24/01/2021	Festivo	111	11	TSC	7132	CHIMICHE (T)	28,10357851	-16,49355561	7134	SAN ISIDRO (T)
24/01/2021	Festivo	111	11	TSC	7132	CHIMICHE (T)	28,10357851	-16,49355561	7134	SAN ISIDRO (T)
24/01/2021	Festivo	111	11	TSC	7132	CHIMICHE (T)	28,10357851	-16,49355561	8184	LOS CRISTIANOS (T)
24/01/2021	Festivo	111	11	TSC	7134	SAN ISIDRO (T)	28,07150279	-16,55524207	7121	CAMINO DEL SOCORRO
24/01/2021	Festivo	111	11	TSC	7134	SAN ISIDRO (T)	28,07150279	-16,55524207	7138	GUAZA
24/01/2021	Festivo	111	11	TSC	7134	SAN ISIDRO (T)	28,07150279	-16,55524207	7944	MAGMA
24/01/2021	Festivo	111	11	TSC	7136	LAS CHAFIRAS (T)	28,05167672	-16,61290359	7138	GUAZA
24/01/2021	Festivo	111	11	TSC	7136	LAS CHAFIRAS (T)	28,05167672	-16,61290359	7944	MAGMA
24/01/2021	Festivo	111	11	TSC	7137	PARQUE DE LA REINA	28,05249849	-16,65831886	7142	ESTACIÓN COSTA ADEJE (T)
24/01/2021	Festivo	111	11	TSC	8185	CASABLANCA (T)	28,05780513	-16,58028626	7117	BARRANCO HONDO (T)
24/01/2021	Festivo	111	11	TSC	8326	AEROPUERTO SUR SALIDAS (T)	28,0481581	-16,57862941	7136	LAS CHAFIRAS (T)
24/01/2021	Festivo	111	11	TSC	9181	INTERCAMBIADOR STA.CRUZ	28,45902179	-16,2527277	7122	GÜÍMAR (T)
24/01/2021	Festivo	111	11	TSC	9181	INTERCAMBIADOR STA.CRUZ	28,45902179	-16,2527277	7126	LOS CHASNEROS
24/01/2021	Festivo	111	11	TSC	9181	INTERCAMBIADOR STA.CRUZ	28,45902179	-16,2527277	7128	EL PORÍS (T)
24/01/2021	Festivo	111	11	TSC	9181	INTERCAMBIADOR STA.CRUZ	28,45902179	-16,2527277	7131	TAJAO
24/01/2021	Festivo	111	11	TSC	9181	INTERCAMBIADOR STA.CRUZ	28,45902179	-16,2527277	8185	CASABLANCA (T)
24/01/2021	Festivo	111	11	VIA MOVIL COND	7571	AEROPUERTO TENERIFE SUR (T)	28,04842302	-16,5767818	7136	LAS CHAFIRAS (T)

Figura 3.1. Datos de pasajeros, tiempo, distancias, etc. Recogidos en el mes de febrero de 2021 por TITSA.

¿Qué datos necesitaremos y utlizaremos?

Trabajaremos solamente con los trayectos de las líneas 120, 121, 122, 124, 111 y 711 con sentido desde Santa Cruz de Tenerife hacia el sur de la misma, por lo que podremos dividir el proyecto según la ubicación de las paradas actuales en:

1. Paradas anteriores.
2. Paradas posteriores.

De esta manera, los datos que extraeremos de TITSA son:

- Conjunto de paradas localizadas entre Santa Cruz y la parada nueva: P^a, $|P^a| = n_a$.
- Conjunto de paradas localizadas entre la parada nueva y el sur de la isla: P^b, $|P^b| = n_b$.
- Conjunto total de paradas: $P = P^a + P^b$.
- Conjunto total de líneas estudiadas: $L = L^a + L^b$, $|L| = 2m$.
  Considerando como una única línea a cada guagua con sus trayectos correspondientes, es decir, habrá tantas líneas como recorridos.
- Capacidad máxima adicional de pasajeros que puede recoger cada línea en las paradas anteriores a la parada nueva: c^a.
- Capacidad máxima adicional de pasajeros que puede recoger cada línea en las paradas posteriores a la parada nueva: c^b.
- Incremento del tiempo total (en minutos)de cada línea al incluir en su recorrido la nueva parada: t_i.
- Demandas de los pasajeros en las paradas anteriores a la nueva: d_j^a, $\forall j \in P^a$.
  Con el fin de ir a dicha parada.
- Demanda de los pasajeros en las paradas posteriores a la nueva: d_j^b, $\forall j \in P^b$.
  Con el fin de subirse en la parada nueva y bajarse en una de las posteriores.
- Conección entre líneas y paradas actuales.
  Paradas por las que pasa cada línea.

3.1.1. Modelización de los datos

En las siguientes páginas veremos dividido en tres partes el archivo CSV que hemos creado a partir de la hoja de cálculo de Excel vista anteriormente (figura 3.1) para facilitar el tratamiento de los datos, además, hemos añadido una explicación de como hemos estructurado este CSV.

- 1ª parte del archivo:

	46	67	13								
1	111	11	4	37	27						
2	120	11	4	10	18						
12	711	12	3	31	32						
13	711	31	4	57	58						

Figura 3.2. Archivo.csv (parte 1).

Desde la fila 0 hasta la 13 del archivo CSV (figura 3.2), la lectura de los valores es la siguiente (figura 3.3), donde los números del 1 al 13 que vemos en la columna coloreada de rojo hacen referencia al número que le hemos asignado a las líneas en el código Python (ver apéndice).

	Número de paradas anteriores	Número de paradas posteriores	Número total de líneas		
Línea **1**	Código línea	Código trayecto	Capacidad de la línea antes de la parada	Capacidad de la línea después de la parada	Incremento del tiempo total del recorrido
Línea **2**	Código línea	Código trayecto	Capacidad de la línea antes de la parada	Capacidad de la línea después de la parada	Incremento del tiempo total del recorrido
Línea **13**	Código línea	Código trayecto	Capacidad de la línea antes de la parada	Capacidad de la línea después de la parada	Incremento del tiempo total del recorrido

Figura 3.3. Estructura del archivo.csv (parte 1).

- 2ª parte del archivo:

1	1115	1									
2	1125	1									
44	9181	10									
45	9413	2									
46	9449	3									
47	2244	1									
112	8184	1									
113	8185	1									

Figura 3.4. Archivo.csv (parte 2).

Desde la fila 14 hasta la 127 del archivo CSV (figura 3.4), la lectura de los valores es la siguiente (figura 3.5), donde los números del 1 al 113 (columna azul) hacen referencia al número que le hemos asignado a las paradas en el código Python (ver apéndice).

Parada **1**	Código parada (localizada antes de la nueva)	Demanda parada antes			
Parada **2**	Código parada (localizada antes de la nueva)	Demanda parada antes			
Parada **46**	Código parada (localizada antes de la nueva)	Demanda parada antes			
Parada **47**	Código parada (localizada después de la nueva)	Demanda parada después			
Parada **48**	Código parada (localizada después de la nueva)	Demanda parada después			
Parada **113**	Código parada (localizada después de la nueva)	Demanda parada después			

Figura 3.5. Estructura del archivo.csv (parte 2).

- 3ª parte del archivo:

1	1	1									
1	2	1									
1	113	0									
2	1	1									
2	2	1									
11	113	1									
12	1	1									
12	2	0									
12	113	1									
13	1	0									
13	2	1									
13	113	1									

Figura 3.6. Archivo.csv (parte 3).

Desde la fila 127 hasta el final del archivo CSV (figura 3.6), la lectura de los valores es la siguiente (figura 3.7):

Línea **1**	Parada 1	1 -> CONECTADAS			
Línea **1**	Parada 2	0 -> NO CONECTADAS			
Línea **1**	Parada 113	1 -> CONECTADAS			
Línea **2**	Parada 1	1 -> CONECTADAS			
Línea **2**	Parada 2	0 -> NO CONECTADAS			
Línea **12**	Parada 113	1 -> CONECTADAS			
Línea **13**	Parada 1	1 -> CONECTADAS			
Línea **13**	Parada 2	0 -> NO CONECTADAS			
Línea **13**	Parada 113	0 -> NO CONECTADAS			

Figura 3.7. Estructura del archivo.csv (parte 3).

3.2. Grafo del problema

En principio se consideran dos problemas de factibilidad separados que finalmente se unirán en un único grafo.

1. Grafo correspondiente a los valores anteriores a la ubicación del punto de acceso a estudiar.
   - Añadimos un nodo fuente S_1.
   - Añadimos un nodo sumidero T.
   - Existe un arco entre L_i y $P_j^a \Leftrightarrow$ la correspondiente línea pasa por la parada.

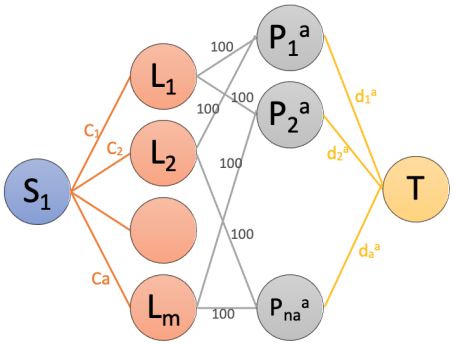


Figura 3.8. Grafo anterior al punto de acceso.

2. Grafo correspondiente a los valores posteriores a la ubicación del punto de acceso a estudiar.
   - Añadimos un nodo fuente S_2.
   - Añadimos un nodo sumidero T.
   - Existe un arco entre L_i y $P_j^b \Leftrightarrow$ la correspondiente línea pasa por la parada.

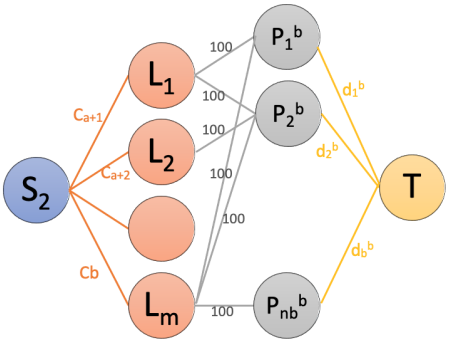


Figura 3.9. Grafo posterior al punto de acceso.

3.3. Modelo del problema

Se pretende desarrollar un modelo matemático de optimización cuya función objetivo sea minimizar el máximo incremento de tiempo incorporado en cada línea al añadir el nuevo punto de acceso. Esto desemboca en dos problemas de flujo de mínimo coste limitados por una única función objetivo con variables $z_i \in \{0, 1\}$ de activación de la correspondiente línea.

De esta manera, el problema final resultante es un problema de Programación Binivel Lineal (PBL) entero:

$$\text{mín}\, Y \tag{3.1}$$

$$\textit{sujeto a}: \tag{3.2}$$

$$\text{mín} \max_{i=1}^{m} \{t_i z_i\} \tag{3.3}$$

$$\textit{sujeto a}: \tag{3.4}$$

$$\sum_{\{i:(i,j)\in(L^a,P^a)\}} x_{i,j} = d_j^a \quad \forall j \in P^a \tag{3.5}$$

$$\sum_{\left\{i:(i,j)\in(L^b,P^b)\right\}} x_{i,j} = d_j^b \quad \forall j \in P^b \tag{3.6}$$

$$\sum_{\{j:(i,j)\in(L^a,P^a)\}} x_{i,j} - x_{S_1,i} = 0 \quad \forall i \in L^a \tag{3.7}$$

$$\sum_{\left\{j:(i,j)\in(L^b,P^b)\right\}} x_{i,j} - x_{S_2,i} = 0 \quad \forall i \in L^b \tag{3.8}$$

$$\sum_{i\in L^a} x_{S_1,i} = d^a \tag{3.9}$$

$$\sum_{i\in L^b} x_{S_2,i} = d^b \tag{3.10}$$

$$x_{S_1,i} \leq \text{mín}\,\{Y, z_i c_i^a\} \quad \forall i \in L^a \tag{3.11}$$

$$x_{S_2,i} \leq \text{mín}\left\{Y, z_i c_i^b\right\} \quad \forall i \in L^b \tag{3.12}$$

$$x_{ij} \geq 0 \quad \forall (i,j) \in A \tag{3.13}$$

$$z_i \in \{0,\ 1\} \quad \forall i \in \{1, ...m\} \tag{3.14}$$

$$Y \in \mathbb{Z} \tag{3.15}$$

La variable Y representa la cantidad de pasajeros en la que se pueden incrementar las líneas para atender las demandas de la nueva parada. De manera que, si obtenemos valores altos de Y significará que se repartirán los pasajeros en menos líneas mientras que, si la cantidad obtenida de Y es más baja se tendrá que hacer un reparto de los pasajeros entre una cantidad mayor de líneas. La segunda función objetivo es no lineal, pero la podemos expresar de una manera lineal simplemente introduciendo la variable a minimizar T, siendo esta un límite superior de los incrementos de tiempo de las líneas activadas de la siguiente manera:

$$\text{mín} \quad c(Y, z) = (Y, T)$$

$$\begin{array}{c} \textit{sujeto a}: \\ (3.3)\text{-}(3.15) \end{array}$$

$$t_i z_i \leq T \quad \forall i \in \{1, ...m\} \tag{3.16}$$

Denotamos el problema anterior como $P(Y, T)$ y definimos los siguientes conjuntos ordenados:

1. La lista de todas las diferentes capacidades:
   $SC = \bigcup_{i=1}^{m} \{c_i^a\} \cup \bigcup_{i=1}^{m} \left\{c_i^b\right\} \cup \{1\} = \{c_1, ..., c_h\}$, incluyendo el valor 1 (aunque 1 no sea una capacidad inicial).El cardinal de SC es $sc \leq 2m + 1$.
2. La lista de los diferentes incrementos de tiempo:
   $ST = \bigcup_{i=1}^{m} \{t_i\}$. Claramente, el cardinal de ST es $st \leq m$.

Debemos tener en cuenta que los posibles valores que podemos obtener de la variable Y son los valores enteros de $[\, c_1, c_{sc}]$.

Introducimos el siguiente $P_{\bar{T}}(Y, T)$ para ayudarnos con la resolución del problema anterior $P(Y, T)$:

$$P_{\bar{T}}(Y, T) = \text{lex mín} \quad (Y, T)$$

$$\begin{array}{c} \textit{sujeto a}: \\ (3.3)\text{-}(3.16) \end{array}$$

$$z_i = 0 \quad \forall i \in \{1, ...m\} : t_i > \bar{T} \tag{3.17}$$

$$T \leq \bar{T} \tag{3.18}$$

3.4. Algoritmos

El criterio en el problema $P_{\bar{T}}(Y,T)$ es asegurarse de que la solución óptima se corresponde con una solución del problema $P(Y,T)$. Es por eso que necesitamos cambiar en el algoritmo principal el parámetro $\bar{Y}$ adecuadamente, como podemos ver en el siguiente pseudocódigo (1):

Algorithm 1 Algoritmo $P(Y,T)$

Require: Problema de añadir parada.
Ensure: Una solución factible (Y,z,x) por cada solución obtenida.
1: Construimos el grafo $G=(V,A)$;
2: Construimos las listas SU y ST;
3: $\bar{T}=t_{st}$;
4: Resolvemos $P_{\bar{T}}(Y,T)$ y obtenemos la función objetivo $(\acute{Y},\acute{T})$ y, si existe, $(\acute{Y},\acute{z},\acute{x})$ la solución factible;

Asumiendo que tenemos un oráculo que dado $\bar{T}$, un valor para T, devuelve una solución óptima para el problema $P_{\bar{T}}(Y,T)$, entonces el algoritmo(1) es correcto. Este devuelve la solución factible obtenida al resolver $P_{t_{st}}(Y,T)$, considerando todas las líneas posibles. La solución es el mínimo valor para Y, denotado por Y' en el rango $[\,c_1,c_{sc}]$.

3.4.1. Resolución del problema $P_{\bar{T}}(Y,T)$

Dado el grafo $G=(V,A)$ que solo contiene las líneas cuyos incrementos de tiempo son menores o iguales a $\bar{T}$, el problema $P_{\bar{T}}(Y,T)$ debe ser resuelto. El principal objetivo a considerar es la minimización de Y, es decir, minimizar el mayor valor de flujo entre los arcos(S_1,i) con $i\in L^a$ y (S_2,i) con $i\in L^b$ del grafo G. Para eso, buscamos este valor mínimo, denotado por Y', por medio de una búsqueda dicotómica entre los posibles valores en $[\,c_1,c_{sc}]$ considerando todas las líneas en G. Básicamente, buscamos el valor mínimo en este intervalo tal que alcancemos la factibilidad de las restricciones, sustituyéndolas por:

$$x_{S_1,i}\leq \text{mín}\left\{\bar{Y},c_i^a\right\}\quad \forall i\in L^a\cap V$$
$$x_{S_2,i}\leq \text{mín}\left\{\bar{Y},c_i^b\right\}\quad \forall i\in L^b\cap V$$

Por lo tanto, el problema de factibilidad se corresponde con un problema de flujo máximo (MFP). Por ello, agregamos al grafo un súper nodo fuente S conectando con los nodos fuente S_1 y S_2 por los arcos (S, S_1) y (S, S_2) con capacidades igual a las sumas de demandas d^a y d^b, respectivamente. Agregamos un nodo sumidero T, los arcos (j, T) para todos los $j \in P^a \cup P^b$ con capacidades d_j^a y d_j^b, respectivamente. La primera búsqueda dicotómica del valor de $\acute{Y}$ aparece en el algoritmo 2. En las líneas 5-17 se determina el menor valor $\acute{Y}$ que coincide con un elemento en SC donde se alcanza la factibilidad. De lo contrario, el problema $P_{\bar{T}}(Y, T)$ y $\acute{Y}$ es igual a $+\infty$. El valor factible más pequeño corresponde al valor $c_{posLeft}$. Por lo tanto, ahora el algoritmo busca el valor óptimo (siempre que exista) en el intervalo de valores enteros $[\, c_{posRight} + 1, c_{posLeft} - 1]$ (Hecho en las líneas 18-31). A estas alturas, el valor óptimo $\acute{Y}$ es ya conocido. Después, el algoritmo determina una solución factible y óptima con los incrementos de tiempo más pequeños. Para eso, todas las capacidades están incluidas en mín $\left\{\bar{Y}, c_i^a\right\}$ para todos los arcos (S_1, i) y en mín $\left\{\bar{Y}, c_i^b\right\}$ para los arcos (S_2, i) cuyos incrementos de tiempo son menores o iguales a t_{mitad} y cero en caso contrario. Este segundo paso o segunda búsqueda dicotómica se realiza en las líneas de la 32 a la 49.

Algorithm 2 Algoritmo $P_{\bar{T}}(Y, T)$

Require: El grafo $G(V, A)$, los conjuntos SC, ST y $\bar{T}$.
Ensure: El par de valores $(\acute{Y}, \acute{T})$ y la solución $(\acute{Y}, \acute{z}, \acute{x})$ donde haya factibilidad

Añadimos el super nodo fuente S y el nodo sumidero T a $G(V, A)$;
Añadimos los arcos (S, S_1) y (S, S_2) con las demandas d_a y d_b;
Añadimos los arcos $(j, T), \forall j \in P^a$ con las demandas d_j^a; Añadimos los arcos $(j, T), \forall j \in P^b$ con las demandas d_j^b;
**** Primera búsqueda dicotómica para determinar $\acute{Y}$ ****
$posRight = sc$; $posLeft = 1$;$\acute{Y} = +\infty$;
while $(posRight \geq posLeft)$ **do**
    $posMiddle = \lfloor (posRight + posLeft)/2 \rfloor$
    Definimos las capacidades de los arcos $(S^1, i) \quad \forall i \in L^a \cap V$ como mín $\{c_{posMiddle}, c_i^a\}$;
    Definimos las capacidades de los arcos $(S^2, i) \quad \forall i \in L^b \cap V$ como mín $\{c_{posMiddle}, c_i^b\}$;
    Calculamos el valor del flujo máximo v desde S hasta T;
    if $(v \geq d^a + d^b)$ **then**
        $\acute{Y} = c_{posMiddle}$;
        $posRight = posMiddle - 1$;
    else
        $posLeft = posMiddle + 1$;
    end if
end while
if $((\acute{Y} < +\infty)$ *y* $(posLeft > 1))$ **then**
    $posRight = c_{posLeft} - 1$; $posLeft = c_{posRight} - 1$;
    while $(posRight \geq posLeft)$ **do**
        $posMiddle = \lfloor (posRight + posLeft)/2 \rfloor$
        Definimos las capacidades de los arcos $(S^1, i) \quad \forall i \in L^a \cap V$ como mín $\{posMiddle, c_i^a\}$;
        Definimos las capacidades de los arcos $(S^2, i) \quad \forall i \in L^b \cap V$ como mín $\{posMiddle, c_i^b\}$;
        Calculamos el valor de flujo máximo v desde S hasta T;
        if $(v \geq d^a + d^b)$ **then**
            $\acute{Y} = posMiddle$;
            $posRight = posMiddle - 1$;
        else
            $posLeft = posMiddle + 1$;
        end if
    end while
    **** Segunda búsqueda dicotómica para determinar $\acute{T}$ conociendo $\acute{Y}$ ****
    $posRight = st$; $posLeft = 1$;$\acute{T} = \bar{T}$;
    while $(posRight \geq posLeft)$ **do**
        $posMiddle = \lfloor (posRight + posLeft)/2 \rfloor$
        for all $(i \in L^a \cap V)$ **do**
            if $(t_i \leq tposMiddle)$ **then**
                Definimos las capacidades de los arcos (S^1, i) como mín $\left\{\acute{Y}, c_i^a\right\}$;
            else
                Definimos las capacidades de los arcos (S^1, i) como 0;
            end if
        end for
        for all $(i \in L^b \cap V)$ **do**
            if $(t_i \leq t_{posMiddle})$ **then**
                Definimos las capacidades de los arcos (S^2, i) como mín $\left\{\acute{Y}, c_i^b\right\}$;
            else
                Definimos las capacidades de los arcos (S^2, i) como 0;
            end if
        end for
        Calculamos el valor de flujo máximo v desde S hasta T;
        if $(v \geq d^a + d^b)$ **then**
            $\acute{T} = t_{posMiddle}$;
            $posRight = posMiddle - 1$;
        else
            $posLeft = posMiddle + 1$;
        end if
    end while
end if

Denotamos $C = \text{máx}_{i=\{2,\ldots,sc\}} \{c_i - c_{i-1}\}$, luego podemos establecer el siguiente resultado:

Teorema 3.1. *El problema $P(Y,T)$ puede ser resuelto en tiempo polinómico.*

Demostración. El algoritmo $P(Y,T)$ (1) llama 1 vez al algoritmo $P_{\bar{T}}(Y,T)$ (2). En el peor de los casos, el algoritmo (2) hace $\log(2m+1) + \log C + \log m = \log m^2 C$ iteraciones. En cada iteración se resuelve un MFP. Todo junto indica que el algoritmo $P(Y,T)$ se ejecuta en $O((|A||V|) \log m^2 C)$ veces.

3.5. Resolución con Python

El grafo dirigido resultante de unir los dos grafos vistos anteriormente (figuras 3.8 y 3.9) es el que podemos observar a continuación. Hemos añadido un súper nodo fuente S y utilizado el mismo nodo sumidero T.

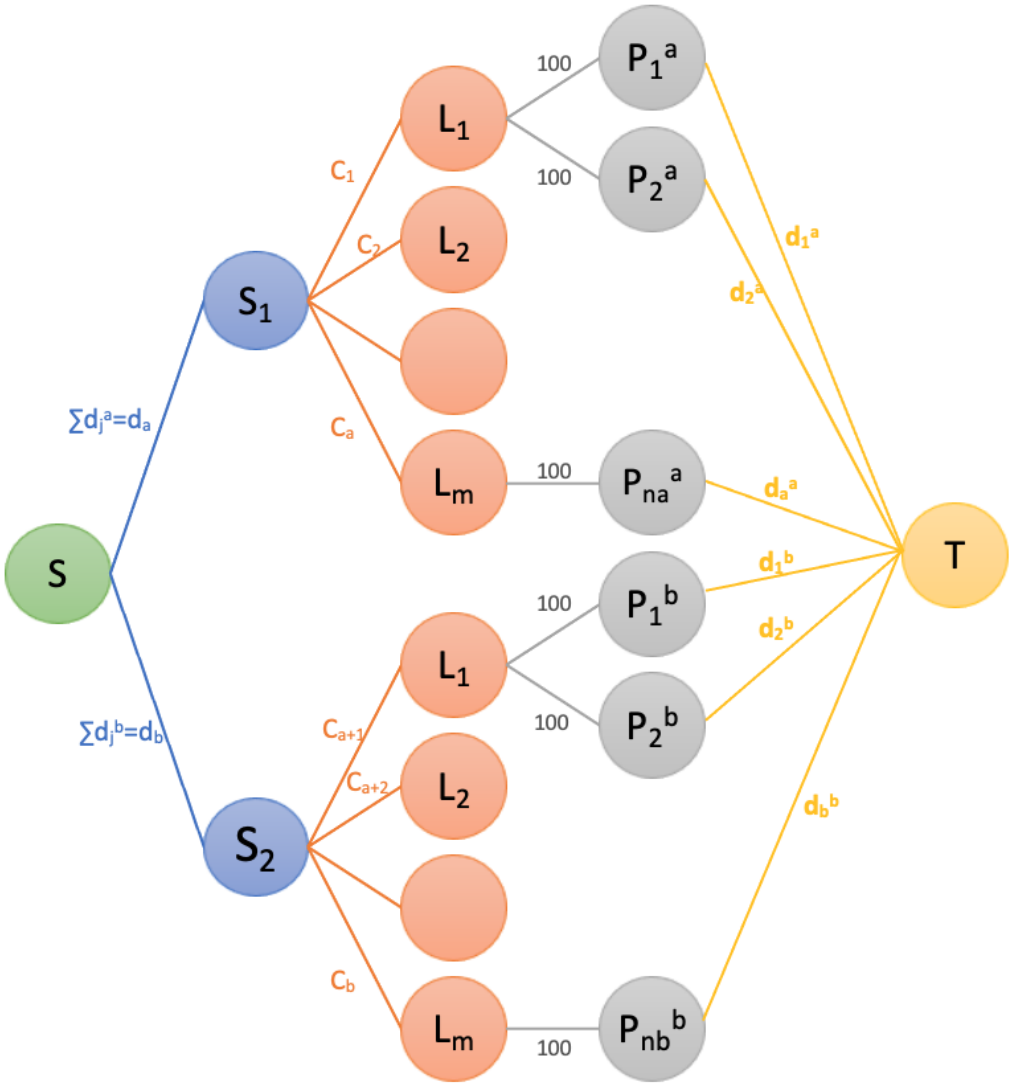


Figura 3.10. Grafo completo.

Veamos ahora paso por paso como hemos ido creando el grafo y por ende como hemos creado el algoritmo (utilizando lenguaje Python) que podremos ver en el apéndice final de este proyecto.

3.5.1. Creación del grafo - Fase 1

1. Añadimos los nodos fuente S, S_1, S_2 y el nodo sumidero T a mano.
2. Para cada línea creamos un nodo con sus correspondientes arcos desde S_1 o S_2.
3. Para cada parada situada antes de la parada nueva (P^a) creamos un nodo y un arco dirigido hacia el nodo T con su correspondiente demanda (d_j^a $\forall j \in P^a$).
4. Para cada parada situada después de la parada nueva (P^b) creamos un nodo y un arco dirigido hacia el nodo T con su correspondiente demanda (d_j^b $\forall j \in P^b$).
   En la figura 3.11 podemos ver los nodos que tenemos por el momento y como los hemos llamado puesto que Python nos permite poner unas etiquetas específicas.

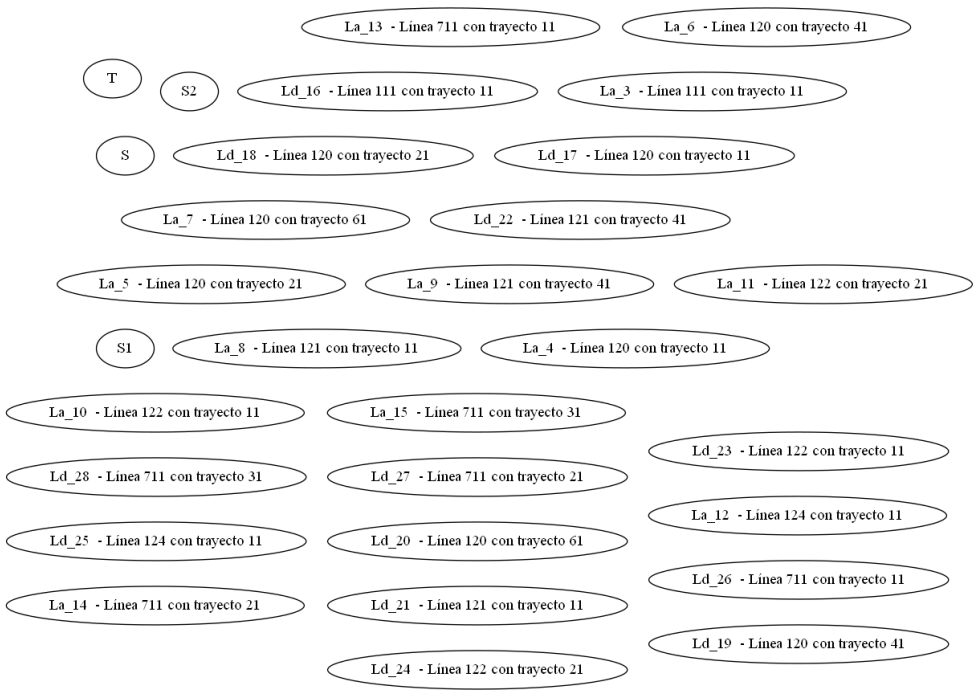


Figura 3.11. Nodos.

5. Una vez tenemos todas las capacidades asignadas a sus arcos, creamos la sumas de demandas parciales, es decir, la suma de las demandas que corresponden a las paradas anteriores a la parada nueva $\sum_{j=1}^{n_a} d_j^a = d_a$ y la suma de demandas de las paradas posteriores $\sum_{j=1}^{n_b} d_j^b = d_b$.
6. Creamos los arcos de S a S_1 y S_2 cuyos flujos serán las sumas de demandas parciales (d^a y d^b, respectivamente) para que exista factibilidad.

7. Por último creamos los arcos que unen líneas y paradas. En el caso de que una línea pase por una parada, se añadirá un arco con capacidad 100 (puesto que no nos es relevante este dato).

Por el momento nos queda unir S_1 y S_2 con las líneas pero lo dejamos para más adelante pues necesitamos calcular el flujo de estos arcos de una manera más específica (siguiendo el algoritmo 2). Por lo que, por el momento, tenemos los arcos que podemos ver en la figura 3.12.

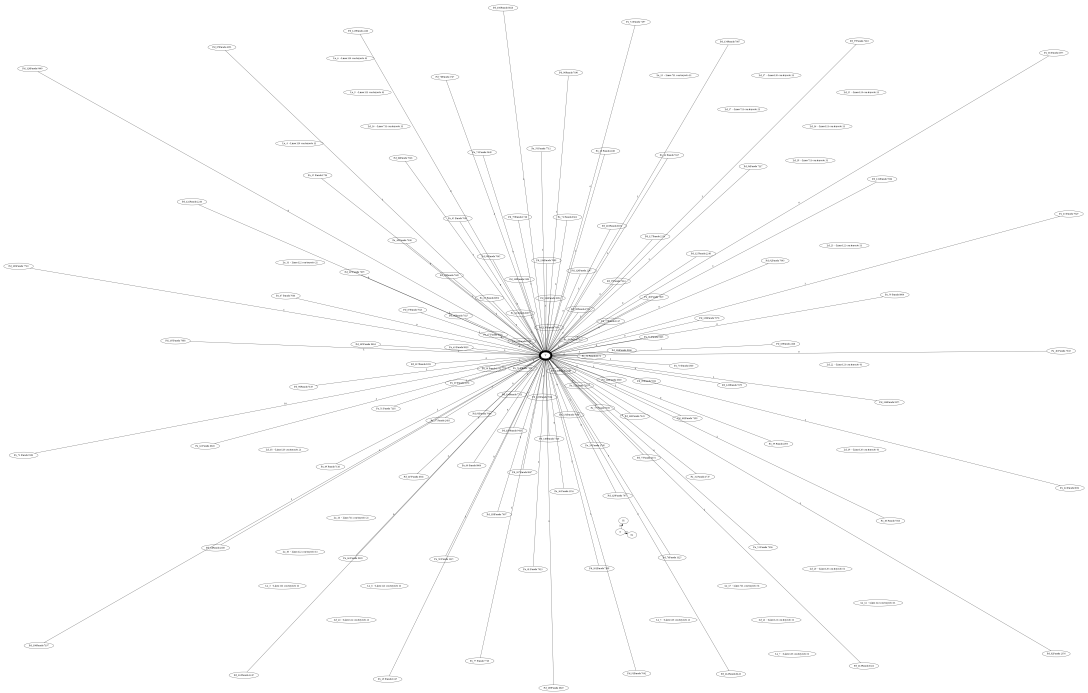


Figura 3.12. Arcos.

Creación de las listas

Declaramos y ordenamos las listas de capacidades e incrementos de tiempo de las líneas (SC y ST, respectivamente). Eliminamos los elementos repetidos en cada lista, obteniendo de esta manera las listas que podemos ver en la siguiente figura 3.13

```
SC (capacidades) = [1, 10, 12, 13, 15, 16, 17, 18, 19, 27, 31, 32, 37, 38, 39, 42, 54, 57, 58]
ST (tiempos) = [0, 3, 4, 14, 19, 21]
```

Figura 3.13. Listas de capacidades y tiempos.

3.5.2. Creación del grafo - Fase 2

1. Para comenzar la creación de los últimos arcos que nos quedan primero debemos entrar en dos búsquedas dicotómicas enlazadas. Realizando en primer lugar la que corresponde a las capacidades de las líneas (SC) y siguiendo con la de incremento de tiempos (ST).
2. Una vez estamos dentro de ambas búsquedas, para cada línea, si ocurre que el incremento de tiempo (t_i) es menor o igual que la mitad correspondiente a la de la lista ST (mitad2) entonces añadimos arcos desde S_1 y S_2 a esa línea. La capacidad que tendrá el arco será el resultante de calcular el mínimo entre la capacidad inicial de la línea (c_i) y la mitad correspondiente a la búsqueda de la lista SC (mitad1).
3. Calculamos el flujo máximo del grafo, puesto que en este instante tenemos ya un grafo conectado.
   Por ejemplo, en este punto, tendremos un grafo como el que podemos ver en la figura 3.14. Donde:
   a) Arco en negro: No hay flujo en el arco.
   b) Arco en rojo: El flujo es igual a la capacidad inicial del arco.
   c) Arco en verde: Hay flujo positivo en el arco, es decir, menor que la capacidad inicial.

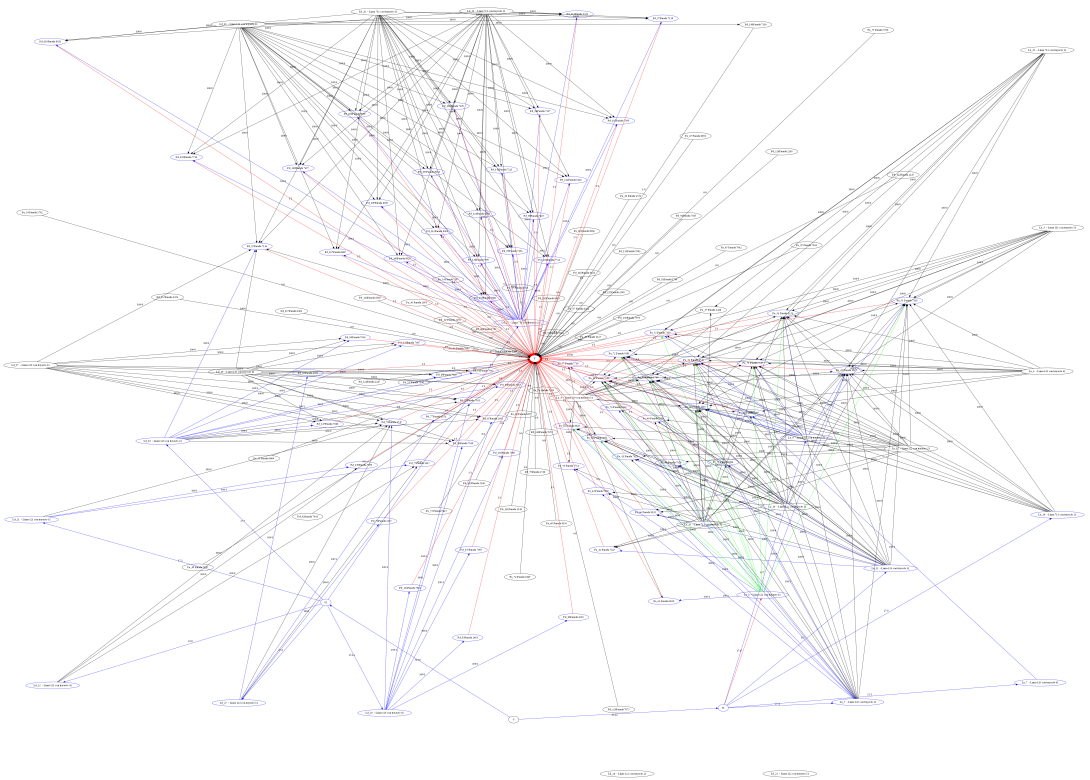


Figura 3.14. Grafo para la capacidad $\bar{Y} = 8$ e incremento de tiempo $\bar{T} = 3$.

4. Declaramos la suma de los flujos desde el nodo fuente S a S_1 y S_2. Si se cumple que dicha suma es mayor o igual a la suma de demandas total (d^a+d^b) entonces el problema es factible.
5. En el caso de que esta desigualdad no se cumpla, se pasa a la siguiente iteración del bucle de las búsquedas dicotómicas y se repite el problema pero con diferentes valores.
   Para poder repetir el problema primero debemos eliminar los arcos creados al principio de esta segunda fase, es decir, eliminamos los arcos de S a S_1 y S_2.
6. En la última iteración factible buscamos la Y óptima. Para ellos declaramos los extremos del intervalo donde se encuentra dicha capacidad.

El intervalo en el que se encuentra la Y óptima es: [Lc , Rc]=[1,10]

7. Comenzamos de nuevo las búsquedas dicotómicas con este intervalo y repetimos el problema, obteniendo así los valores óptimos del problema.

Soluciones factibles

Como ya hemos dicho, hemos obtenido varias soluciones factibles antes de calcular la óptima, de entre las cuales se encuentran:

- Con $\bar{T} = 19$
  - $\bar{Y} = 16$
  - $\bar{Y} = 12$
  - $\bar{Y} = 8$
  - $\bar{Y} = 7$
- Con $\bar{T} = 4$
  - $\bar{Y} = 13$
  - $\bar{Y} = 10$

Cada una de las soluciones tiene asociado un grafo factible.

Valores óptimos

Los valores óptimos y los flujos de S a S_1 y S_2 también óptimos que obtenemos son los siguientes:

Valores óptimos: Y = 7, T = 19
Flujos óptimos de S a S1 y S2, respectivamente: (35,49)

Se puede observar que esta solución puede no beneficiar a la empresa, por lo que TITSA optaría por la solución factible $\bar{Y} = 10$ y $\bar{T} = 4$ pues es más equilibrada ya que el tiempo de trayecto influye mucho a la hora de tomar decisiones sobre la modificación de los recorridos. Aún así, en el caso de que la empresa escogiera utilizar la solución óptima, estos serían los datos obtenidos:

1. La relación entre líneas, paradas y pasajeros recogidos según la ubicación de las paradas (**Tabla A** para los valores que correspondan a paradas ubicadas antes del punto de acceso y **Tabla B** para los posteriores).
2. Grafo correspondiente a dicha solución (figura 3.15).

Línea	**Trayecto**	**Parada**	**Pasajeros**
111	11	1115	1
111	11	1151	2
111	11	1747	1
111	11	1748	1
111	11	2610	1
120	11	7116	1
120	11	7117	1
120	11	7118	1
120	11	7710	1
120	11	7711	1
120	11	8007	1
120	21	7030	1
120	21	8028	1
120	21	8031	1
120	21	8360	1
120	21	8361	1
120	21	9181	1
120	41	7119	1
120	41	9181	5
121	11	8029	1
121	11	9181	4
122	11	7018	1
122	11	7023	1
122	11	7025	1
122	11	7029	1
122	11	8128	1
711	11	2418	1

Tabla A. Solución óptima correspondiente a las paradas anteriores al punto de acceso del estudio.

Línea	Trayecto	Parada	Pasajeros
111	11	7117	1
111	11	7118	1
111	11	7119	1
111	11	7196	1
111	11	7197	1
111	11	7097	1
120	11	1747	4
120	11	2693	1
120	11	7018	1
120	21	7023	2
120	21	7025	1
120	21	7029	1
120	21	7042	1
120	41	7116	1
120	41	7087	1
120	41	7088	1
120	61	2418	1
120	61	2610	1
120	61	7084	1
120	61	7095	1
121	11	1115	1
121	11	7099	1
124	11	1125	1
124	11	1151	1
124	11	2356	1
711	11	7198	1
711	11	7199	1
711	11	7710	1
711	11	7711	1
711	11	8006	1
711	21	8007	1
711	21	8028	3
711	21	8029	1
711	21	8031	1
711	31	8094	1
711	31	8128	2
711	31	8321	1
711	31	8355	1

Tabla B. Solución óptima correspondiente a las paradas posteriores al punto de acceso del estudio.

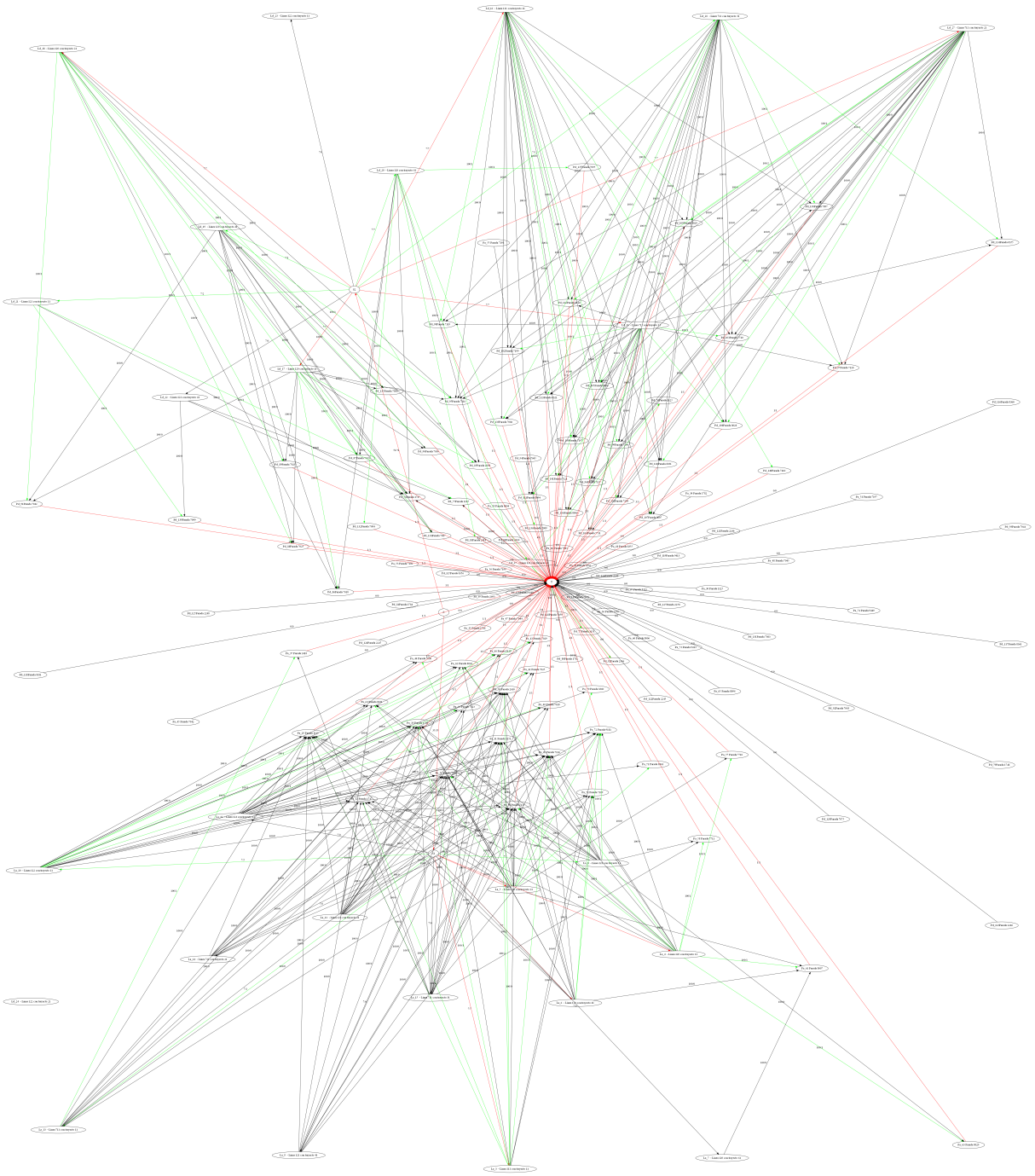


Figura 3.15. Grafo óptimo.

Conclusiones

En este trabajo se ha conseguido mostrar como las técnicas de programación matemática se pueden combinar con otras técnicas algorítmicas con el fin de desarrollar procedimientos para dar respuesta a problemas de organización de recorridos de autobuses.

El modelo creado se puede convertir en un instrumento útil para la empresa TITSA encargada del sector de transporte de la isla de Tenerife.

Bien es conocido que TITSA ya se nutre de sus propios modelos y algoritmos y que por tanto implementar este puede ser un tanto laborioso, pero por ello nos hemos intentado ceñir en la medida de lo posible a la forma de programación de la empresa.

Es fácil ver que aún teniendo una solución óptima esta puede no ser la que interese a la empresa, es por este motivo por el que presentamos también el resto de soluciones factibles que satisfacen todas las restricciones.

Tras la realización de este trabajo se pretende estudiar el caso en que el problema no pueda ser resuelto de manera óptima en un número logarítmico de aplicaciones del MFP. En este caso, en el que la función objetivo sería la de mín $\sum_{i=1}^{m} t_i z_i$ deberíamos utilizar el concepto de *Red Residual* e ir añadiendo cada línea en orden creciente según los incrementos de tiempo hasta alcanzar la factibilidad.

A

Apéndice

Como ya hemos mencionado con anterioridad, los scripts con la creación de los grafos están implementados en lenguaje Python. Las librerías utilizadas para todos los scripts son:

```
pip install coinor.gimpy
from coinor.gimpy import Graph, DIRECTED_GRAPH
import pandas as pd
```

A.1. Preproceso

Hemos creado el archivo 'DATOSVERSIONFINAL.CSV" a partir de la libreta de excel proporcionada por TITSA que contiene los correspondientes datos relacionados con las paradas y las líneas:

"DATOS_MATRIZ_DISTANCIA_TIEMPO_PASAJE"

```
#IMPORTAMOS LOS DATOS
datos = pd.read_csv('DATOSVERSIONFINAL.csv', delimiter=";", header=None)
datos = datos.fillna(0)
print(datos)

#NÚMERO DE PARADAS ANTES, DESPUÉS, TOTAL Y NÚMERO DE LÍNEAS
nparadasantes= datos.iat[0,1]
nparadasdespues= datos.iat[0,2]
nparadas = nparadasdespues + nparadasantes
nlineas= datos.iat[0,3]
print('Numero de paradas antes=', int(nparadasantes))
print('Numero de paradas despues=', int(nparadasdespues))
print('Numero de paradas total=', int(nparadas))
print('Numero de lineas=', int(nlineas))

#INCREMENTO DE TIEMPO Y CAPACIDADES DE LAS LÍNEAS ANTES Y DESPUÉS
incrementotiempos = list(int(n) for n in datos.iloc[1:14, 3])
capacidadantes= list(int(n) for n in datos.iloc[1:14, 4])
capacidaddespues= list(int(n) for n in datos.iloc[1:14,5])
```

```
#DEMANDAS DE PASAJEROS ANTES Y DESPUÉS
demandasantes= list(int(n) for n in datos.iloc[14:60, 2])
demandasdespues= list(int(n) for n in datos.iloc[60:127, 2])
print("DEMANDAS ANTES: ", demandasantes)
print("DEMANDAS DESPUES: ", demandasdespues)

#CÓDIGO DE LAS LÍNEAS CON SUS TRAYECTOS Y CÓDIGO DE LAS PARADAS
codigolineas= list(int(n) for n in datos.iloc[1:14, 1])
print('lin', codigolineas)
codigotrayecto= list(int(n) for n in datos.iloc[1:14, 2])
print('tra', codigotrayecto)
etiquetalineas = (codigolineas, codigotrayecto)
print('CÓDIGO LINEAS:', etiquetalineas)
codigoparadas= list(int(n) for n in datos.iloc[14:127, 1])
print('CÓDIGO PARADAS:',codigoparadas)

#LÍNEAS Y PARADAS CONECTADAS ENTRE SI
lineas= list(int(n) for n in datos.iloc[246:554, 0])
print("LÍNEAS (arcos):", lineas)
paradas= list(int(n) for n in datos.iloc[246:554, 1])
print("PARADAS (arcos):", paradas)
```

A.2. Creación del grafo - 1ª FASE

```
#DECLARAMOS LAS SUMAS DE DEMANDAS PARCIALES (ANTES Y DESPUÉS)
sum1d=0
sum2d=0

#COMENZAMOS LA CREACIÓN DEL GRAFO
if __name__=='__main__':
    g = Graph(type = DIRECTED_GRAPH, display = 'off')

    #CREAMOS LOS NODOS FUENTE Y SUMIDERO A MANO
    g.add_node(0, label='S')    #Nodo fuente S
    g.add_node(1, label= 'S1')    #Nodo S1
    g.add_node(2, label='S2')    #Nodo S2
    g.add_node(int(2*nlineas + nparadas + 3), label='T')  #Nodo sumidero T

    #PARA CADA LÍNEA CREAMOS UN NODO Y SUS ARCOS CORRESPONDIENTES CON S1 O S2
    for i in range(3, int(nlineas) +3):
            g.add_node(i, label= 'La_' + str(i)+'  - Línea '+ str(codigolineas[i-3])+
            ' con trayecto '+ str(codigotrayecto[i-3])) #de 3 a 15
            g.add_node(i + int(nlineas), label= 'Ld_'+str(i+int(nlineas))+'  - Línea '
            + str(codigolineas[i-3])+' con trayecto '+ str(codigotrayecto[i-3])) #de 16 a 28

    #CREAMOS UN ARCHIVO .PNG CON TODOS LOS NODOS
    g.set_display_mode('file')
    g.display(highlight = None, basename = 'graphnodes', format = 'png', pause = True)
```

```
#PARA CADA PARADA SITUADA ANTES DE LA PARADA NUEVA CREAMOS UN NODO Y UN ARCO DIRIGIDO
#HACIA EL NODO SUMIDERO T CON SU DEMANDA CORRESPONDIENTE
for j in range(29, int(nparadasantes)+29):
    if g.get_node(j) == None:
        g.add_node(j, label='Pa_'+str(j) + ' Parada ' + str(codigoparadas[j-29]))
        #de 29 a 74
    if g.check_edge(j, int(2*nlineas + nparadas + 3)) == False:
        g.add_edge(j, int(2*nlineas + nparadas + 3), capacity= demandasantes[j-29],
        label= demandasantes[j-29]) #demanda de los pasajeros de cada parada antes
        #de la parada nueva

#CREAMOS LA SUMA DE DEMANDAS CORRESPONDIENTE A LAS PARADAS ANTES
for n in demandasantes:
    sum1d= sum1d + n

#PARA CADA PARADA SITUADA ANTES DE LA PARADA NUEVA CREAMOS UN NODO Y UN ARCO DIRIGIDO
HACIA EL NODO SUMIDERO T CON SU DEMANDA CORRESPONDIENTE
for j in range(int(nparadasantes)+29, (int(nparadasantes)+ int(nparadasdespues)+29)):
    if g.get_node(j) == None:
        g.add_node(j, label='Pd_'+str(j) + 'Parada ' + str(codigoparadas[j-75]))
        #de 75 a 141
    if g.check_edge(j, int(2*nlineas + nparadas + 3)) == False:
        g.add_edge(j,int(2*nlineas + nparadas + 3), capacity= demandasdespues[j-75],
        label= demandasdespues[j-75]) #demanda de los pasajeros de cada parada despues
        #de la parada nueva

#CREAMOS LA SUMA DE DEMANDAS CORRESPONDIENTE A LAS PARADAS DESPUÉS
for m in demandasdespues:
    sum2d= sum2d + m

#CREAMOS LOS ARCOS DE S A S1 Y S2 CON CAPACIDAD LA SUMA DE DEMANDAS QUE LES CORRESPONDA
#(ANTES O DESPUÉS)
g.add_edge(0, 1, capacity=sum1d, label= sum1d)
g.add_edge(0, 2, capacity=sum2d, label= sum2d)

#DECLARAMOS LA SUMA DE DEMANDAS TOTAL E IMPRIMIMOS ESTA Y LAS PARCIALES
sumademandas= sum1d + sum2d
print("SUMA DEMANDAS ANTES:", sum1d)
print("SUMA DEMANDAS DESPUÉS:", sum2d)
print("SUMA DE LAS DEMANDAS =", sumademandas)

#CREAMOS LOS ARCOS ENTRE LAS LÍNEAS Y LAS PARADAS POR LAS QUE PASAN
count=0
for x in lineas:
  y=paradas[count]
  if y <= nparadasantes:
    if g.check_edge(x+2, y+28)==False:
      g.add_edge(x+2, y+28, capacity=100)
  else:
    if g.check_edge(x+15, y+28)==False:
      g.add_edge(x+15,y+28, capacity=100)
  count=count+1

#CREAMOS UN ARCHIVO .PNG CON TODOS LOS ARCOS CREADOS, ES DECIR, TODOS SALVO LOS QUE
#UNEN S1 Y S2 CON LAS LÍNEAS
g.set_display_mode('file')
g.display(highlight = None, basename = 'grapharcs', format = 'png', pause = True)
```

A.3. Creación de las listas para las búsquedas dicotómicas

Se declaran las listas de capacidades de las líneas e incremento de tiempo de cada recorrido.

```
#DECLARAMOS LAS LISTAS DE CAPACIDADES Y DE INCREMENTO DE TIEMPO
listacapacidades= list(capacidadantes + capacidaddespues)
listatiempos = list(int(n) for n in incrementotiempos)

#ORDENAMOS LAS LISTAS
listacapacidades.sort()
listatiempos.sort()
print("Listacapacidades: ", listacapacidades)
print("listatiempos: ", listatiempos)

#ELIMINAMOS LOS ELEMENTOS REPETIDOS EN CADA LISTA
lista1 = []
for i in listacapacidades:
    if i not in lista1:
        lista1.append(i)

lista2 = []
for i in listatiempos:
    if i not in lista2:
        lista2.append(i)

print("LISTA 1 (capacidades):", lista1)
print("LISTA 2 (tiempos):", lista2)
```

A.4. Creación del grafo - 2ª FASE

Se crean los arcos restantes para completar el grafo y con ello comienzan las búsquedas discotómicas.

```
#COMIENZA LA BÚSQUEDA DICOTÓMICA DE LA LISTA DE CAPACIDADES
Lc=0
Rc= len(lista1)-1
while Lc < Rc:
    mitad1 = (Lc+Rc)/2
```

```
#COMIENZA LA BÚSQUEDA DICOTÓMICA DE LA LISTA DE INCREMENTO DE TIEMPO
Lt=0
Rt= len(lista2)-1
factible = False
while Lt < Rt:
    mitad2=(Lt+Rt)/2

    #PARA CADA LÍNEA, SI OCURRE QUE EL INCREMENTO DE TIEMPO ES MENOR O
    #IGUAL QUE LA MITAD2 (CONSEGUIDA EN LA SEGUNDA BÚSQUEDA)
    #ENTONCES AÑADIMOS ARCOS DESDE S1 A S2 A ESA LÍNEA CUYA CAPACIDAD
    #SERÁ EL MÍNIMO ENTRE LA CAPACIDAD INICIAL QUE LE PERNTENEZCA
    #Y MITAD1
    g.set_display_mode('off')
    for n in range(3,16):
        if incrementotiempos[n-3] <= int(lista2[int(mitad2)]):
            if g.check_edge(1, n)==False:
                g.add_edge(1, n, capacity= min(int(lista1[int(mitad1)]),
                int(capacidadantes[n-3])))
            if g.check_edge(2, n + int(nlineas))==False:
                g.add_edge(2, n + int(nlineas), capacity=
                min(int(lista1[int(mitad1)]), int(capacidaddespues[n-3])))

    #CALCULAMOS EL FLUJO MÁXIMO DEL GRAFO TOTAL RESULTANTE DE AÑADIR LOS ARCOS
    #ANTERIORES
    g.max_flow(0, int(2*nlineas + nparadas + 3), algo= 'BFS')

    #CREAMOS UN ARCHIVO .PNG PARA DICHO GRAFO
    #LO NOMBRAMOS SEGÚN LA MITAD1 Y MITAD2 CON LA QUE ESTEMOS TRABAJANDO EN
    #ESTA ITERACIÓN DE LAS BÚSQUEDAS
    g.set_display_mode('file')
    g.display(highlight = None, basename = 'graph'+ str(int(mitad1))+ '_'+
    str(lista2[int(mitad2)]), format = 'png', pause = True)

    #DECLARAMOS LA SUMA DE LOS FLUJO DESDE EL NODO FUENTE S A S1 Y S2
    sumaflujo = g.get_edge_attr(0,1, 'flow') + g.get_edge_attr(0,2, 'flow')
    print("sumaflujo",sumaflujo)

    #SI SE CUMPLE QUE DICHA SUMA ES MAYOR O IGUAL A LA SUMA DE DEMANDAS
    #DECLARADA CON ANTERIORIDAD
    #ENTONCES EL PROBLEMA ES FACTIBLE
    if sumaflujo >= sumademandas:
        Y  = lista1[int(mitad1)]  #capacidad de la lista1 que se encuentra
        #en la posicion de mitad1
        time = lista2[int(mitad2)]  #tiempo de la lista2 que se encuentra
        #en la posición de mitad2
        Rt   = int(mitad2)
        factible = True
        print("PROBLEMA FACTIBLE PARA LOS SIGUIENTES VALORES:")
        print("CAPACIDAD:",lista1[int(mitad1)], "TIEMPO:", lista2[int(mitad2)])

    #EN CASO DE NO FACTIBILIDAD, REPETIMOS EL PROBLEMA. CAMBIAMOS LOS VALORES
    #DE LA BÚSQUEDA.
    else:
        Lt = int(mitad2) + 1
```

```
        #PARA CADA LÍNEA ELIMINAMOS LOS ARCOS DESDE S1 Y S2 PARA PODER REPETIR EL
        #PROCESO EN LA SIGUIENTE ITERACIÓN
        for n in range(3,16):
            if incrementotiempos[n-3] <= lista2[int(mitad2)]:
                g.del_edge((1, n))
                g.del_edge((2, n+int(nlineas)))
    if factible:
        Rc = int(mitad1)

    else:
        Lc = int(mitad1) + 1

#EN LA ÚLTIMA ITERACIÓN FACTIBLE BUSCAMOS LA CAPACIDAD ÓPTIMA
print("BUSCANDO LA CAPACIDAD ÓPTIMA")

#DECLARAMOS LOS EXTREMOS DEL INTERVALO DONDE SE ENCUENTRA DICHA CAPACIDAD
if factible==False:
    Rc = Rc+1
if Rc==0:
    Lc=1
else:
    Lc=lista1[(Rc-1)+1]
Rc = lista1[Rc]

print("Este es el intervalo en el que se encuentra la capacidad óptima: (Lc,Rc)=",
(Lc,Rc))

#COMIENZA DE NUEVO UNA PRIMERA BÚSQUEDA DICOTÓMICA ENTRE LAS CAPACIDADES DE DICHO
#INTERVALO
while Lc < Rc:
    mitad1 = (Lc+Rc)/2

    #COMIENZA LA SEGUNDA BÚSQUEDA, ESTA VEZ ENTRE LOS TIEMPOS
    Lt=0
    Rt= len(lista2)-1
    factible = False

    while Lt < Rt:
        mitad2=(Lt+Rt)/2

        #PARA CADA LÍNEA, SI OCURRE QUE EL INCREMENTO DE TIEMPO ES MENOR O IGUAL QUE
        #LA MITAD2 (CONSEGUIDA EN LA SEGUNDA BÚSQUEDA)
        #ENTONCES AÑADIMOS ARCOS DESDE S1 A S2 A ESA LÍNEA CUYA CAPACIDAD SERÁ EL
        #MÍNIMO ENTRE LA CAPACIDAD INICIAL QUE LE PERNTENEZCA
        #Y MITAD1
        g.set_display_mode('off')
        for n in range(3,16):
            if incrementotiempos[n-3] <= int(lista2[int(mitad2)]):
                if g.check_edge(1, n)==False:
                    g.add_edge(1, n, capacity= min(int(mitad1),
                    int(capacidadantes[n-3])))
                if g.check_edge(2, n + int(nlineas))==False:
                    g.add_edge(2, n + int(nlineas), capacity= min(int(mitad1),
                    int(capacidaddespues[n-3])))
```

```
        #VOLVEMOS A CALCULAR EL FLUJO TOTAL DEL GRAFO QUE TENEMOS POR EL MOMENTO
        g.max_flow(0, int(2*nlineas + nparadas + 3), algo= 'BFS')

        #CREAMOS UN ARCHIVO .PNG CON DICHO GRAFO
        g.set_display_mode('file')
        g.display(highlight = None, basename = 'graph'+ str(int(mitad1))+ '_'+
        str(lista2[int(mitad2)]), format = 'png', pause = True)

        #DECLARAMOS LA SUMA DE LOS NUEVOS FLUJOS DESDE S A S1 Y S2
        sumaflujo = g.get_edge_attr(0,1, 'flow') + g.get_edge_attr(0,2, 'flow')
        print("sumaflujo",sumaflujo)

        #SI SE CUMPLE QUE DICHA SUMA ES MAYOR O IGUAL A LA SUMA DE DEMANDAS
        #DECLARADA CON ANTERIORIDAD
        #ENTONCES EL PROBLEMA ES FACTIBLE
        if sumaflujo >= sumademandas:
            Y  = int(mitad1)
            time = lista2[int(mitad2)]
            Rt   = int(mitad2)
            factible = True
            print("PROBLEMA FACTIBLE PARA LOS SIGUIENTES VALORES:")
            print("CAPACIDAD:",int(mitad1), "TIEMPO:", lista2[int(mitad2)] )

        #EN CASO DE NO FACTIBILIDAD, REPETIMOS EL PROBLEMA. CAMBIAMOS LOS VALORES
        #DE LA BÚSQUEDA.
        else:
            Lt = int(mitad2) + 1

        #PARA CADA LÍNEA ELIMINAMOS LOS ARCOS DESDE S1 Y S2 PARA PODER REPETIR EL
        #PROCESO EN LA SIGUIENTE ITERACIÓN
        for n in range(3,16):
            if incrementotiempos[n-3] <= lista2[int(mitad2)]:
                g.del_edge((1, n))
                g.del_edge((2, n+int(nlineas)))
    if factible:
        Rc = int(mitad1)

    else:
        Lc = int(mitad1) + 1
```

A.5. Script de correspondencia de línea y trayecto con la capacidad óptima

Una vez hallados los valores de capacidad y tiempo óptimos buscamos la línea y el trayecto correspondiente.

```
#YA TENEMOS LOS VALORES CAPACIDAD Y TIEMPO ÓPTIMOS
print("Optimal values: Y=",Y, "time=",time)

#PARA ENCONTRAR LOS FLUJOS ÓPTIMOS EJECUTAMOS UNA ITERACIÓN CON LOS VALORES DE
#CAPACIDAD Y TIEMPO ÓPTIMOS
print("Flujos óptimos de S a S1 y S2, respectivamente:",
(g.get_edge_attr(0,1, 'flow'),g.get_edge_attr(0,2, 'flow')))

#IMPRIMIMOS EL GRAFO FINAL CON LOS FLUJOS CORRESPONDIENTES
print("GRAFO FINAL CON LOS FLUJOS CORRESPONDIENTES: ")
nl = list(int(n) for n in g.get_node_list())
nl.sort()
for n in nl:
    for m in nl:
        if g.check_edge(n, m):
            if g.get_edge_attr(n, m, 'flow') > 0:
                if 3<=n<=15 and 29<=m<=74:
                    print("La línea", codigolineas[n-3], "con trayecto",
                    codigotrayecto[n-3],"recoge en la parada",codigoparadas[m-29],
                    "los siguientes pasajeros:", g.get_edge_attr(n, m, 'flow'))
                elif 16<=n<=28 and 75<=m<=141:
                    print("La línea", codigolineas[n-16], "con trayecto",
                    codigotrayecto[n-16], "lleva a la parada",codigoparadas[m-75],
                     "los siguientes pasajeros:", g.get_edge_attr(n, m, 'flow'))
                else:
                    print(n, m, g.get_edge_attr(n, m, 'flow'))


#NOS INTERESA SABER A QUE LÍNEA Y TRAYECTO CORRESPONDE LA CAPACIDAD ÓPTIMA
if Y in listacapacidades:
    print("La capacidad óptima", Y, "corresponde a la línea",
    codigolineas[listacapacidades.index(Y)], "con trayecto",
    codigotrayecto[listacapacidades.index(Y)])
else:
    while Y not in listacapacidades:
        Y=Y+1
    print("La capacidad óptima corresponde a la línea",
    codigolineas[listacapacidades.index(Y)], "con trayecto",
    codigotrayecto[listacapacidades.index(Y)])
```

Bibliografía

[1] HILLIER F.S. y LIEBERMAN G. J. *Introducción a la Investigación de Operaciones.* Mcgraw-Hill, 2010.

[2] BAZARAA, M. S.; JARVIS, J.J. y SHERALI, H. D. *Linear Programming and Network Flows.* John Wiley, 2010.

[3] HILL, CHRISTIAN. *Learning Scientific Programming with Python.* Cambridge University Press, 2016.

[4] JOHANSSON, ROBERT. *Numerical Python Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib.* Apress, 2019.

[5] Computacional Infrastrusture for Operations Research. *Projects* [Graphs]. [Fecha de consulta: 17-01-2022]. Disponible en: https://www.coin-or.org/projects/#ffs-tabbed-13.

[6] González Martín, C (2017-2018). Apuntes de la asignatura de Optimización del Grado de Matemáticas de la ULL. *Fotocopias campus.*

[7] Del Río Gómez, Diego. Un método simplex en programación lineal multiobjetivo. Universidad de Valladolid. Disponible en: https://uvadoc.uva.es/bitstream/handle/10324/50558/TFG-G5354.pdf?sequence=1&isAllowed=y.

[8] Coursera. *Google IT Automation with Python.* 2022. https://www.coursera.org/.

[9] TITSA S.A. *Power Bi* 2019.

Lista de símbolos y abreviaciones

AI	Artificial Intelligence
ML	Machine Learning
PEP	Problema para Eliminar Parada
PMR	Problema para Modificar Recorridos
MFP	Problema de Flujo Máximo
PL	Problema Lineal
PBL	Problema Binivel Lineal

Smart system to find the efficiency of the access points (stops) to the transport network in Tenerife

Sección de Matemáticas
Universidad de La Laguna

Saray Pérez Delgado
Facultad de Ciencias · Sección de Matemáticas
Universidad de La Laguna
alu0100953428@ull.edu.es

Abstract

On Tenerife island there are unsatisfactory access points or bus stops from the perspective of the Tenerife Interurban Transport company, TITSA. In this final degree project we will try to model and solve this problem from a practical case, as well as how to use an intelligent system to study the data of the lines susceptible to change their routes. The purpose is to optimize the travel time of the modified service.

1. Introduction

In this project we are going to show how the application of artificial intelligence techniques together with mathematical programming help to make better decisions in the performance of a company. Our goal is to improve the efficiency of certain access points (stops) of the Tenerife Irterurban Transport company. Firts, we will see how TITSA draws on machine learning and artificial intelligence to enter in the world of Big Data & Data Science. Then, we'll talk about linear programming and how we have treated the data that has been provided to us. In addition, we will study several branches of the problem and we will emphasize in a particular case. Finally we'll model and solve this case thanks to linear programming.

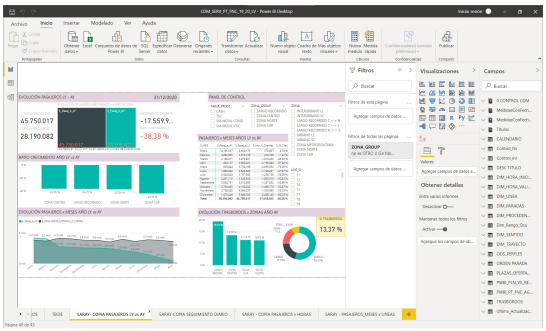


Figure 1: *Dashboard of visualizations in Power BI desktop.*

TITSA has found some problems in Polígono de Güímar, since it is observed that many people in this sector demand to go to a stop located in front of 3 huge supermarkets but there is no route that facilitates it. Therefore, we must study whether it is possible to modify the routes of the lines that pass through Güímar and make them pass through this stop without harming the company.

February 2021 passenger data from Polígono de Güímar:

Incoming and outgoing	Incoming	Outgoing	Demand
59	16	43	86

2. Resolution with Python

It is intended to develop a mathematical optimization model whose objective function is to minimize the maximum sum of the time increments of each line when adding the new access point. This leads to two least cost flow problems bounded by a single objective function with variables $z_i \in \{0, 1\}$ of activation of the corresponding line.
We have created a code in the python programming language where all the feasible solutions of the problem will appear in the form of directed graphs.

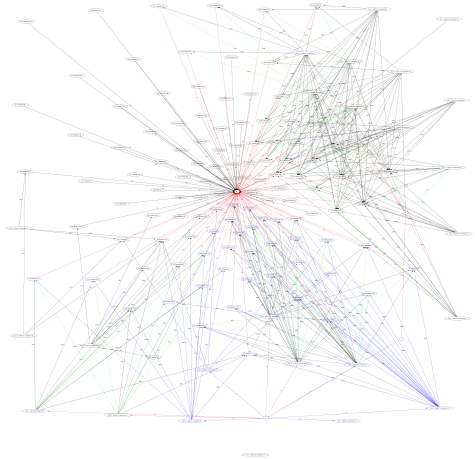


Figure 2: *Feasible solution where capacity of the lines is $\bar{Y} = 7$ and time is $\bar{T} = 4$.*

The final optimal graph (where capacity of the lines is 7 and the increase of time is 19) has some of the following flow relationships, that is, we can see the relationship of lines, stops and passengers picked up:

Line	Route	Stops	Passengers
111	11	1115	1
120	41	9181	5
711	21	8028	3

Small part of all the corresponding feasible solutions.

References

[1] HILLIER F.S. y LIEBERMAN G. J. *Introducción a la Investigación de Operaciones*. Mcgraw-Hill, 2010.

[2] Computacional Infrastrusture for Operations Research. *Projects* [Graphs]. [Fecha de consulta: 17-01-2022]. Disponible en: https://www.coin-or.org/projects.

[3] Coursera. *Google IT Automation with Python*. 2022. https://www.coursera.org/.

[4] TITSA S.A. *Power Bi*. 2019.