ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

OPTIMIZACIÓN EN SISTEMAS DE TRANSPORTE PÚBLICO. MODELO DE CUBRIMIENTO DE DEMANDA PARA EL PROBLEMA DE PLANIFICACIÓN DE LÍNEAS Y FRECUENCIAS

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO MATEMÁTICO

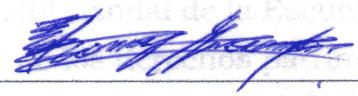
RONNY EDUARDO GUAMÁN YUNGA
ronny.guaman@epn.edu.ec

DIRECTOR: RAMIRO DANIEL TORRES GORDILLO
ramiro.torres@epn.edu.ec

DMQ, AGOSTO 2022

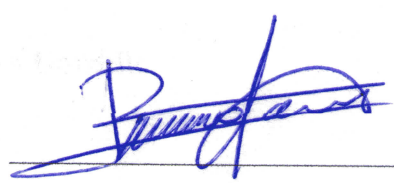
CERTIFICACIONES

Yo, RONNY EDUARDO GUAMÁN YUNGA, declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.



Ronny Eduardo Guamán Yunga

Certifico que el presente trabajo de integración curricular fue desarrollado por Ronny Eduardo Guamán Yunga, bajo mi supervisión.



Ramiro Daniel Torres Gordillo

DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el(los) producto(s) resultante(s) del mismo, es(son) público(s) y estará(n) a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

Ronny Eduardo Guamán Yunga

Ramiro Daniel Torres Gordillo

RESUMEN

En la presente componente, nos enfocamos en estudiar el Problema de Planificación de Líneas con aplicación al Sistema Municipal de Transporte del Distrito Metropolitano de Quito. El problema consiste en hallar un conjunto de líneas factibles, con sus respectivas frecuencias, de manera que cubran una demanda de pasajeros establecida en un horizonte de tiempo. El objetivo de ello es minimizar los costos de operación del sistema. En este trabajo, se discute un modelo de programación lineal entera para el Problema de Planificación de Líneas y Frecuencias con el fin de minimizar los costos totales mientras se garantiza un cierto nivel de calidad de servicio, en términos de la capacidad de transporte disponible. Se analiza el comportamiento del modelo usando diferentes topologías de redes (líneas, árboles y grafos generales) y se describe un método heurístico de solución. Finalmente, experimentos computacionales con instancias de diferentes tamaños son reportados.

Palabras clave: Planificación de Líneas y Frecuencias, Programación Lineal Entera, Métodos Heurísticos, Transporte Público.

ABSTRACT

In this component, the Line Planning Problem with application to the Public Transportation System of the Distrito Metropolitano de Quito is studied. The problem consists of finding a set of feasible lines, with their respective frequencies, in such a way that passenger demand is covered in an specific time horizon. The objective of this problem is to minimize the operational costs of the system. In this work, an integer linear programming model for such a problem is discussed in order to minimize the total cost, while guaranteeing a certain level of quality of service, in terms of available transport capacity. The behavior of the model is analyzed using different network topologies (lines, trees and general graphs) and a heuristic solution method is described. Finally, computational experiments with instances of different sizes are reported.

Keywords: Line Planning Problem, Integer Programming, Heuristic methods, Public Transport.

Índice general

Índice de figuras

Capítulo 1

Introducción

En varias ciudades de la región, principalmente en las metrópolis o distritos grandes de América Latina, el progresivo incremento de la población humana y la continua extensión demográfica han ocasionado diversos problemas de movilidad en sus habitantes. Estas situaciones están atadas al continuo requerimiento de grandes demandas de transporte con todos los efectos negativos que esto ocasiona: congestión vehicular, contaminación, vulnerabilidad e inseguridad, entre otros. Por este motivo, la Planicación y Gestión del Transporte reconoce la necesidad de tener un sistema de movilidad digno para sus usuarios.

En Quito, se han implementado algunos sistemas de transporte que están a cargo de la movilidad de miles de pasajeros. Algunas de estas redes son conocidas comúnmente como sistemas troncales de transporte que están coordinadas por el municipio de la ciudad. Sistemas como el Trolebús, Ecovía, Corredor Central Norte, Corredor Sur Oriental o Corredor Sur Occidental son algunos de los ejemplos que operan para transportar a la mayoría de la población capitalina. La frecuente expansión territorial de la zona urbana del distrito, que ya bordea los 380 km^2, promueve el trabajo de adaptar nuevas líneas de transporte que garanticen la inclusión de nuevos pasajeros al sistema, lo que desemboca en resolver un problema de planificación con mayor complejidad para los siguientes años. Asimismo, la incorporación de medidas que ofrezcan un servicio eficiente junto con la optimización de recursos también es indispensable.

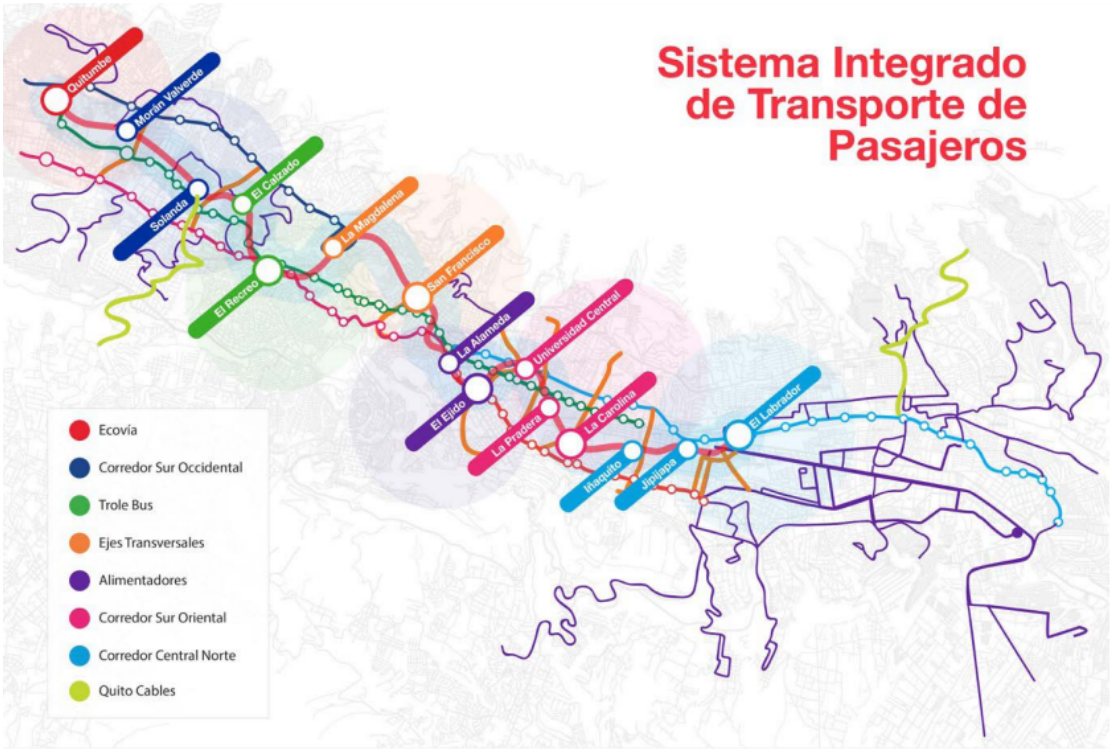


Figura 1.1: Corredores del Sistema Metropolitano de Transporte de Quito.

El Problema de Planificación de Líneas y Frecuencias (LPP) es un problema que ha sido estudiado en los últimos años por la comunidad de la investigación de operaciones, la matemática aplicada, la ingeniería del transporte y otras áreas relacionadas. La fase de planificación del transporte es una de los procedimientos más importantes dentro de toda la gestión, desarrollo y organización del transporte, puesto que permite conocer los problemas, crear soluciones, diseñar estrategias y principalmente, focalizar y optimizar los recursos para atender los requerimientos de movilidad en las grandes ciudades con constante crecimiento urbano y extensión geográfica.

En esta primera etapa de planificación, se encuentra el problema centro de nuestro estudio: La Planificación de Líneas y Frecuencias. El problema consiste en encontrar un conjunto de líneas posibles (de uno o varios modos de transporte), junto con su frecuencia, con la finalidad de cubrir una demanda preestablecida de pasajeros. Generalmente, se plantea que los costos operacionales y de servicio del sistema sean minimizados. En otras palabras, el problema se enfoca en determinar las líneas que deberían implementarse de tal forma que se llegue a cumplir

con el traslado de un número establecido de personas y que se garantice una buena calidad del servicio.

Varios investigadores en el campo de la invetigación de operaciones y de la optimización combinatoria se han enfrentado a este problema. Desde 1990, algunos científicos empezaron con el análisis y tratamiento de problemas relacionados con el LPP. Así, Bussieck et al. [3] estudiaron el Problema de Planificación de Líneas y Frecuencias para trenes en el sistema de transporte público ferrovario. Los autores proponen un modelo de programación lineal entera cuyo objetivo es la maximización de los viajes directos de los pasajeros. En esta modelización, se aseguran que los tiempos de espera y de tránsito de los pasajeros sean minimizados. De igual forma, Bussieck et al. [4] discuten la elección óptima de líneas con horarios períodicos en un sistema ferroviario. Los autores describen varias desigualades válidas y reportan soluciones sobre instancias reales. Por otro lado, Claessens et al. [5] también se enfretaron al problema de planificación de líneas aplicado al sistema ferroviario de Holanda. Los autores plantean un modelo de optimización no lineal que minimiza los costos operacionales totales junto con una versión linealizada del modelo anterior. Un algoritmo del tipo "Branch and Bound" es utilizado en la solución. Goossenes et al. [6] usan los resultados expuestos en [5] e impulsan técnicas que faciliten la resolución del LPP por medio de un algoritmo del tipo "Branch and Cut". En su trabajo, exponen métodos de pre-procesamiento que generan una reducción de variables y restricciones.

Borndörfer et al. [7] proponen un modelo de flujo multiproducto para la planificación de líneas. En esta versión, los pasajeros son enrutados de forma integrada y una generación dinámica de las líneas es considerada. Los autores desarrollaron un algoritmo de generación de columnas y se reportan experimentos computacionales derivados de datos reales de una ciudad en Alemania.

Se conoce que el problema de planificación de líneas y frecuencias pertenece a la clase NP duro para redes generales. Sin embargo, se han estudiado casos en los que se los puede resolver en tiempo polinomial teniendo en cuenta algunas condiciones en redes simplificadas. Así, en [1] y [2] se muestran resultados en los que se puede encontrar algoritmos

polinomiales si se trabaja con topologías tipo líneas o árboles. Su estudio y resultados computacionales son la guía para esta componente.

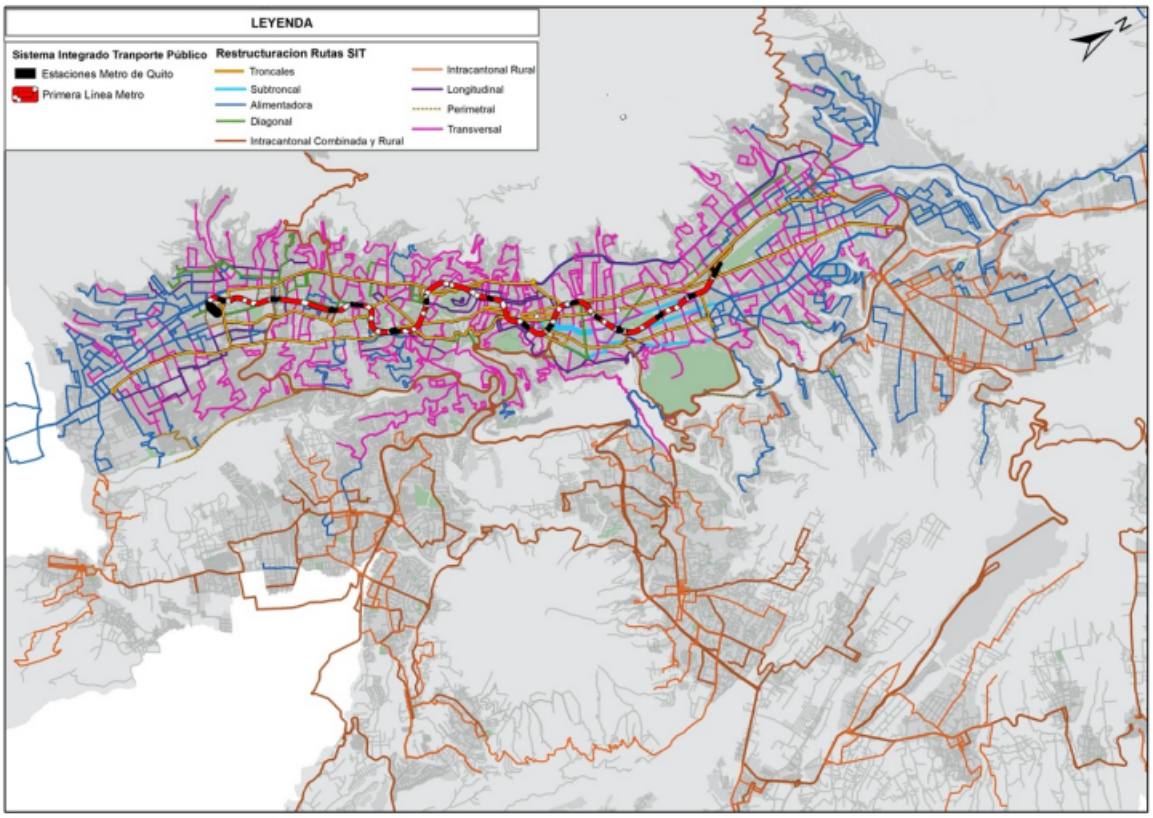


Figura 1.2: Red completa de Transporte Público DMQ 2021

En este trabajo, nos enfocamos en estudiar el Problema de Planificación de Líneas y Frecuencias aplicado a redes simples que se derivan del Sistema de Transporte Público de Quito. Se pretende utilizar los resultados expuestos en [1] y [2] para enriquecerlos realizando experimentos con instancias de gran tamaño y verificar el comportamiento del modelo usando topologías diversas como líneas, árboles y grafos dispersos. Además, debido a la complejidad de ciertas instancias, un método heurístico para encontrar soluciones iniciales de buena calidad es propuesto. El documento está organizado de la siguiente forma. Se inicia con la identificación de los objetivos del trabajo y con la descripción del problema. En el capítulo 3, se introduce formalmente al problema LPP junto con su notación y formulación entera. En el capítulo siguiente se presenta los resultados obtenidos de los experimentos computacionales. Finalmente, en el último capítulo se detallan las conclusiones.

Capítulo 2

Descripción del componente desarrollado

2.1. Objetivos

2.1.1. General

1. Estudiar y adaptar los resultados reportados en [1] y [2] para ser aplicados en redes derivadas del Sistema Integrado Metropolitano de Transporte Público de Quito.

2.1.2. Específicos

1. Realizar una revisión del estado del arte sobre el Problema de Planificación de Líneas y Frecuencias.
2. Implementar un modelo de programación lineal entera para el Problema de Planificación de Líneas y Frecuencias usando el solver GUROBI.
3. Implementar un método de solución heurístico para el Problema de Planificación de Líneas y Frecuencias.
4. Conducir experimentos computacionales usando datos reales y simulados, que se apegan a la realidad del sistema de movilidad de Quito

5. Diseñar instancias aleatorias de prueba (líneas, árboles, grafos dispersos) para verificar el comportamiento de los métodos de solución.

6. Reportar los resultados obtenidos.

2.2. Alcance

En el presente trabajo, se pretende implementar un modelo de programación lineal entera y diseñar un método de solución heurístico para el Problema de Planificación de Líneas y Frecuencias. Se usarán las diferentes ideas y resultados expuestos en trabajos anteriores respecto a la planificación de transporte y métodos de solución disponibles para problemas de programación lineal entera. En particular, usaremos y extenderemos los resultados expuestos en [1] con el apoyo de un segundo artículo base [2].

Esencialmente, se trata de conocer cómo es el comportamiento del problema en diferentes topologías de redes derivadas del Sistema de Transporte de Quito a través de experimentos computacionales. Además, es importante mencionar que no se pretende obtener una nueva formulación del Modelo de Cubrimiento de Demanda para el Problema de Planificación de Líneas y Frecuencias (enunciado en [2]), sino únicamente replicar los resultados existentes a un caso particular junto con el método heurístico desarrollado.

En esta componente se aplicarán conocimientos obtenidos en materias y teoría afines a la programación lineal, teoría de grafos y optimización en redes e implementación de modelos de programación entera.

2.3. Marco teórico

2.3.1. Programación lineal entera

Se llama Programación lineal a la formulación algebraica de una función lineal de varias variables que debe ser optimizada y que está sujeta a cierto número de restricciones, también lineales, expresadas mediante

ecuaciones o inecuaciones. De forma matricial, se la puede representar mediante la siguiente expresión estándar:

$$min\{c^T x | Ax \geq b, x \in \mathbb{R}^n\} \vee max\{c^T x | Ax \leq b, x \in \mathbb{R}^n\},$$

donde:
$c \in \mathbb{R}^n, b \in \mathbb{R}^m$ son vectores de coeficientes dados.
$x \in \mathbb{R}^n$ es el vector de variables a determinar.
$A \in \mathbb{R}^{m \times n}$ es una matriz de coeficientes dada.

De forma general, se tiene que $c^T x$ es la función objetivo que pretende ser optimizada, y al conjunto $P = \{x | x \in \mathbb{R}^n \wedge Ax \leq b\}$ se lo conoce como poliedro o región de factibilidad del problema. El vector x tiene entradas $x_i \in \mathbb{R}\ (i = 1, ..., n)$ que son las denominadas variables de decisión.

Cuando se requiere que los valores de las variables de decisión tomen valores estrictamente enteros, entonces se trata de un **problema de programación lineal entera**. Su expresión matricial, en el sentido de la minimización, es la siguiente:

$$min\{c^T x | Ax \geq b, x \in \mathbb{Z}^n\}$$

En otras ocasiones, se necesita que los variables tomen únicamente valores de cero o uno. En este caso, al modelo se lo conoce como **programación lineal binaria**.

$$min\{c^T x | Ax \geq b, x \in \{0, 1\}^n\}$$

Se puede demostrar que este tipo de problemas pertenecen a la clase de complejidad NP-completo. Sin embargo, en el sentido de maximización, se puede verificar que la siguiente relación se cumple:

$$max\{c^T x | Ax \leq b, x \in \mathbb{Z}^n\} \leq max\{c^T x | Ax \leq b, x \in \mathbb{R}^n\}$$

puesto que

$$\{x|Ax \leq b, x \in \mathbb{Z}^n\} \subseteq \{x|Ax \leq b, x \in \mathbb{R}^n\}$$

y siempre que $\{x|Ax \leq b, x \in \mathbb{R}^n\}$ sea acotado. La **relajación lineal** de un problema entero se obtiene cuando se eliminan las restricciones de integrabilidad de las variables.

Existen también otras caracterizaciones que tiene que ver con la optimalidad del problema y que están sujetas a las condiciones de la matriz de restricciones A. Esto es el caso de las matrices totalmente unimodulares, donde se puede asegurar que los vértices del poliedro P son enteros si b es entero.

Por otro lado, si A no es totalmente unimodular, hasta el momento no existe algoritmo polinomial que resuelva un problema de programación lineal entera. Por tal motivo, es necesario introducir otros métodos de solución que garanticen una respuesta adecuada. Para nuestro caso, explicaremos de forma general el funcionamiento de dos de los más estudiados: uno exacto y uno aproximado. El método exacto más utilizado es de un algoritmo iterativo conocido como "Branch and Bound"; mientras que los métodos heurísticos permiten encontrar soluciones factibles aproximadas en un tiempo polinomial.

2.3.2. Método de solución exacta: Branch and Bound

El algoritmo ”Branch and Bound” (ramificación y acotamiento) recibe su nombre en relación a las dos técnicas en la que se basa su desarrollo. Combina un método denominado ramificación que produce la apertura de nuevos submodelos, con otra técnica denominada acotamiento que básicamente identifica cotas en las que se encuentra la solución óptima. Es un procedimiento iterativo que en el peor de los casos puede ser de orden exponencial. Este es uno de los algoritmos pioneros en la resolución de problemas enteros siendo la técnica base y fundamental en los solvers actuales de optimización.

El proceso comienza cuando el algoritmo resuelve el problema relajado asociado. Si el resultado verifica las condiciones de integrabilidad del problema entero, se tiene la solución óptima. Si no es el caso, la iteración del

algoritmo debe entrar en ejecución. La **ramificación** consiste en dividir el problema entero en dos subproblemas adicionales obtenidos al incluir restricciones excluyentes sin eliminar soluciones enteras. Esto se logra, cuando en la solución de la relajación lineal se encuentra una variable x_i, con valor fraccionario h_i, entonces se pueden generar dos restricciones de la forma:

$$x_i \leq \lfloor h_i \rfloor$$

$$x_i \geq \lfloor h_i \rfloor + 1$$

Estas nuevas restricciones se imponen al modelo relajado para generar los dos nuevos subproblemas (en el caso de la maximización):

$$max\{c^T x | Ax \leq b, x \in \mathbb{R}^n, x_i \leq \lfloor h_i \rfloor\}$$

$$max\{c^T x | Ax \leq b, x \in \mathbb{R}^n, x_i \geq \lfloor h_i \rfloor + 1\}$$

Este proceso se repite para cada uno de los subproblemas formados. Si existe una nueva variable x_j con valor fraccionario d_j se impone nuevamente estas condiciones excluyentes según la forma establecida. Consecuentemente, si se mantiene este ciclo iterativo para cada uno de los subproblemas obtenidos, se garantiza que en alguna de las repeticiones, las variables tengan una solución entera. Al identificar la solución entera con mayor valor objetivo, se obtiene la solución del problema original.

Sin embargo, aunque este procedimiento de ramificación parecería ser bueno, si se ejecutase por si solo, no resulta eficaz debido a que el número de nodos en el proceso de ramificación depende del número de variables del modelo y que en el peor de los casos puede ser de orden exponencial. Por esta razón, se combina a la ramificación con la acotación para reducir significativamente el número de operaciones.

La **acotación** se basa en el principio básico de la contenencia de conjuntos y en la formulación de cotas inferiores y superiores para el valor objetivo en las que se puede asegurar que se encuentra la solución óptima. Estas cotas se obtienen de la resolución de las relajaciones lineales y de las soluciones enteras obtenidas en los subproblemas, de tal forma

que se van actualizando en cada una de las iteraciones. Este procedimiento nos permite descartar todos los subproblemas a ramificar cuyo valor ojetivo se encuentra fuera de dichas cotas.

Eventualmente, al considerar ambos métodos en conjunto, el número de subproblemas a ramificar en el primer proceso se ve reducido considerablemente, lo que implica también una reducción adecuada en el tiempo de resolución de problemas enteros. Sin embargo, no se puede asegurar que su tiempo de ejecución sea polinomial. De esta manera, el algoritmo de Branch and Bound se convierte en un método eficiente y potencialmente bueno.

Los siguientes pasos, en pseudocódigo, resumen las etapas del Branch and Bound para un problema de minimización:

Algorithm 1 Pasos generales del Branch and Bound

Require: *Un problema lineal entero* Π
Ensure: *Solución entera del problema*

1: $x \leftarrow$ solucion de la relajacion lineal del problema Π con valor objetivo z
2: **if** x es entero **then**
3: **return** x
4: **else**
5: Elegir x_i cuyo valor d_i sea continuo
6: Definir las restricciones: $x_i \leq [d_i] \wedge x_i \geq [d_i] + 1$
7: Cremos dos subproblemas $\Pi_1, \Pi_2 \subset \Pi$ tales que $\Pi_1 := \Pi \cup \{x | x_i \leq [d_i]\} \wedge \Pi_2 := \Pi \cup \{x | x_i \geq [d_i] + 1\}$
8: **for** $\Pi' \in \{\Pi_1, \Pi_2\}$ **do**
9: $x' \leftarrow$ solución de la relajación lineal del problema Π' con valor objetivo z'
10: **if** $z' < z$ **then**
11: $x \leftarrow x'$
12: $z \leftarrow z'$
13: $\Pi \leftarrow \Pi'$
14: Regresar al paso 2.
15: **end if**
16: **end for**
17: **end if**

2.3.3. Método de solución aproximada: Heurística

En muchas ocasiones, aunque el método de ramificación y acotamiento es bastante útil a la hora de enfrentarse a la programación entera, se

presentan problemas NP completos bastente fuertes y complicados de resolver. En efecto, cuando esto sucede, la optimización combinatoria propone estudiar técnicas que faciliten la resolución de estos problemas con el objetivo de alcanzar al menos una solución aproximada, que no difiera mucho de la óptima, en un tiempo polinomial.

En muchos estudios laborales, académicos o empresariales, es suficiente alcanzar un valor cercano que cumpla con todas las restricciones del modelo sin invertir muchos recursos computacionales o económicos.

Para conocer la eficiencia de una heurística, se define dos conceptos claves denominados **Relación absoluta y asíntotica de rendimiento** que básicamente miden cuán alejada está la solución encontrada con la heurística de la óptima en el peor de los casos. Así, formalmente se define a la relación absoluta de rendimiento R^H, y a la relación asintótica de rendimiento R^H_∞ como:

$$R^H = inf\{\gamma \geq 1 : \frac{z^H(I)}{Z^*(I)} \leq \gamma, \forall\, instancia\ I\ del\ problema\},$$

$$R^H_\infty = inf\{\gamma \geq 1 : \exists\, n\ tal\ que \frac{z^H(I)}{Z^*(I)} \leq \gamma, \forall\, instancia\ I\ del\ problema\ que\ cumple$$

$$z^*(I) \geq n\},$$

donde $z^H(I)$ es la solución encontrada por la heurística y $z^*(I)$ es la solución óptima

Además, se tiene que:

$$R^H_\infty \leq R^H$$

Capítulo 3

Modelo de Planificación de Líneas y Frecuencias

En este capítulo se introduce el modelo de programación lineal entera para el Problema de Planificación de Líneas y Frecuencias expuesto en [2] y un método heurístico de solución. Algunos criterios de factibilidad serán presentados.

3.1. Modelo de Cubrimiento de Demanda

El Modelo de Cubrimiento de Demanda (DCM) es una formulación matemática para resolver el Problema de Planificación de Líneas y Frecuencias en una red de transporte público con una demanda de pasajeros que ha sido enrutada a priori. Se denomina cubrimiento debido a que se debe identificar un conjunto de líneas, con sus respectivas frecuencias, de tal forma que se garantice el traslado de todos los pasajeros entre cada una de las estaciones del sistema, es decir, se cubra a todos ellos.

Partimos entonces de la formulación matemática del DCM. Este planteamiento se encuentra en [2]. Tenemos las siguientes notaciones:

- Se considera una red de transporte público como un grafo dirigido $D = (V, A)$, donde cada estación de autobuses está representada por un nodo $v \in V$ y los arcos representan enlaces directos entre

estaciones, es decir, $(u, v) \in A$ si y solo si un autobús puede visitar la estación asociada al nodo v directamente después de la estación asociada al nodo u.

- Un **modo de transporte** es un tipo específico de autobús. Así, se define M como el conjunto de todos los modos de transporte del sistema. Cada modo de transporte $m \in M$ tiene una capacidad unitaria específica $k_m \in \mathbb{Z}_+$. Además, para cada $m \in M$, se identifican ciertas estaciones denominadas **terminales**, donde los autobuses del modo m pueden iniciar o finalizar una ruta de servicio, y **estaciones de giro** o **no giro** adecuadas.
- Se establece únicamente el uso de línea cerradas para cada modo de transporte. Una **línea cerrada** es un ciclo (camino) dirigido que parte de un terminal y vuelve al mismo. Para cada $m \in M$, sea L^m, el conjunto de líneas seleccionadas (a priori) que potencialmente pueden implementarse en el sistema de transporte. Denotamos por $L = \cup_{m \in M} L^m$ el conjunto de todas las líneas posibles y por L_a^m al conjunto de líneas del modo m que utilizan el arco a.
- Para cada línea $l \in L^m$, con $m \in M$, se imponen costos variables $c_l \in \mathbb{R}_+$ que es el costo de un solo viaje a través de esta línea. De forma similar, se establecen costos fijos $K_l \in \mathbb{R}_+$.
- La demanda de transporte se expresa en términos de la matriz origen-destino $D = (d_{ij}) \in \mathbb{Z}_+^{V \times V}$. Cada entrada d_{ij} indica el número de pasajeros que viajan de la estación i a la estación j dentro de un cierto horizonte de tiempo a través de las conexiones posibles del sistema.
- Se asume que cada pasajero ha sido enrutado a lo largo de alguna ruta específica en un procesamiento previo, lo que resulta en una demanda de transporte agregada g_a en cada arco a de la red. En las siguientes páginas se explica a mayor detalle la forma en que se obtiene esta demanda agregada.
- Todo arco $a \in A$ con demanda positiva g_a está cubierto por al menos una línea, de lo contrario la instancia es infactible.

El análisis en cuestión se realizará solamente para líneas que son cerradas y simétricas, es decir, para el caso específico en que un autobús

recorre ciertas estaciones en el sentido de "ida" desde un terminal hasta una estación de giro y las mismas estaciones de "regreso" hacia el mismo terminal. Así, si una línea parte de un terminal u hasta otra estación v y recorre las estaciones i, j, k en el transcurso, entonces, en el recorrido de regreso a su terminal de partida también recorre las estaciones i, j, k en el sentido contrario. El siguiente diagrama muestra de mejor manera la idea.

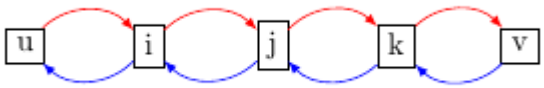


Figura 3.1: Ejemplo de línea cerrada

Por tanto, la línea l visitaría los nodos: $l : (u, i, j, k, v, k, j, i, u)$ con los arcos en rojo para el sentido de ida y los de color azul para el regreso.

La simetría de este tipo de líneas puede aprovecharse para reducir el problema a una versión equivalente en un grafo no dirigido $G = (V, E)$. Esto se logra al sustituir pares de arcos antiparalelos $(u, v), (v, u) \in A$ con aristas no dirigidas $\{u, v\}$. Las demandas agregadas en estas aristas están dadas por:

$$\hat{g}_{\{u,v\}} = max\{g_{(u,v)}, g_{(v,u)}\}$$

para segurar que la demanda de ambos sentidos pueda ser cubierta.

En esta versión no dirigida, las líneas corresponden a caminos simples no dirigidos en G, con los mismos costos que sus contrapartes dirigidas. Así, sea L_e^m el conjunto de líneas de modo m usando la arista e.

Usando una red de árbol, se considera el siguiente esquema para explicar la idea anterior (ver Figuras 3.2 y 3.3). Los arcos antiparalelos de color negro en D definen las conexiones entre las estaciones posibles $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$. Además, cada arco $(v_i, v_j) \in A$ tiene su demanda agregada cargada $g_{(v_i,v_j)}$. Los caminos en colores representan a las líneas $l \in L$. Se puede notar que las líneas retornan a su estación de partida por el mismo camino en el que salieron.

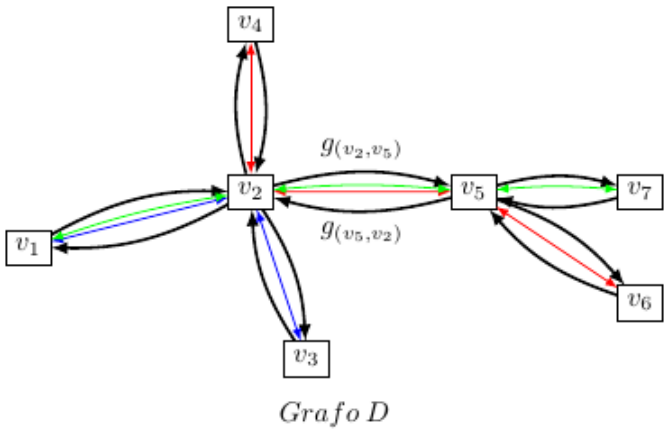


Figura 3.2: Grafo dirigido con líneas

En este caso, las siguientes líneas han sido consideradas:

$$l_1\ (azul) : (v_1, v_2, v_3, v_2, v_1)$$

$$l_2\ (rojo) : (v_4, v_2, v_5, v_6, v_5, v_2, v_4)$$

$$l_3\ (verde) : (v_1, v_2, v_5, v_7, v_5, v_2, v_1)$$

Una vez asumido que las líneas deben ser simétricas, el grafo D se transforma de forma equivalente en G, tal como se ve en la Figura 3.3. El conjunto de estaciones es el mismo y las aristas $e = \{v_i, v_j\}$ de color negro representan a cada par de arcos antiparalelos $a = (v_i, v_j)$ y $\bar{a} = (v_j, v_i)$ de A.

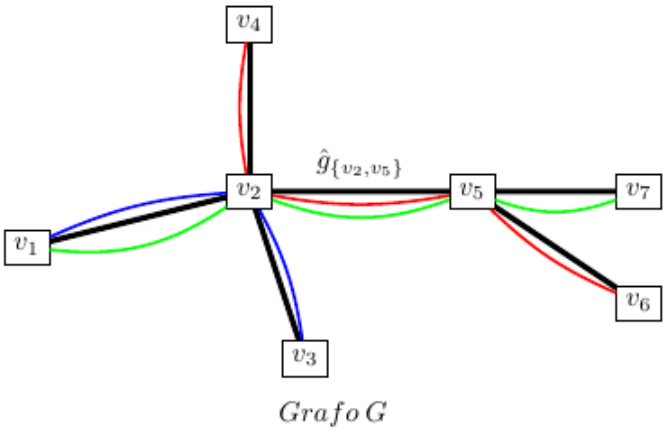


Figura 3.3: Grafo no dirigido con líneas

En el que las líneas equivalentes son:

$$l_1\,(azul) : (v_1, v_2, v_3)$$

$$l_2\,(rojo) : (v_4, v_2, v_5, v_6)$$

$$l_3\,(verde) : (v_1, v_2, v_5, v_7)$$

y la demanda agregada se toma según la definición anterior. En el caso del ejemplo: $\hat{g}_{\{v_2,v_5\}} = max\{g_{(v_2,v_5)}, g_{(v_5,v_2)}\}$.

Una vez que se propone esta equivalencia, se formula un Modelo de Cubrimiento de Demanda para el problema de Planificación de Líneas y Frecuencias para el caso de un grafo no dirigido $G = (V, E)$. Se definen entonces las siguientes variables de decisión:

- f_l: variable entera no negativa que representa la frecuencia asignada a la línea $l \in L$.
- y_l: variable binaria que es igual a uno siempre que la línea $l \in L$ es considerada en la planificación; caso contrario, toma el valor de cero.

Consideramos ahora el siguiente modelo para planificación de líneas y frecuencias (DCM):

$$min \sum_{m \in M} \sum_{l \in L^m} (c_l f_l + K_l y_l) \tag{3.1}$$

sujeto a las restricciones:

$$\sum_{m \in M} \sum_{l \in L_e^m} k_m f_l \geq \hat{g}_e, \qquad \forall\, e \in E, \tag{3.2}$$

$$0 \leq f_l \leq f_l^{max} y_l, \qquad \forall\, l \in L^m,\ m \in M, \tag{3.3}$$

$$f_l \in \mathbb{Z}_+, \qquad \forall\, l \in L^m,\ m \in M, \tag{3.4}$$

$$y_l \in \{0, 1\}, \qquad \forall\, l \in L^m,\ m \in M. \tag{3.5}$$

- La función objetivo (3.1) pretende minimizar los costos totales de operación (fijos y variables). Podemos asumir que los costos de ope-

ración variables en el sistema dependen de la frecuencia de cada línea. Los costos fijos dependen únicamente de la condición lógica de activación, es decir, cuando la línea $l \in L^m$ es considerada o no en la planificación. Como se puede ver en la notación, los valores de estos costos influyen en la sumatoria total solamente cuando se tenga que $f_l > 0$ y $y_l = 1$.

- La restricción (3.2) garantiza que la demanda de transporte agregada sea cubierta en cada arista, es decir, se asegura que la capacidad total de todas las líneas elegidas de cada uno de los modos de transporte logren cubrir la cantidad total de pasajeros que utilizan dicha arista.

- La restricción (3.3) asegura que las frecuencias de cada línea no superen la cota superior máxima. Es importante mencionar que, tal como se muestra en la desigualdad, solo las frecuencias de las líneas elegidas $(y_l = 1)$ están acotadas superiormente. Cuando $y_l = 0$, se exige también que $f_l = 0$. Esto asegura que no sucedan casos para los que una línea no elegida, tenga una frecuencia mayor a cero.

- Las restriccciones (3.4) y (3.5) imponen que f_l y y_l sean enteras estrictamente.

Cuando se trabaja con un solo modo de transporte $(|M| = 1)$ o si se considera que la capacidad de los autobuses es la misma en todos los modos de transporte, el modelo DCM puede ser simplificado. Asi, si $k_m = k$, $\forall\, m \in M$, un modelo de cubrimiento de demanda con flota homogénea (DCM-HF) es considerado. En este caso, se puede dividir a la restricción (3.2) por k y tomar el lado derecho de esta desigualdad por:

$$\hat{h}_e = \lceil \frac{\hat{g}_e}{k} \rceil$$

Por otro lado, como es de nuestro interés reportar el número total de líneas elegidas una vez que el modelo se resuelve, denotamos al conjunto $L' \subset L$ como el conjunto solución.

$$L' = \{l \in L \mid f_l > 0 \wedge y_l = 1, \forall\, l \in L^m,\, m \in M\}$$

3.2. Condiciones de factibilidad

En algunos casos, el modelo DCM puede no tener solución. Por ello, en esta sección se identifican algunas condiciones necesarias para que el poliedro P no sea vacío.

Lema 1: Si existe una arista $e \in E$ con $\cup_{m \in M} L_e^m = \emptyset$, entonces el DCM no tiene solución.

Demostración: Sea $e \in E$. Si se tiene que $\cup_{m \in M} L_e^m = \emptyset$ entonces las restricción (3.2) puede ser expresada como:

$$0 \geq \hat{g}_e$$

Si al menos un pasajero ha sido enrutado por dicha arista, entonces $\hat{g}_e > 0$, lo que muestra claramente que el DCM no puede producir una solución factible.

■

La Figura 3.4 muestra un ejemplo de red para el cual el DCM es infactible. En ella, asumiendo que las líneas parten y regresan del nodo 2, se puede notar que ninguna de las líneas en colores cubre la conexión $\{2, 1\}$. Por otro lado, la arista $\{14, 5\}$ es cubierta por las líneas azul y verde.

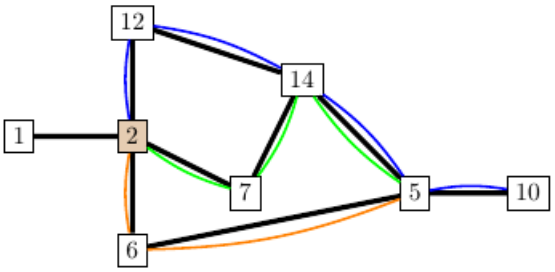


Figura 3.4: Red no cubierta en su totalidad para el DCM

Otra condición importante implica considerar sistemas de transporte que dispongan de redes conexas, es decir, que se establezca al menos un camino entre cada par de estaciones. Con esto, los pasajeros podrán movilizarse a lo largo de toda la red y por las aristas que requieran.

Lema 2: Sea G un grafo disconexo y C_1, C_2 dos componentes conexas. Si

$i \in C_1$, $j \in C_2$ y $d_{ij} > 0$ entonces el System Split no puede encontrar la demanda agregada sobre la red.

Demostración: Es fácil notar que si no existe un camino entre algún par de nodos $i, j \in V$ con $d_{ij} > 0$, entonces dichos pasajeros no pueden ser enrutados. Esto impide que las demandas agregadas sean calculadas y por lo tanto no se pueda obtener un plan de líneas.

■

La siguiente figura (3.5) muestra una red no conexa. En ella, las estaciones $\{10, 9, 13\}$ no se pueden conectar con ninguna otra estación de la red. Por ejemplo, los pasajeros que quisieran ir desde 13 a 7 no lo pueden hacer puesto que no hay un camino posible entre esos nodos.

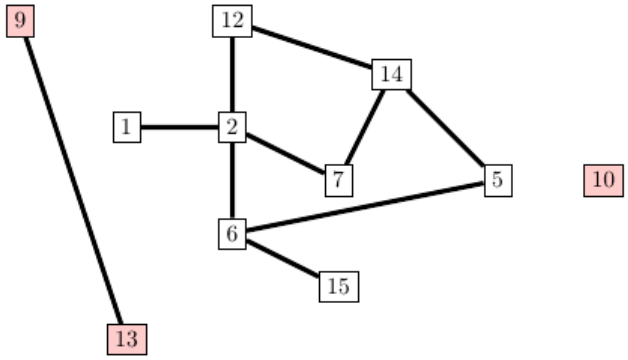


Figura 3.5: Red no conexa para el DCM

Otro de las condiciones para obtener factibilidad en el DCM se basa en la capacidad total de las líneas. Parte también de la restricción (3.2) en donde se pretende garantizar el traslado de todas las personas que se describe en la matriz OD. En este sentido, hay que asegurar que la capacidad de los autobuses de todas las líneas que cubren cada arista sea suficiente.

Lema 3: Una condición necesaria para la factiilidad del DCM es:

$$\sum_{m \in M} \sum_{l \in L_e^m} k_m f_l^{max} \geq \hat{g}_e, \qquad \forall e \in E$$

Por tanto, si no se cumple la desigualdad anterior, implica que aunque se asigne la frecuencia máxima a cada una de las líneas posibles de

todos los modos de transporte, la capacidad total de todo ese grupo de líneas no alcanza para transportar a todas las personas que utilizan esa conexión. Así, el DCM también se vuelve infactible. En este sentido, se debe considerar valores adecuados de k_m de tal forma que la capacidad de los autobuses pueda satisfacer a toda la demanda de pasajeros.

3.3. Heurística para el DCM

En esta sección, se propone un método heurístico sencillo que brinda soluciones que no están muy alejadas de la solución entera óptima. Su uso dependerá del comportamiento del método exacto utilizado para la solución del DCM.

La idea base del algoritmo consiste en la resolución iterativa de la relajación lineal del DCM redondeando algunos variables que son cercanas a su valor entero. En efecto, se comienza con la solución del problema relajado y se obtiene su solución fraccionaria (f^*, y^*). Para alguna línea $l \in L$ con valor fraccionario $f_l^* > 0$, se fija $f_l = \lceil f_l^* \rceil$, es decir, se toma la función techo de la solución encontrada. Luego, se resuelve nuevamente el modelo relajado. Este procedimiento se repite mientras existan variables con solución fraccionaria. Al terminar el proceso, se asegura la inclusión de a lo más $|L|$ restricciones de tal forma que se obtiene una solución para el problema entero. El algoritmo siguiente resume los pasos propuestos, en pseudocódigo.

Algorithm 2 Heurística para el DCM

Require: Modelo DCM

Ensure: Una solución aproximada (f, y) del DCM

$\Pi \leftarrow$ Relajación lineal del DCM.

do

$(f^*, y^*) \leftarrow$ Solución del problema Π

Para algún $l \in L$ con f_l^* fraccionaria

Añadir al problema Π la restricción: $f_l = \lceil f_l^* \rceil$

while Π tenga soluciones fraccionarias

end do while

return (f, y)

Capítulo 4

Resultados Computacionales

4.1. Implementación

4.1.1. Líneas factibles y costos

Las líneas factibles para cada modo de transporte se crean en base a los estaciones que son determinadas como terminales, de giro o no giro. Así, cada nodo tiene asignado un identificador que indica si la estación asociada es un terminal (t), un nodo en el que la unidad puede girar (g) o una estación en la que no lo puede hacer (ng). Estos indicadores están ligados a características técnicas de cada estación.

Tal como se habló en la formulación del DCM, solo las líneas cerradas simétricas serán consideradas en este trabajo. Cada línea debe partir desde un terminal y avanzar hasta una posible estación de giro u otro terminal para que pueda regresar a la misma terminal de partida. Bajo esta estructura, todas las líneas factibles serán incluidas en L

Por ejemplo, en una red de 15 nodos y 2 modos de transporte (ver Figura 4.1), se tiene:

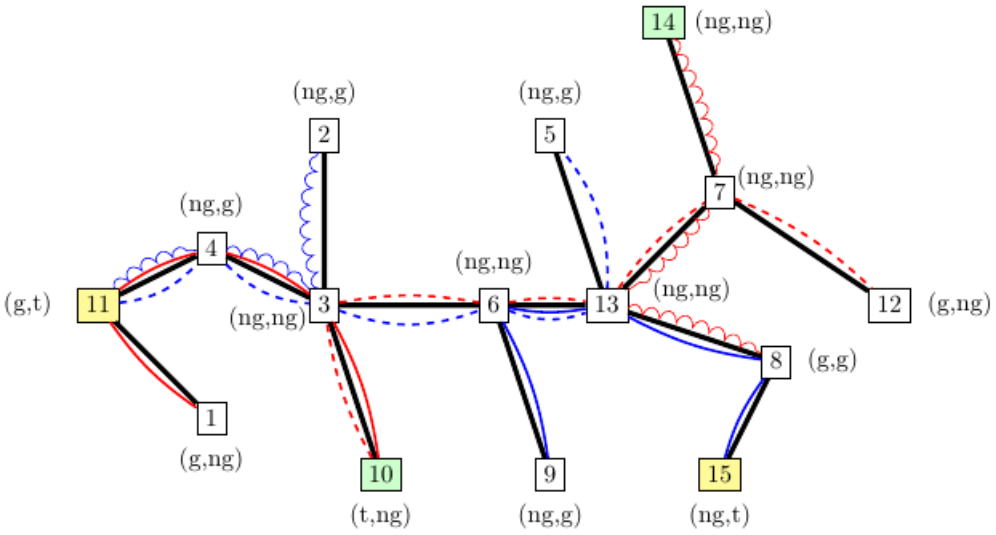


Figura 4.1: Ejemplo de red con dos modos de trasporte

Estaciones terminales del modo uno (verdes): $\{10, 14\}$. Estaciones terminales del modo dos (amarillas): $\{11, 15\}$. La tupla (a, b) corresponde a las identificaciones creadas con a que representa al tipo de estación para el modo uno, mientras que la segunda componente b representa el tipo de estación para el modo dos. Las líneas en color rojo corresponden a líneas posibles del modo uno mientras que las de color azul son líneas factibles para el modo dos.

Por otra parte, los costos de las líneas vienen dados por las dos categorías antes mencionadas: variables y fijos. Los costos de operación variables se calculan en función de la distancia total que recorre cada línea. En términos reales, se esperaría que esta distancia pueda ser medida entre cada par de estaciones, sin embargo, para nuestros experimentos dichos valores fueron simulados. A este valor de distancia se lo multiplica por el costo de kilómetro previamente conocido para cada uno de los modos de transporte. Además, el costo de cada línea será considerado como dos veces este producto puesto que tomamos líneas simétricas cerradas y se asume que la distancia desde la estación i a la j es la misma que de la estación j a la i. El siguiente gráfico (ver Figura 4. 2) representa a una red con 10 estaciones para un modo de transporte. Los valores en cada arista representan a la distancia entre los nodos que la forman.

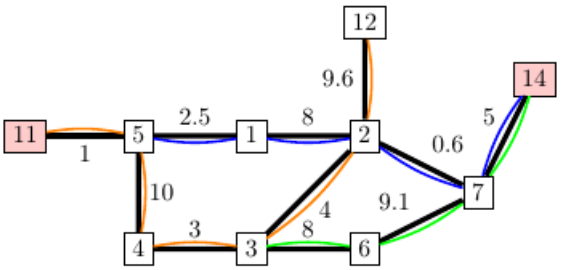


Figura 4.2: Red con distancias entre estaciones

Asumiendo que las estaciones terminales son $\{11, 14\}$ de color rojo, podemos considerar a las líneas que ahí se muestran según los diferentes colores. Si asignamos un costo de kilómetro con el valor de 4, los costos estarían dados por:

$$l_1\ (verde) : (14, 7, 6, 3),\ c_{l_1} = 176{,}8$$

$$l_2\ (anaranjado) : (11, 5, 4, 3, 2, 12),\ c_{l_2} = 220{,}8$$

$$l_3\ (azul) : (14, 7, 2, 1, 5),\ c_{l_3} = 88{,}8$$

Los costos de operación fijos se activan cuando la línea l del modo m es elegida y lo hacen por una sola vez. Los datos de estos costos dependen también de las condiciones técnicas que, por ejemplo, pueden representar el costo del conductor.

4.1.2. Matrices OD

Se asume que la Matriz Origen Destino es dada. De forma general, esta matriz representa a la cantidad total de pasajeros que tienen que ser transportados en la red. Así, se tiene para la matriz OD $D \in \mathbb{Z}^{n \times n}$, donde $n = |V|$ es el número total de estaciones, cada una de las entradas d_{ij} representa la cantidad de pasajeros que desean ir desde la estación i hasta la estación j. Claramente, esta matriz no es necesariamente simétrica debido a que la cantidad de pasajeros que desean ir de i a j no puede ser igual a la de los pasajeros que quieren ir en el sentido contrario (de j a i). De forma general, se tiene que:

$$d_{ij} \neq d_{ji}, \quad \forall\, i, j \in V,\ i \neq j$$

$$d_{ii} = 0, \quad \forall\, i \in V$$

En la industria del transporte y modelización, estas matrices han sido una forma estándar para estimar la demanda del transporte. Para este trabajo, las matrices OD son tomadas de [1] y [2].

La información que se presenta en la matriz OD nos sirve para calcular la demanda agregada de pasajeros en cada arista del grafo. Para ello, tal como se presentó en la literatura estudiada, se utiliza un método denominado System Split que culmina con una división total de la demanda.

System Split, demanda agregada y camino más corto

El System Split ha sido considerado un método efectivo a la hora de enrutar a los pasajeros sobre una red. Para ello, para cada par origen destino se debe encontrar el camino más corto entre estas estaciones y cargar la cantidad de pasajeros asociada al par OD a todas las aristas en el camino. Por ejemplo, en la red de la figura (4.3), si d_{ij} y d_{uv} son la cantidad de pasajeros que deben ser transportados desde i hasta j y desde u hasta v, respectivamente, se tiene que existen aristas intermedias en cada una de estas rutas y que son compartidas en ambos trayectos.

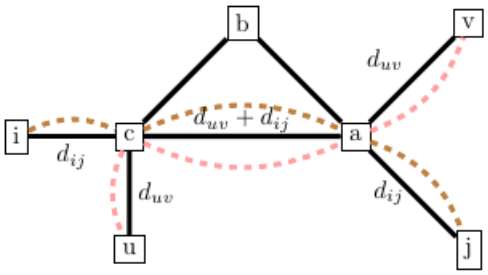


Figura 4.3: Red con caminos más cortos y demanda agregada

El camino más corto desde i hasta j (camino marrón) es: $i - c - a - j$, mientras que le camino más corto de u hasta v (camino rosado) es $u - c -$

$a-v$. Ambos trayectos comparten la arista $\{c, a\}$ por lo que a esta conexión se debe sumar la cantidad de pasajeros de cada uno de los pares origen destino. Para el resto de aristas se debe cargar la demanda según el par origen destino que lo requiera.

Para encontrar el camino mas corto entre cada par de estaciones, se puede utilizar algún método clásico de la literatura. Frecuentemente, se puede optar por el algoritmo de Dijkstra que es un procedimiento eficiente. En nuestro caso, utilizamos la función $shortest\_path()$ del módulo $networkx$ de Python. Esta es una librería dirigida al tratamiento de grafos que nos ayuda a optimizar tanto el tiempo como los recursos computacionales. El siguiente algoritmo, en pseudocódigo, muestra las operaciones para calcular la demanda agregada en cada arista.

Algorithm 3 Demanda agregada

Require: Una Matriz OD $D = (d_{ij}) \in \mathbb{Z}^{nxn}$, un grafo no dirigido $G = (V, E)$ con distancias $c : E \to \mathbb{R}_+$.

Ensure: Demanda agregada para cada arista $e \in E$

```
for (u, v) ∈ E do
    g_uv = 0
end for
for i ← 1, |V| do
    for j ← 1, |V| do
        if i ≠ j then
            t ← Camino más corto desde i hasta j según las distancias c
            for k ← 1, |t| − 1 do
                if (t_k, t_{k+1}) ∈ E or (t_{k+1}, t_k) ∈ E then
                    g_{t_k t_{k+1}} ← g_{t_k t_{k+1}} + d_ij
                end if
            end for
        end if
    end for
end for
for (p, q) ∈ E do
    ĝ_{p,q} ← max{g_pq, g_qp}
end for
return ĝ
```

4.1.3. Diseño de redes

Para la implementación, utilizaremos algunas topologías de redes de transporte que se derivan del Sistema de Transporte Público de Quito. Se crean tres tipos de redes principales: líneas, árboles y grafos generales. Para cada uno de ellos, se pretende verificar cómo actúan los algoritmos de solución mencionados.

Líneas

Una línea consiste de un grafo conexo y acíclico que conecta a sus nodos de forma sucesiva. Por ejemplo, ver la siguiente figura (4.4).

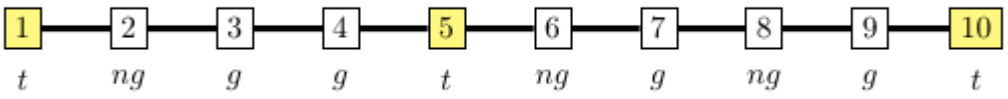


Figura 4.4: Red de transporte: Línea

De forma general, este tipo de líneas se asocian al grafo $G = (V, E)$ con:

$$V = \{i | i = 1, ..., n\}$$

$$E = \{(i, i+1), | i = 1, ..., n-1\}$$

donde $n = |V|$ es el número de estaciones de la red y $|E| = n - 1$

Árboles

Un segundo tipo de red analizada son las estructuras de árboles. En este tipo, las aristas se crean de forma indistinta, de tal forma que se garantiza la existencia de un camino único entre cada par de estaciones, la no presencia de ciclos y se asegura la conexidad.

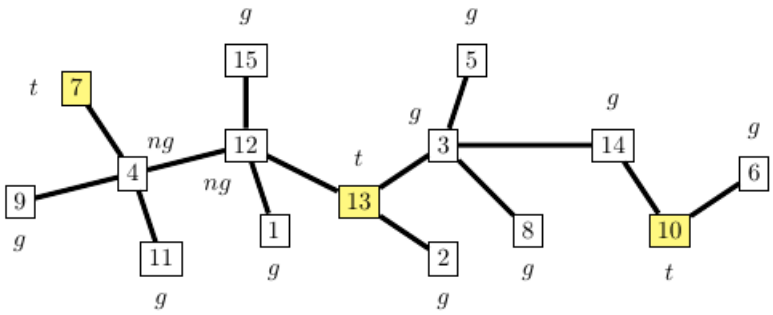


Figura 4.5: Red de transporte: Árbol

De forma similar al caso anterior, el conjunto de aristas y nodos viene dado por:

$$V = \{i | i = 1, ..., n\}$$

$$E = \{(i, j) | i, j = 1, ..., n, i \neq j\}$$

con $|E| = n - 1$ y $G = (V, E)$ conexo.

Para crear estas redes, se constuyen n puntos aleatorios en el plano cartesiano de tal forma que cada uno de ellos se asocia con una estación de la red. Además, las aristas se incluyen también de forma aleatoria de tal manera que se crea el árbol. La distancia entre cada par de estaciones viene dada por la distancia euclidea entre ellas.

Grafos generales

En este caso, se crea un grafo $G = (V, E)$, con $n \leq |E| \leq \frac{n(n-1)}{2}$

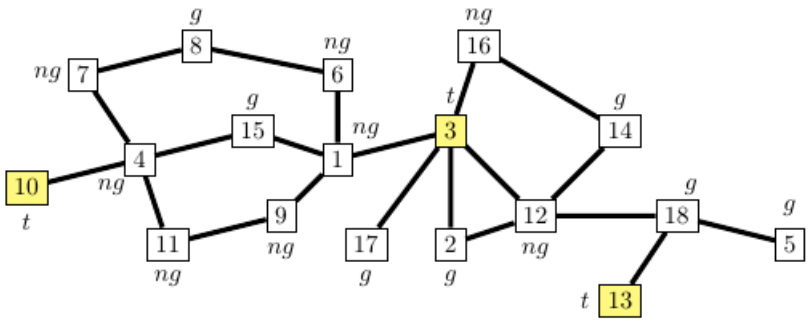


Figura 4.6: Red de transporte: Grafo disperso

El grafo es creado de la misma forma que el caso de árboles, pero se añade una cantidad finita de aristas para garantizar la inclusión de ciclos. Las distancias entre estaciones también se calculan con el mismo procedimiento.

Es importante mencionar que la demanda agregada de las aristas para estos tres tipos de redes son calculadas según el método del System Split, mencionado anteriormente. Para las estructuras de líneas y árboles, el camino entre cada par de nodos es único, mientras que para los grafos generales se debe encontrar el camino más corto.

4.2. Reporte de Tablas

A continuación, se muestran algunas tablas que resumen los resultados obtenidos de los experimentos computaciones para el modelo DCM. Se tiene en cuenta lo siguiente:

- Se realizaron implementaciones para los tres tipos de redes de transporte mencionadas: Líneas, árboles y grafos generales. Para este último caso, dada la complejidad del sistema, se considera la heurística descrita. Además, la solución obtenida heurísticamente es incluida como solución inicial para el modelo entero.
- Se ejecuta la implementación del modelo en cada una de las redes tanto para uno como para dos modos de transporte.
- Se utilizó el software Gurobi (versión 9.1.2 con licencia académica) a través de la interfaz de Python. Además, se empleó un computador con procesador Intel Core I5 4200U con 1.60-2.30 GHz, 4 GB de memoria RAM y sistema operativo Windows 8.1.
- Se fijó en 300 segundos el tiempo máximo de cálculo.
- Se emplearon las matrices OD obtenidas de [1] y [2]. La instancia bajo el nombre n_i, corresponde a la $i-ésima$ matriz OD con n estaciones.
- Se asignaron los siguientes valores fijos de frecuencia máxima, capacidad de las unidades, costos fijos y costo por kilómetro para cada

modo de transporte. Estos valores garantizan la factibilidad del DCM y se apegan a las condiciones técnicas reales.

$\|M\|$	**Datos**				
	Modo	**Costo por km.**	**Costos Fijos**	**Frec. Máx**	**Capacidad**
1	1	3	20	30	180
2	1	3	20	30	180
	2	4	22	30	210

- Se sabe que en el caso de árboles y líneas, el número de aristas cumple con $|E| = |V| - 1$ por lo que no es necesario detallarlo. Mientras que, para el caso de grafos generales, si se lo hace.
- La cantidad de pasajeros a transportar para cada grupo de instancias se reporta en las siguiente tabla:

$\|V\|$	$\sum_{u,v \in V} d_{uv}$
30	59316
40	79088
50	98860
60	118632
70	138404
80	158175
90	177947
100	197719

La cantidad de líneas factibles ($|L|$), número de variables, número de restricciones, valor (costo) objetivo, gap (brecha de optimalidad), tiempo, cantidad de líneas elegidas ($|L'|$) y la sumatoria de las frecuencias obtenidas ($\sum_{l \in L'} f_l$) son reportados.

Red de transporte: Líneas									
Instancia	$\|M\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l \in L'} f_l$
30_1	1	54	108	83	7060.08	0.00	3.45	12	116
	2	93	216	137	7048.72	0.01	47.29	12	100
30_2	1	54	108	83	7338.30	0.00	3.98	11	92
	2	93	216	137	7338.30	0.01	10.85	11	92
30_3	1	54	108	83	7384.78	0.00	2.63	11	95
	2	93	216	137	7384.78	0.00	29.80	11	95
30_4	1	54	108	83	7724.38	0.00	5.68	10	142
	2	93	216	137	7723.75	0.01	138.05	11	123
30_5	1	54	108	83	7723.50	0.00	2.38	11	91
	2	93	216	137	7713.65	0.00	32.01	11	90
40_1	1	72	144	111	12510.40	0.01	6.51	17	146
	2	123	288	183	12510.40	0.01	139.56	17	139
40_2	1	72	144	111	13530.08	0.00	2.75	15	122
	2	123	288	183	13530.08	0.00	13.10	15	122
40_3	1	72	144	111	12752.44	0.01	1.10	16	120
	2	123	288	183	12752.44	0.01	193.48	16	126
40_4	1	72	144	111	12555.37	0.00	10.19	14	147
	2	123	288	183	12549.24	0.01	281.01	15	161
40_5	1	72	144	111	12782.73	0.00	2.62	14	121
	2	123	288	183	12767.94	0.01	18.84	14	120

Red de transporte: Líneas									
Instancia	$\|M\|$	$\|L\|$	**# Vars**	**# Restr**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l \in L'} f_l$
50_1	1	120	240	169	18742.07	0.00	7.71	17	145
	2	208	480	289	18742.07	0.01	34.41	17	145
50_2	1	120	240	169	19912.32	0.01	84.16	17	149
	2	208	480	289	19899.50	0.11	300.10	19	169
50_3	1	120	240	169	19036.13	0.01	90.60	17	162
	2	208	480	289	19004.01	0.08	300.12	17	148
50_4	1	120	240	169	18885.47	0.01	51.35	14	199
	2	208	480	289	18869.95	0.04	300.12	16	175
50_5	1	120	240	169	18796.97	0.01	11.05	18	175
	2	208	480	289	18792.83	0.07	300.10	19	151
60_1	1	180	360	239	24014.16	0.04	300.11	21	191
	2	315	720	419	24007.24	0.12	300.08	22	206
60_2	1	180	360	239	24673.81	0.01	82.13	24	193
	2	315	720	419	24673.81	0.17	300.10	24	176
60_3	1	180	360	239	26824.79	0.01	14.04	23	203
	2	315	720	419	26805.60	0.04	300.09	23	175
60_4	1	180	360	239	25149.41	0.02	300.10	22	177
	2	315	720	419	25116.73	0.12	300.08	24	186
60_5	1	180	360	239	24982.46	0.00	52.39	24	201
	2	315	720	419	24976.04	0.12	300.09	23	178
70_1	1	210	420	279	33879.84	0.01	65.73	26	208
	2	365	840	489	33857.86	0.05	300.14	27	210

Red de transporte: Líneas									
Instancia	$\|M\|$	$\|L\|$	**# Vars**	**# Restr**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l \in L'} f_l$
70_2	1	210	420	279	33412.15	0.00	10.22	22	218
	2	365	840	489	32875.23	0.14	300.11	26	263
70_3	1	210	420	279	35175.90	0.03	300.07	23	227
	2	365	840	489	35170.05	0.18	300.09	26	222
70_4	1	210	420	279	34304.82	0.01	17.90	26	225
	2	365	840	489	34290.95	0.14	300.11	29	206
70_5	1	210	420	279	35275.45	0.03	300.03	26	224
	2	365	840	489	35205.63	0.05	300.05	29	236
80_1	1	288	576	367	43189.78	0.01	7.63	28	230
	2	504	1152	655	43208.71	0.13	300.04	30	235
80_2	1	288	576	367	45196.33	0.01	9.68	32	229
	2	504	1152	655	45191.59	0.12	307.44	33	253
80_3	1	288	576	367	45203.94	0.00	10.93	27	246
	2	504	1152	655	44881.32	0.18	300.13	30	275
80_4	1	288	576	367	44921.28	0.01	25.32	27	259
	2	504	1152	655	44923.25	0.14	300.19	30	247
80_5	1	288	576	367	44581.83	0.03	300.10	30	236
	2	504	1152	655	44594.08	0.17	300.13	32	230
90_1	1	378	756	467	54773.45	0.01	111.57	34	279
	2	665	1512	845	54813.95	0.24	300.15	36	265
90_2	1	378	756	467	53686.67	0.03	300.14	30	257
	2	665	1512	845	53679.50	0.18	300.11	34	273

Red de transporte: Líneas									
Instancia	$\|M\|$	$\|L\|$	**# Vars**	**# Restr**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l\in L'} f_l$
90_3	1	378	756	467	55091.64	0.03	300.19	36	261
	2	665	1512	845	55096.02	0.16	300.14	37	274
90_4	1	378	756	467	53546.42	0.01	247.36	32	258
	2	665	1512	845	53515.71	0.15	300.11	37	304
90_5	1	378	756	467	56742.34	0.03	300.08	33	275
	2	665	1512	845	56552.58	0.21	300.13	38	290
100_1	1	420	840	519	67000.40	0.01	84.08	36	283
	2	735	1680	939	67012.30	0.12	300.04	38	295
100_2	1	420	840	519	69994.95	0.02	300.08	33	285
	2	735	1680	939	69986.62	0.07	300.03	37	309
100_3	1	420	840	519	63742.86	0.01	284.97	33	317
	2	735	1680	939	63751.40	0.22	300.11	41	306
100_4	1	488	976	587	66742.54	0.02	300.10	33	282
	2	848	1952	1075	66711.94	0.18	300.09	38	309
100_5	1	488	976	587	71137.36	0.01	300.12	36	283
	2	848	1952	1075	71133.12	0.12	300.11	40	302
100_7	1	420	840	519	70708.70	0.01	91.95	37	285
	2	735	1680	939	70685.40	0.18	300.09	43	305
100_8	1	488	976	587	67963.98	0.02	300.12	34	297
	2	848	1952	1075	68048.77	0.27	300.11	41	326
100_9	1	420	840	519	69053.75	0.01	294.38	36	289
	2	735	1680	939	69059.09	0.17	300.12	40	303
100_10	1	420	840	519	71092.45	0.02	300.05	36	327
	2	735	1680	939	71077.62	0.18	300.06	39	306

Red de transporte: Árboles									
Instancia	$\|M\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l\in L'} f_l$
30_1	1	72	144	101	19749.54	0.01	2.30	23	170
	2	117	252	155	20166.38	0.01	57.57	25	163
30_2	1	78	156	107	19181.51	0.00	2.87	22	175
	2	117	252	155	19925.24	0.01	108.63	27	165
30_3	1	72	144	101	22098.99	0.01	3.45	23	175
	2	117	252	155	22648.08	0.00	168.50	27	171
30_4	1	75	150	104	18385.25	0.00	1.46	23	184
	2	117	252	155	18997.76	0.01	11.39	26	174
30_5	1	72	144	101	19833.47	0.00	1.53	22	178
	2	117	252	155	20203.81	0.01	28.21	24	169
40_1	1	108	216	147	27061.68	0.00	1.00	34	264
	2	162	360	219	28107.67	0.00	2.95	34	251
40_2	1	111	222	150	30144.31	0.00	1.30	33	258
	2	159	348	213	31582.27	0.05	300.11	34	244
40_3	1	114	228	153	30635.19	0.00	0.83	34	253
	2	159	348	213	31971.44	0.01	5.52	36	237
40_4	1	105	210	144	31839.91	0.00	0.47	31	250
	2	159	348	213	33012.03	0.00	3.59	32	238
40_5	1	114	228	153	29313.60	0.00	1.20	33	257
	2	162	360	219	30633.79	0.01	318.99	36	240
50_1	1	176	352	225	41765.85	0.02	300.12	40	333
	2	252	528	313	42852.34	0.11	300.10	43	312

Red de transporte: Árboles									
Instancia	$\|M\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l\in L'} f_l$
50_2	1	160	320	209	45867.81	0.00	25.74	33	303
	2	244	496	297	47387.37	0.04	300.08	37	297
50_3	1	160	320	209	43925.44	0.00	16.37	34	305
	2	244	496	297	45191.28	0.07	300.08	37	292
50_4	1	160	320	209	47767.07	0.08	300.10	38	306
	2	248	512	305	48920.77	0.20	300.12	41	301
50_5	1	168	336	217	46368.10	0.01	20.90	37	304
	2	248	512	305	48201.41	0.21	300.09	39	290
60_1	1	270	540	329	68332.98	0.00	82.28	48	387
	2	380	800	459	70800.04	0.17	300.11	54	392
60_2	1	245	490	304	71886.17	0.01	300.12	43	352
	2	385	820	469	73483.50	0.07	300.10	48	384
60_3	1	245	490	304	78986.88	0.02	300.11	42	344
	2	375	780	449	81543.79	0.12	300.14	45	345
60_4	1	255	510	314	66319.22	0.06	306.83	46	380
	2	385	820	469	67459.64	0.20	300.12	53	366
60_5	1	250	500	309	71489.16	0.06	300.10	46	354
	2	385	820	469	72893.57	0.13	300.13	49	358
70_1	1	285	570	354	89451.57	0.00	31.92	53	408
	2	435	900	519	91642.41	0.15	300.13	54	390
70_2	1	315	630	384	80456.26	0.00	19.03	57	512
	2	440	920	529	83204.47	0.08	300.10	61	533

Red de transporte: Árboles									
Instancia	$\|M\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l\in L'} f_l$
70_3	1	305	610	374	85392.48	0.00	70.29	56	443
	2	445	940	539	88886.12	0.24	300.11	61	438
70_4	1	295	590	364	87961.51	0.01	69.74	55	426
	2	440	920	529	91526.97	0.08	300.13	57	406
70_5	1	285	570	354	85366.36	0.01	25.61	53	440
	2	440	920	529	86760.19	0.06	300.13	55	414
80_1	1	384	768	463	99456.83	0.06	300.09	59	479
	2	588	1200	679	101539.32	0.11	300.17	64	452
80_2	1	384	768	463	96837.40	0.00	198.50	55	482
	2	594	1224	691	99050.55	0.25	300.13	62	458
80_3	1	414	828	493	109701.57	0.01	89.28	57	515
	2	594	1224	691	113348.18	0.13	300.16	65	481
80_4	1	414	828	493	105183.19	0.09	300.12	62	486
	2	594	1224	691	108548.54	0.26	300.12	67	460
80_5	1	402	804	481	101949.64	0.06	300.10	64	473
	2	594	1224	691	104380.81	0.18	300.13	64	454
90_1	1	546	1092	635	110554.69	0.05	300.12	71	526
	2	791	1652	915	114653.44	0.20	562.35	78	502
90_2	1	553	1106	642	113164.58	0.04	300.12	72	546
	2	791	1652	915	116136.41	0.30	300.11	78	530
90_3	1	525	1050	614	111483.23	0.01	300.17	70	521
	2	791	1652	915	114988.03	0.24	300.15	77	496

Red de transporte: Árboles									
Instancia	$\|M\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l \in L'} f_l$
90_4	1	553	1106	642	95719.47	0.00	60.36	71	610
	2	812	1736	957	98880.06	0.14	300.12	72	546
90_5	1	546	1092	635	104079.95	0.01	75.30	68	512
	2	784	1624	901	108104.68	0.21	300.12	74	489
100_1	1	630	1260	729	121662.28	0.01	135.20	81	653
	2	896	1904	1051	127299.27	0.19	300.13	85	629
100_2	1	609	1218	708	114241.92	0.02	300.14	76	627
	2	882	1848	1023	118337.11	0.24	442.74	83	598
100_3	1	595	1190	694	123869.32	0.03	355.43	75	599
	2	889	1876	1037	127882.61	0.16	300.12	80	554
100_4	1	704	1408	803	129021.15	0.01	300.32	78	598
	2	1024	2144	1171	133860.71	0.16	300.08	86	583
100_5	1	680	1360	779	123548.55	0.01	69.68	72	601
	2	1024	2144	1171	126952.07	0.31	300.12	85	626
100_7	1	630	1260	729	123714.65	0.03	300.13	84	673
	2	896	1904	1051	126966.58	0.26	300.13	93	619
100_8	1	712	1424	811	123490.75	0.05	300.10	80	605
	2	1016	2112	1155	127008.20	0.24	300.12	90	616
100_9	1	602	1204	701	121989.67	0.02	300.12	78	628
	2	882	1848	1023	127152.93	0.19	300.11	87	639
100_10	1	609	1218	708	121972.84	0.00	30.78	76	596
	2	889	1876	1037	125684.98	0.16	300.13	88	592

Red de transporte: Grafos generales										
Instancia	$\|M\|$	$\|E\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l \in L'} f_l$
30_1	1	38	2367	4734	2405	22071.30	0.65	300.10	23	137
	2	38	3330	8512	4294	22437.37	1.35	300.17	27	137
30_2	1	38	2276	4552	2314	20592.53	0.64	300.16	24	146
	2	38	3160	7832	3954	20772.64	1.69	300.14	26	144
30_3	1	38	2918	5836	2956	21627.85	0.48	300.16	22	136
	2	38	4030	10096	5086	21740.04	1.95	300.12	30	131
30_4	1	38	3145	6290	3183	21920.84	0.68	300.14	25	140
	2	38	4048	10612	5344	22432.18	1.92	300.19	25	132
30_5	1	38	2488	4976	2526	21951.47	0.73	300.15	23	143
	2	38	3313	8444	4260	22372.07	1.94	300.13	27	144
40_1	1	47	2310	4620	2357	20218.23	0.94	300.47	37	216
	2	47	2919	6824	3459	20830.84	1.63	300.13	36	208
40_2	1	47	2591	5182	2638	23486.24	0.69	300.11	31	184
	2	47	3367	7640	3867	24051.00	1.34	300.19	35	183
40_3	1	47	2727	5454	2774	21323.53	0.90	302.06	34	197
	2	47	3609	8832	4463	21957.69	2.20	300.09	36	199
40_4	1	47	2481	4962	2528	24392.64	0.71	300.12	33	197
	2	47	3475	8072	4083	24948.31	1.26	300.68	34	192
40_5	1	47	2495	4990	2542	22832.84	0.70	304.59	31	191
	2	47	3442	7940	4017	23551.16	1.79	300.36	32	182
50_1	1	56	4151	8302	4207	43027.17	0.41	300.24	40	260
	2	56	5187	12288	6200	44318.45	1.40	300.70	45	244

Red de transporte: Grafos generales										
Instancia	$\|M\|$	$\|E\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l \in L'} f_l$
50_2	1	56	3748	7496	3804	43971.89	0.56	300.09	41	241
	2	56	5275	12712	6412	44655.77	1.72	300.30	47	221
50_3	1	56	3721	7442	3777	44000.81	0.64	300.20	40	246
	2	56	5129	11968	6040	45166.16	1.54	300.14	42	242
50_4	1	56	4129	8258	4185	46624.07	0.61	300.20	41	232
	2	56	5674	13364	6738	47607.42	1.38	300.11	45	247
50_5	1	56	3831	7662	3887	50980.39	0.37	300.13	38	240
	2	56	5204	11896	6004	51541.20	1.31	300.12	38	230
60_1	1	68	12950	25900	13018	58390.24	0.66	300.72	50	282
	2	68	17241	39980	20058	59847.52	1.66	300.25	55	320
60_2	1	68	11492	22984	11560	60543.47	0.47	300.18	49	295
	2	68	16556	39128	19632	61541.26	1.61	301.29	49	284
60_3	1	68	12765	25530	12833	67194.86	0.58	300.07	45	256
	2	68	18292	42920	21528	68375.14	1.02	300.08	54	250
60_4	1	68	12776	25552	12844	61699.47	0.68	300.36	52	281
	2	68	15897	33787	17018	62804.49	1.34	305.74	57	272
60_5	1	68	12048	24096	12116	60669.32	0.47	300.11	48	297
	2	68	16434	38640	19388	61454.03	1.28	300.12	51	278
70_1	1	75	5724	11448	5799	78615.90	0.36	300.07	56	359
	2	75	7918	16912	8531	80637.33	1.24	300.19	61	323
70_2	1	75	5812	11624	5887	69889.64	0.31	300.32	57	387
	2	75	7873	16760	8455	71319.77	1.36	300.16	61	367

Red de transporte: Grafos generales										
Instancia	$\|M\|$	$\|E\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l\in L'} f_l$
70_3	1	75	5574	11148	5649	78156.37	0.28	300.13	57	361
	2	75	7688	16276	8213	80851.16	1.29	301.41	61	357
70_4	1	75	5844	11688	5919	73064.84	0.31	300.14	54	363
	2	75	8389	17760	8955	75070.80	1.07	300.23	57	346
70_5	1	75	5190	10380	5265	68121.09	0.48	300.19	54	373
	2	75	7930	16724	8437	68835.62	1.17	300.17	57	351
80_1	1	89	39104	78208	39193	77564.24	0.66	300.34	61	390
	2	89	57476	124784	62481	79317.43	1.94	300.25	66	386
80_2	1	89	39331	78662	39420	73449.97	0.85	300.27	67	391
	2	89	57921	125472	62825	74977.93	1.91	300.50	63	391
80_3	1	89	41783	83566	41872	80228.79	0.67	300.65	62	398
	2	89	59399	127628	63903	82337.29	1.98	300.31	68	395
80_4	1	89	51547	103094	51636	79893.54	0.84	300.34	73	407
	2	89	70845	152140	76159	82382.06	2.54	300.24	71	393
80_5	1	89	44792	89584	44881	81517.37	0.79	300.46	68	384
	2	89	65800	143272	71725	82909.71	2.04	300.63	68	393
90_1	1	97	20862	41724	20959	85014.86	0.45	300.15	67	433
	2	97	29834	61288	30741	87057.55	1.48	300.16	73	443
90_2	1	97	20386	40772	20483	89747.00	0.48	300.87	69	425
	2	97	29118	61952	31073	91889.91	1.43	300.43	79	465
90_3	1	97	18111	36222	18208	90026.99	0.60	301.81	76	447
	2	97	26271	55204	27699	92193.23	1.62	300.34	81	450

Red de transporte: Grafos generales										
Instancia	$\|M\|$	$\|E\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l\in L'} f_l$
90_4	1	97	13660	27320	13757	75408.92	0.49	300.76	71	472
	2	97	18363	37764	18979	77326.07	1.65	300.23	78	465
90_5	1	97	19105	38210	19202	92351.04	0.71	300.20	77	464
	2	97	25211	53020	26607	95321.82	1.45	300.32	84	450
100_1	1	105	13572	27144	13677	95166.92	0.36	300.64	81	585
	2	105	18488	37868	19039	99070.44	1.03	300.79	83	563
100_2	1	105	9753	19506	9858	95045.75	0.44	300.20	76	538
	2	105	13584	27700	13955	98305.30	1.09	300.25	84	507
100_3	1	105	11378	22756	11483	95028.52	0.55	301.84	77	523
	2	105	16419	33800	17005	97656.70	1.11	300.31	83	540
100_4	1	105	14034	28068	14139	103604.88	0.42	300.90	78	502
	2	105	19874	41540	20875	106683.11	1.12	300.22	91	481
100_5	1	105	14802	29604	14907	96825.09	0.42	300.60	77	516
	2	105	21604	44204	22207	99216.71	1.07	300.41	89	560
100_7	1	105	12604	25208	12709	94245.77	0.44	300.16	78	500
	2	105	17672	36128	18169	96668.09	1.07	300.29	89	507
100_8	1	105	14505	29010	14610	92709.29	0.55	300.25	84	536
	2	105	19573	40732	20471	95437.93	1.29	300.28	88	527
100_9	1	105	12979	25958	13084	94845.30	0.38	300.33	80	542
	2	105	18643	38156	19183	99100.15	0.99	300.27	86	538
100_10	1	105	11532	23064	11637	96494.15	0.43	300.96	78	520
	2	105	16419	33916	17063	98967.31	0.86	303.88	81	552

Red de transporte: Grafos generales usando el método heurístico									
Instancia	$\|M\|$	$\|E\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**Tiempo (s)**	$\|L'\|$	$\sum_{l \in L'} f_l$
30_1	1	38	2367	4734	2405	22957.19	7.61	42	146
	2	38	3330	8512	4294	23232.54	11.51	48	145
30_2	1	38	2276	4552	2314	21050.96	6.36	38	147
	2	38	3160	7832	3954	21490.81	6.83	36	151
30_3	1	38	2918	5836	2956	21863.35	4.82	30	138
	2	38	4030	10096	5086	22278.23	7.49	33	135
30_4	1	38	3145	6290	3183	22328.21	5.10	33	141
	2	38	4048	10612	5344	23350.92	8.89	38	137
30_5	1	38	2488	4976	2526	22844.15	7.54	41	147
	2	38	3313	8444	4260	23090.89	7.50	37	150
40_1	1	47	2310	4620	2357	20632.95	8.39	47	221
	2	47	2919	6824	3459	21489.70	12.80	55	219
40_2	1	47	2591	5182	2638	24405.73	3.75	59	195
	2	47	3367	7640	3867	24867.16	4.30	51	186
40_3	1	47	2727	5454	2774	21800.59	8.63	48	202
	2	47	3609	8832	4463	22535.69	14.75	55	202
40_4	1	47	2481	4962	2528	24994.96	6.85	44	203
	2	47	3475	8072	4083	25465.01	8.56	43	197
40_5	1	47	2495	4990	2542	23468.84	7.93	50	195
	2	47	3442	7940	4017	23968.94	8.21	43	185
50_1	1	56	4151	8302	4207	44267.31	21.10	70	249
	2	56	5187	12288	6200	45511.11	19.12	63	245

Red de transporte: Grafos generales usando el método heurístico									
Instancia	$\|M\|$	$\|E\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**Tiempo (s)**	$\|L'\|$	$\sum_{l \in L'} f_l$
50_2	1	56	3748	7496	3804	44972.52	13.09	65	234
	2	56	5275	12712	6412	45858.78	25.77	71	234
50_3	1	56	3721	7442	3777	44628.24	11.71	56	239
	2	56	5129	11968	6040	46213.19	19.53	65	246
50_4	1	56	4129	8258	4185	47815.72	15.92	65	239
	2	56	5674	13364	6738	48620.84	25.69	68	247
50_5	1	56	3831	7662	3887	52178.91	22.31	74	231
	2	56	5204	11896	6004	52879.92	29.54	67	239
60_1	1	68	12950	25900	13018	59849.27	102.94	91	292
	2	68	17241	39980	20058	61203.23	113.39	87	329
60_2	1	68	11492	22984	11560	62262.27	84.30	91	299
	2	68	16556	39128	19632	63253.48	147.22	87	303
60_3	1	68	12765	25530	12833	68944.93	57.40	76	265
	2	68	18292	42920	21528	70207.74	122.78	87	265
60_4	1	68	12776	25552	12844	63099.69	68.23	72	284
	2	68	15897	33787	17018	64310.94	106.45	81	298
60_5	1	68	12048	24096	12116	62084.85	47.25	99	302
	2	68	16434	38640	19388	63564.72	81.87	103	303
70_1	1	75	5724	11448	5799	80869.01	50.72	104	354
	2	75	7918	16912	8531	82680.83	82.74	102	344
70_2	1	75	5812	11624	5887	71505.68	48.36	96	402
	2	75	7873	16760	8455	73364.75	74.20	97	388

Red de transporte: Grafos generales usando el método heurístico									
Instancia	$\|M\|$	$\|E\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**Tiempo (s)**	$\|L'\|$	$\sum_{l \in L'} f_l$
70_3	1	75	5574	11148	5649	79813.22	48.19	98	347
	2	75	7688	16276	8213	82666.81	57.44	97	367
70_4	1	75	5844	11688	5919	74678.86	59.42	98	370
	2	75	8389	17760	8955	77174.83	93.83	99	365
70_5	1	75	5190	10380	5265	70065.99	44.68	97	382
	2	75	7930	16724	8437	70904.40	63.42	93	361
80_1	1	89	39104	78208	39193	79513.76	181.94	117	393
	2	89	57476	124784	62481	81568.36	230.96	111	412
80_2	1	89	39331	78662	39420	74781.43	164.24	108	398
	2	89	57921	125472	62825	76579.56	249.81	102	416
80_3	1	89	41783	83566	41872	82310.01	176.75	110	417
	2	89	59399	127628	63903	83827.55	263.81	106	406
80_4	1	89	51547	103094	51636	82044.41	170.99	122	424
	2	89	70845	152140	76159	83949.28	255.67	108	407
80_5	1	89	44792	89584	44881	83211.77	162.44	113	389
	2	89	65800	143272	71725	84941.72	262.77	117	407
90_1	1	97	20862	41724	20959	87056.44	138.42	116	446
	2	97	29834	61288	30741	89171.97	158.55	122	469
90_2	1	97	20386	40772	20483	91520.35	104.96	112	446
	2	97	29118	61952	31073	93020.43	120.43	128	454
90_3	1	97	18111	36222	18208	91950.95	109.07	126	465
	2	97	26271	55204	27699	94630.43	116.76	132	469

Red de transporte: Grafos generales usando el método heurístico									
Instancia	$\|M\|$	$\|E\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**Tiempo (s)**	$\|L'\|$	$\sum_{l \in L'} f_l$
90_4	1	97	13660	27320	13757	77175.92	90.43	118	489
	2	97	18363	37764	18979	78742.75	101.75	112	462
90_5	1	97	19105	38210	19202	94383.80	110.11	135	481
	2	97	25211	53020	26607	97834.54	123.88	123	457
100_1	1	105	13572	27144	13677	97093.40	67.03	126	548
	2	105	18488	37868	19039	101420.41	90.67	125	541
100_2	1	105	9753	19506	9858	98025.34	68.85	134	553
	2	105	13584	27700	13955	100761.90	80.64	126	543
100_3	1	105	11378	22756	11483	97137.81	64.64	131	553
	2	105	16419	33800	17005	100036.96	81.46	118	537
100_4	1	105	14034	28068	14139	106091.75	93.22	138	538
	2	105	19874	41540	20875	109409.76	174.25	143	509
100_5	1	105	14802	29604	14907	99274.31	72.12	149	541
	2	105	21604	44204	22207	101519.58	130.67	142	588
100_7	1	105	12604	25208	12709	96802.77	74.79	140	527
	2	105	17672	36128	18169	98932.63	142.94	141	535
100_8	1	105	14505	29010	14610	94958.99	93.87	126	556
	2	105	19573	40732	20471	98010.41	138.41	134	558
100_9	1	105	12979	25958	13084	97136.24	74.76	140	563
	2	105	18643	38156	19183	101248.59	143.51	131	566
100_10	1	105	11532	23064	11637	98996.57	78.59	133	543
	2	105	16419	33916	17063	101592.58	144.61	134	547

Red de transporte: Grafos generales usando DCM + método heurístico										
Instancia	$\|M\|$	$\|E\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l \in L'} f_l$
30_1	1	38	2367	4734	2405	22089.55	0.65	300.14	24	139
	2	38	3330	8512	4294	22404.14	1.28	300.15	26	137
30_2	1	38	2276	4552	2314	20572.53	0.55	300.15	23	160
	2	38	3160	7832	3954	20750.90	1.49	300.15	26	140
30_3	1	38	2918	5836	2956	21630.70	0.54	300.11	23	139
	2	38	4030	10096	5086	21767.33	2.01	300.13	32	132
30_4	1	38	3145	6290	3183	21940.84	0.80	300.13	26	140
	2	38	4048	10612	5344	22494.30	2.28	300.13	26	134
30_5	1	38	2488	4976	2526	21946.44	0.80	300.13	23	138
	2	38	3313	8444	4260	22382.50	2.08	300.16	27	143
40_1	1	47	2310	4620	2357	20178.23	0.88	300.33	35	222
	2	47	2919	6824	3459	20837.26	2.06	300.14	35	204
40_2	1	47	2591	5182	2638	23454.24	0.41	300.09	30	186
	2	47	3367	7640	3867	24049.35	1.41	300.18	35	183
40_3	1	47	2727	5454	2774	21332.02	0.96	300.13	34	201
	2	47	3609	8832	4463	21958.29	2.15	301.10	38	199
40_4	1	47	2481	4962	2528	24392.64	0.63	300.19	33	196
	2	47	3475	8072	4083	24928.31	1.14	300.25	33	196
40_5	1	47	2495	4990	2542	22830.84	0.66	300.10	30	190
	2	47	3442	7940	4017	23511.07	1.68	300.13	33	181
50_1	1	56	4151	8302	4207	42967.17	0.14	300.15	37	248
	2	56	5187	12288	6200	44397.11	1.55	300.79	43	240

Red de transporte: Grafos generales usando DCM + método heurístico										
Instancia	$\|M\|$	$\|E\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l\in L'} f_l$
50_2	1	56	3748	7496	3804	43944.74	0.58	300.16	40	235
	2	56	5275	12712	6412	44777.54	1.78	300.36	42	220
50_3	1	56	3721	7442	3777	43986.85	0.67	302.43	39	255
	2	56	5129	11968	6040	45129.49	1.43	300.13	43	244
50_4	1	56	4129	8258	4185	46585.70	0.50	300.12	38	232
	2	56	5674	13364	6738	47572.89	1.39	301.40	43	248
50_5	1	56	3831	7662	3887	51072.52	0.84	637.23	40	232
	2	56	5204	11896	6004	51443.14	1.26	300.14	43	227
60_1	1	68	12950	25900	13018	58373.27	0.62	300.63	50	278
	2	68	17241	39980	20058	59855.96	1.83	300.31	50	311
60_2	1	68	11492	22984	11560	60583.47	0.62	300.16	51	283
	2	68	16556	39128	19632	61529.32	1.59	300.45	50	285
60_3	1	68	12765	25530	12833	67234.86	0.64	300.16	47	254
	2	68	18292	42920	21528	68389.73	1.47	300.42	53	241
60_4	1	68	12776	25552	12844	61686.23	0.59	300.08	51	279
	2	68	15897	33787	17018	62804.49	1.25	300.76	56	269
60_5	1	68	12048	24096	12116	60657.19	0.56	300.12	46	278
	2	68	16434	38640	19388	61555.19	1.85	300.31	52	281
70_1	1	75	5724	11448	5799	78559.75	0.24	300.15	56	347
	2	75	7918	16912	8531	80692.79	1.29	300.15	57	321
70_2	1	75	5812	11624	5887	69861.59	0.26	300.11	56	387
	2	75	7873	16760	8455	71392.59	1.38	300.17	61	369

Red de transporte: Grafos generales usando DCM + método heurístico										
Instancia	$\|M\|$	$\|E\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l\in L'} f_l$
70_3	1	75	5574	11148	5649	78156.37	0.30	300.37	57	353
	2	75	7688	16276	8213	80850.33	1.25	300.26	60	354
70_4	1	75	5844	11688	5919	73116.64	0.38	300.07	57	356
	2	75	8389	17760	8955	75074.20	1.10	300.15	56	347
70_5	1	75	5190	10380	5265	68016.15	0.20	300.17	52	369
	2	75	7930	16724	8437	68855.24	1.23	300.21	57	357
80_1	1	89	39104	78208	39193	77538.94	0.63	300.39	63	386
	2	89	57476	124784	62481	79326.5	1.95	301.91	65	385
80_2	1	89	39331	78662	39420	73368.00	0.72	301.79	63	389
	2	89	57921	125472	62825	75084.09	2.05	303.01	69	387
80_3	1	89	41783	83566	41872	80256.79	0.67	300.39	64	398
	2	89	59399	127628	63903	82337.29	1.98	304.19	68	395
80_4	1	89	51547	103094	51636	79856.04	0.80	315.49	69	407
	2	89	70845	152140	76159	82357.07	2.51	300.32	73	392
80_5	1	89	44792	89584	44881	81527.20	0.79	304.75	66	381
	2	89	65800	143272	71725	82978.86	2.12	300.28	69	393
90_1	1	97	20862	41724	20959	85107.92	0.60	301.77	70	432
	2	97	29834	61288	30741	87146.92	1.59	300.36	77	445
90_2	1	97	20386	40772	20483	89695.59	0.40	300.19	68	439
	2	97	29118	61952	31073	91879.76	1.36	300.18	78	463
90_3	1	97	18111	36222	18208	90010.99	0.61	300.19	74	447
	2	97	26271	55204	27699	92121.98	1.57	300.20	77	452

Red de transporte: Grafos generales cusando DCM + método heurístico										
Instancia	$\|M\|$	$\|E\|$	$\|L\|$	**# Vars**	**# Restr.**	**F. Obj.**	**GAP (%)**	**Tiempo (s)**	$\|L'\|$	$\sum_{l\in L'} f_l$
90_4	1	97	13660	27320	13757	75425.41	0.31	300.17	72	470
	2	97	18363	37764	18979	77409.00	1.70	308.46	78	448
90_5	1	97	19105	38210	19202	92269.89	0.60	300.12	73	466
	2	97	25211	53020	26607	95247.13	1.39	304.44	80	452
100_1	1	105	13572	27144	13677	95194.59	0.37	301.24	81	584
	2	105	18488	37868	19039	99070.44	1.03	300.64	83	563
100_2	1	105	9753	19506	9858	94994.29	0.34	300.28	75	533
	2	105	13584	27700	13955	98301.62	0.82	300.34	86	518
100_3	1	105	11378	22756	11483	94978.67	0.37	300.09	78	525
	2	105	16419	33800	17005	97640.32	1.06	301.71	83	539
100_4	1	105	14034	28068	14139	103575.91	0.41	300.85	78	501
	2	105	19874	41540	20875	106625.01	1.04	300.30	85	485
100_5	1	105	14802	29604	14907	96822.99	0.45	300.23	76	514
	2	105	21604	44204	22207	99194.64	1.07	300.49	87	562
100_7	1	105	12604	25208	12709	94227.52	0.43	301.44	80	542
	2	105	17672	36128	18169	96747.84	1.16	300.27	90	500
100_8	1	105	14505	29010	14610	92746.63	0.63	302.52	84	536
	2	105	19573	40732	20471	95426.67	1.27	300.25	85	529
100_9	1	105	12979	25958	13084	94848.33	0.40	300.84	81	558
	2	105	18643	38156	19183	99470.08	2.76	715.22	83	541
100_10	1	105	11532	23064	11637	96496.82	0.43	301.33	79	531
	2	105	16419	33916	17063	99059.42	0.96	300.21	86	553

4.3. Discusión de resultados

Para cada una de las pruebas realizadas, se puede determinar el comportamiento del algoritmo de solución mediante dos indicadores importantes: la brecha de optimalidad y el tiempo de ejecución. De forma general, se reporta lo siguiente:

- La solución óptima para topologías de líneas es encontrada para la gran mayoría de instancias, incluso para sistemas con 100 nodos que incluyen 840 variables y 519 restricciones. El gap promedio para un solo modo de transporte es de $0{,}01\,\%$ con un tiempo promedio menor a los 300 segundos fijados. Para dos modos de transporte, las instancias con 50 nodos en adelante requieren de a lo más 300 segundos y reportan un gap promedio del $0{,}15\,\%$
- En las redes de árboles, el comportamiento es similar a las líneas (dado que estás últimas son un caso particular de los árboles). Usando un modo de transporte, el solver encuentra una solución óptima en todas las instancias en los 300 segundos, con un gap del $0{,}03\,\%$ aproximadamente. Para el caso de dos modos, los sistemas entre 30 y 50 nodos, le toma un poco más de tiempo que las instancias anteriores, pero de igual forma no supera el tiempo máximo fijado. Para instancias grandes ($n > 50$) el algoritmo se detiene en los 300 segundos y se reporta un gap promedio de $0{,}25\,\%$. En esta red, las instancias más grandes ($n = 100$) utilizaron hasta 2144 variables y 1171 restricciones.
- Para redes de grafos dispersos, la inclusión de ciclos complica aún más al algoritmo por la cantidad de líneas factibles, variables y restricciones consideradas. En este caso, el algoritmo alcanza los 300 segundos fijados en todas las pruebas. Para instancias de entre 30 y 60 nodos, los experimentos con un modo de transporte muestran un gap promedio de $0{,}5\,\%$ mientras que para dos modos se reporta un gap del $1{,}6\,\%$. En instancias entre 70 y 100 estaciones, el gap promedio para un modo de transporte alcanza el $0{,}6\,\%$ mientras que para dos modos sube a $1{,}7\,\%$. Es importante mencionar que, las instancias con 80 nodos fueron las que más trabajo le costó al solver

por la gran cantidad de líneas que se generaron, produciendo modelos con 152140 variables y 76159 restricciones. Para estas matrices, el gap promedio con dos modos de transporte sube hasta el $2{,}54\,\%$.

- La heurística implementada obtuvo valores adecuados en corto tiempo para instancias pequeñas. Para un modo de transporte, la heurística reporta soluciones con diferencias de hasta 2500 unidades respecto del costo obtenido en la red de grafos generales, lo que equivale a un incremento aproximado del $2{,}6\,\%$. Esto se logra en un tiempo no superior a 110 segundos y para todas las instancias, a excepción de las de 80 nodos. Cuando se toma en cuenta a dos modos, el valor de costo aumenta en $2{,}55\,\%$ y lo realiza en aproximadamente 175 segundos. En las instancias de 80 nodos, se necesitaron hasta 170 segundos para reportar un incremento del $2{,}7\,\%$ en un solo modo de transporte. Para el caso de dos modos, el costo se incrementa en promedio el $2{,}4\,\%$ con un tiempo de 265 segundos.

- Al considerar los valores obtenidos por el método heurístico e incluirlos como solución inicial en el modelo entero DCM, el costo objetivo disminuye en promedio el $0{,}12\,\%$ respecto de los grafos generales en donde no se incluye esta solución, tanto para uno como para dos modos de transporte y para la mayoría de instancias. Sin embargo, hay sistemas (grandes y pequeños) para los cuales este procedimiento no es bueno debido a que el costo se mantiene igual o superior al costo del modelo entero orginal, tras el mismo tiempo de ejecución. De forma general, incluir esta solución permitió obtener resultados adecuados para sistemas con dos modos de transporte.

En el reporte de tablas se puede comparar cada una de las pruebas y resultados obtenidos. Es lógico pensar que para los grafos generales, el solver demande de más recursos computacionales y tiempo para alcanzar una solución lo suficientemente buena, situación que es causada por la presencia de ciclos en estas redes. En la líneas y árboles, el solver resuelve esas instancias en un tiempo adecuado y con mayor facilidad, logrando respuestas óptimas o con una pequeña brecha de optimalidad.

La heurística aquí propuesta es un algoritmo sencillo que obtiene soluciones buenas y que no son demasiado alejadas de la óptima. El uso

de este método dependerá netamente de si el modelo entero reporta altos valores en la brecha de optimalidad. En el caso de que se llegue a una solución adecuada en poco tiempo, éste método puede ser innecesario.

Los valores del gap, en todas las instancias experimentadas, son un factor clave que hay que resaltar. En ninguna prueba se obtuvo un valor superior al 3%. Así, en los 300 segundos fijados se alcanzaron valores de gap's adecuados.

Capítulo 5

Conclusiones

La Planificación de Líneas y Frecuencias puede tener objetivos diversos encaminados a resolver condiciones específicas según el interés deseado. Propósitos como la minimización de los costos de operación (aquí planteado), la minimización del tiempo de traslado de los pasajeros, la maximización del número de viajes directos, entre otros, son algunos de los ejemplos que muestran diferentes enfoques para este problema. En este trabajo, discutimos un Modelo de Cubrimiento de Demanda que resuelve al Problema de Planificación de Líneas y Frecuencias en términos de garantizar la calidad del servicio al satisfacer una demanda de pasajeros y minimizar los costos totales de operación. Varias condiciones de factibilidad para este modelo fueron expuestas así como un método heurístico de solución.

Esta componente fue desarrollada en total apego al estudio de los artículos base [1] y [2]. Nuestro principal objetivo fue enriquecerlos con los resultados aquí alcanzados y evidenciar que el LPP puede ser modelado usando la programación lineal entera. Se puede evidenciar que la topología de la red afecta directamente en la complejidad y en el tiempo de solución de la instancia. Al resolver sistemas de diferentes tamaños (de 30 a 100 estaciones), se puede corroborar que el modelo DCM puede trabajar con datos reales del Sistema de Transporte de Quito. En las líneas, se alcanza una solución óptima. En el caso de árboles, depende mucho de si en la red se trabaja con uno o más modos de transporte, mientras que

en las estructuras de grafos dispersos, una solución bastante próxima a la óptima es producida. Los resultados con el método heurístico tambien son interesantes ya que reporta soluciones con un incremento no mayor al $3\,\%$ del costo obtenido por el método exacto (Branch and Bound), en un tiempo significativamente menor.

Con esto, se puede confirmar que el Sistema de Transporte Público de Quito puede ser resuelto como un caso particular de nuestras instancias. En efecto, evidenciamos que el DCM actúa adecuadamente en las topologías de red mencionadas, por lo que al tratar con datos reales de este sistema, se espera que su ejecución también sea oportuna. Para el caso de los corredores troncales (Ecovía, Trolebús, Corredor Central Norte) una topológia de líneas puede aplicarse. De igual manera, para el caso de rutas alimentadoras desde una terminal, se puede adaptar una estructura de árbol en la que la resolución del DCM es pertinente. Cuando se combinan estas rutas o se incluyen estaciones que combinan a varios corredores, la topología de grafos generales es conveniente, con buenos resultados.

Referencias bibliográficas

[1] Torres, L., Torres, R., Borndörfer, R., Pfetschd, M., 2011. Line planning on tree networks with applications to the Quito Trolebús system. International Transactions in Operational Research. IFORS. Escuela politécnica Nacional. Quito, Ecuador; Zuse Institute Berlín 14195, Berlín, Alemania.

[2] Torres, L., Torres, R., Borndörfer, R., Pfetschd, M., 2008. Line Planning on Paths and Tree Networks with Applications to the Quito Trolebús System. 8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems. ATMOS. Escuela politécnica Nacional. Quito, Ecuador; Zuse Institute Berlín 14195, Berlín, Alemania.

[3] Bussieck, M.R., Winter, T., Zimmermann, U.T., 1997b. Discrete optimization in public rail transport. Programación Matemática. 79:415–444

[4] Bussieck, M.R., Kreuzer, P., Zimmermann, U.T., 1997a. Optimal lines for railway systems. Revista europea de Investigación Operativa. 96, 54–63.

[5] Claessens M., van Dijk N.M., Zwaneveld P.J., 1998. Cost optimal allocation of rail passenger lines. Revista europea de Investigación Operativa. 110, 474–489.

[6] Goossens, J.-W.H.M., van Hoesel, S., Kroon, L.G., 2004. A Branch-and-Cut Approach for Solving Railway Line-Planning Problems. Ciencia del Transporte. Vol. 38, No. 3, pp. 379–393.

[7] Borndörfer, R., Grötschel, M., Pfetsch, M.E., 2007. Un enfoque de generación de columnas para la planificación de líneas en público trans-

porte. Ciencias del Transporte 41, 123–132.

[8] Odoni, A.R., Rousseau, J.-M., Wilson, N.H.M., 1994. Models in urban and air transportation. In Pollock, S.M., Rothkopf, M.H., Barnett, A. (eds) Handbooks in OR & MS 6. Capítulo 5. Holanda del Norte, Ámsterdam, pp. 107–150.

[9] Borndörfer, R., Liebchen, C., Pfetsch, M., 2066. Integer Optimization on Public Transportation. Universidad Técnica de Berlín, Facultad II, Departamento de Matemáticas WS

[10] Pfetsch, M., 2007. Line Planning. Instituto Zuse de Berlín.

Capítulo A

Código Python

```
#CODIGO DEL MODELO DCM PARA DOS MODOS DE TRANSPORTE.
#SE CONSIDERA EL CASO DE GRAFOS GENERALES, GRAFOS GENERALES
#CON METODO HEURISTICO Y GRAFOS GENERALES CUANDO SE USA EL DCM
#INCLUYENDO LA SOLUCION HEURISTICA COMO SOLUCION INICIAL

#MODULOS A USAR
import random as rm
from gurobipy import *
import networkx as nx
import ipycytoscape
import numpy as np
import time

#MODOS DE TRANSPORTE

#Determinar el numero de modos de transporte
M=2
print('El numero de modos de transporte a considerar es: {}'.format(M))

#indexamos el conjunto de modos de transporte
index_modos=[i for i in range (1,M+1)]
print('\nEl conjunto de modos de transporte es: {}'.format(index_modos)
    )

#MATRIZ ORIGEN DESTINO (ARCHIVO TXT EXTERNO)

#Lectura de la matriz para determinar el numero de estaciones
```

```
datos= open('Matriz_OD\M_aleatoria_30_1.txt','r') #ejemplo para la instancia "30_1"
# Numero de nodos del grafo
n=int(datos.readline())
print('\nNumero de nodos o estaciones de la red: {}'.format(n))
# Nodos del grafo
V = tuplelist(range(0,n))
print('El conjunto de nodos es: {}'.format(V))

#CREACION DE LA RED (GRAFO GENERAL CON CICLOS)
#Se crea primero un arbol aleatorio y se incluye una cantidad finita de aristas

#arbol aleatorio
nod1=[i for i in range(0,n)]
nod2=[i for i in range(0,n)]
rm.seed(19645174) #Se fija la semilla para pruebas computacionales
l=rm.choice(nod1)
nod2.remove(l)
ll=rm.choice(nod2)
E=tuplelist([(l,ll)])
usados=[E[0][0],E[0][1]]
nousados=[k for k in V if k not in usados]
nod2.append(l)
while len(E) < n-1:
    a=rm.choice(nod1)
    b=rm.choice(nod2)
    if a!=b:
        for j in E:
            if a==j[0] and b in nousados:
                E.append((a,b))
                usados.append(a)
                usados.append(b)
                nousados=[k for k in V if k not in usados]
                break
            elif b==j[0] and a in nousados:
                E.append((a,b))
                usados.append(a)
                usados.append(b)
                nousados=[k for k in V if k not in usados]
                break
            elif a==j[1] and a in nousados:
                E.append((a,b))
```

```
                usados.append(a)
                usados.append(b)
                nousados=[k for k in V if k not in usados]
                break
            elif b==j[1] and b in nousados:
                E.append((a,b))
                usados.append(a)
                usados.append(b)
                nousados=[k for k in V if k not in usados]
                break
            elif a in usados and b in usados:
                break
#Otro arbol aleatorio para elegir una cantidad de aristas adicionales
prob=0.8
edges1 = [(i,j) for i in range(0,n) for j in range(0,n) if rm.random()
    < prob and i!=j]
edges2=[]
for i in range(0,len(edges1)):
    a=rm.choice(edges1)
    edges2.append(a)
    edges1.remove(a)
E1=tuplelist([])
tt=rm.choice(edges2)
E1.append(tt)
edges2.remove(tt)
usados1=[E1[0][0],E1[0][1]]
nousados1=[k for k in V if k not in usados1]
aleatorio=rm.randint(6,11) #numero de aristas adicionales que se
    incluyen al primer arbol
for (i,j) in edges2:
    for p in E1:
        if (i,j)!=p:
            E1.append((i,j))
            usados1.append(i)
            usados1.append(j)
            nousados1=[k for k in V if k not in usados1]
            break
        elif p[0]==i and j in nousados1:
            E1.append((i,j))
            usados1.append(i)
            usados1.append(j)
            nousados1=[k for k in V if k not in usados1]
            break
```

```
        elif p[0]==j and i in nousados1:
            E1.append((i,j))
            usados1.append(i)
            usados1.append(j)
            nousados1=[k for k in V if k not in usados1]
            break
        elif p[1]==i and j in nousados1:
            E1.append((i,j))
            usados1.append(i)
            usados1.append(j)
            nousados1=[k for k in V if k not in usados1]
            break
        elif p[1]==j and i in nousados1:
            E1.append((i,j))
            usados1.append(i)
            usados1.append(j)
            nousados1=[k for k in V if k not in usados1]
            break
        elif p[0]!=i and i in usados1:
            continue
        elif p[1]!=i and i in usados1:
            continue
        elif p[0]!=j and j in usados1:
            continue
        elif p[1]!=j and j in usados1:
            continue
    if len(E1)==aleatorio:
        break

#Conjunto de aristas obtenido
E=tuplelist(list(set(E+E1)))
print('El conjunto de aristas es: {}'.format(E))
print('El numero de aristas del grafo es: {}'.format(len(E)))

#LECTURA DE DATOS DE DEMANDA DE PASAJEROS (MTARIZ OD)

#El origen de estos datos es el mismo archivo de la matriz OD.
#Aqui se describen las estaciones que son terminales, estaciones de
   giro y estaciones de giro.
#Para el caso de dos modos de transporte, se toman las variaciones
   respectivas a fin de
#tener factibilidad en el DCM.

```

```
150 #Guardamos cada una de las lineas del archivo en una lista.
151 dl= datos.readlines()
152 datos.close()
153 Matriz_aux=[]
154 for i in dl:
155     lectura=i.split()[3:]
156     for j in range(0,len(lectura)):
157         lectura[j]=int(lectura[j])
158     Matriz_aux.append(lectura)
159 
160 Matriz_datos=np.matrix(Matriz_aux)
161 print('\nLa matriz origen destino es: \n {}'.format(Matriz_datos))
162 
163 #TERMINALES, ESTACIONES DE GIRO Y NO GIRO PARA CADA MODO DE TRANSPORTE
164 
165 #Funcion auxiliar que busca los vecinos de un nodo
166 def incidencia2(nodo):
167     lista_incidencia=[]
168     for i in range(0,n):
169         if(i,nodo) in E or (nodo,i) in E:
170             lista_incidencia.append(i)
171     return lista_incidencia
172 
173 Terminales_mod={i:[] for i in index_modos}
174 Est_giro_mod={i:[] for i in index_modos}
175 Est_no_giro_mod={i:[] for i in index_modos}
176 for k in index_modos:
177     if k==1:  #el tipo de estacion para el modo uno viene dado en la
    matriz OD
178         for i in range(0,len(dl)):
179             if dl[i].split()[1]=='2':
180                 Terminales_mod[1].append(i)
181             elif dl[i].split()[1]=='1':
182                 Est_giro_mod[1].append(i)
183             else:
184                 Est_no_giro_mod[1].append(i)
185     if k==2:   #para el modo dos se toma como una variacion del modo 1
186         Terminales_mod[2]=Terminales_mod[1]
187         est_giro_aux_nodos_hojas=[i for i in V if len(incidencia2(i))
    ==1 and i not in Terminales_mod[2]]
188         Est_giro_mod[2]=est_giro_aux_nodos_hojas
189         Est_no_giro_mod[2]=[i for i in V if i not in Terminales_mod[2]
    and i not in Est_giro_mod[2]]
```

```
190
191     print('\nEl conjunto de nodo terminales del modo {} es: {}'.format(
        k, Terminales_mod[k]))
192     print('El conjunto de estaciones de giro del modo {} es: {}'.format
        (k, Est_giro_mod[k]))
193     print('El conjunto de estaciones de no giro del modo {} es: {}'.
        format(k,Est_no_giro_mod[k]))
194
195 #SIMULAMOS LA RED DE TRANSPORTE
196
197 #Distancias entre estaciones que nos servira para encontrar el camino
        mas corto
198
199 #Tomamos puntos aleatorios en el plano cartesiano (estaciones) y
        tomamos la distancia euclidea entre ellos
200 coordx={i : rm.randint(0,10) for i in V}
201 coordy={i : rm.randint(0,10) for i in V}
202 distancias = tupledict({(i,j) : math.sqrt((coordx[i] - coordx[j])**2 +
        (coordy[i] - coordy[j])**2)
203     for (i,j) in E})
204 lista_arista_costos=[(i,j,distancias[(i,j)]) for (i,j) in E]
205 print("La distancia entre cada estacion es: {}".format(distancias))
206
207 #Creacion del grafo
208 D = nx.Graph()
209 D.add_nodes_from(V)
210 #t :  estacion terminal
211 #g : estacion de giro
212 #ng: estacion de no giro
213
214 #Se agrega una etiqueta para identificar el tipo de estacion
215 for i in V:
216     if i in Terminales_mod[1] and i in Terminales_mod[2]:
217         D.nodes[i]['etiq']= str(i) + '\n' + '(t,t)'
218     elif i in Est_giro_mod[1] and i in Est_giro_mod[2] :
219         D.nodes[i]['etiq']= str(i) + '\n' + '(g,g)'
220     elif i in Est_no_giro_mod[1] and i in Est_no_giro_mod[2] :
221         D.nodes[i]['etiq']= str(i) + '\n' + '(ng,ng)'
222     elif i in Est_giro_mod[1] and i in Est_no_giro_mod[2] :
223         D.nodes[i]['etiq']= str(i) + '\n' + '(g,ng)'
224     elif i in Est_no_giro_mod[1] and i in Est_giro_mod[2] :
225         D.nodes[i]['etiq']= str(i) + '\n' + '(ng,g)'
226     elif i in Est_giro_mod[1] and i in Teminales_mod[2] :
```

```
        D.nodes[i]['etiq']= str(i) + '\n' + '(g,t)'
    elif i in Est_no_giro_mod[1] and i in Terminales_mod[2]:
        D.nodes[i]['etiq']= str(i) + '\n' + '(ng,t)'
    elif i in Terminales[1] and i in Est_giro_mod[2] :
        D.nodes[i]['etiq']= str(i) + '\n' + '(t,g)'
    elif i in Terminales[1] and i in Est_no_giro_mod[2]:
        D.nodes[i]['etiq']= str(i) + '\n' + '(t,ng)'

D.add_weighted_edges_from(lista_arista_costos,weight='weight') # agregamos el costo (distancia) de cada arista


#Demanda agregada de cada arista del grafo

#Elementos para guardar la demanda de las aristas en ambos sentidos
ar_sentido1=[(i,j) for (i,j) in E]
ar_sentido2=[(j,i) for (i,j) in E]
ar_sentido12=ar_sentido1+ar_sentido2
x_sentido12={(i,j) : 0 for (i,j) in ar_sentido12}

#calculamos la demanda agregada de cada arista segun el algoritmo del System Split con las funcion shortest_path()
for i in range (0,n):
    for j in range (0,n):
        if i!=j:
            t=nx.shortest_path(D,source=i,target=j,weight='weight')
            for k in range(0,len(t)-1):
                if (t[k],t[k+1]) in ar_sentido12:
                    x_sentido12[(t[k],t[k+1])]=x_sentido12[(t[k],t[k+1])]+Matriz_datos[i,j]

#Demanda agregada de cada arista
demanda={(i,j):max(x_sentido12[(i,j)],x_sentido12[(j,i)]) for (i,j) in E}
print("La demanda agregada de pasajeros en cada arista es: {}".format(demanda))

#Dibujamos el grafo
for i,j in E:
    D.edges[i,j]['etiq'] = str(demanda[(i,j)]) #etiquetamos cada arista con la demanda agregada calculada
    D.edges[i,j]['color'] =  '#9dbaea'
grafo = ipycytoscape.CytoscapeWidget()
```

```
grafo.graph.add_graph_from_networkx(D, directed=False)
grafo.set_style([{'selector': 'node', 'style' : {'background-color': '
    data(color)', 'font-family': 'helvetica', 'font-size': '12px', '
    color':'black', 'label': 'data(etiq)', 'text-wrap' : 'wrap', 'text-
    valign' : 'center'}},
                    {'selector': 'node:parent', 'css': {'background-
    opacity': 0.333}, 'style' : {'font-family': 'helvetica', 'font-size
    ': '20px', 'label': 'data(etiq)'}},
                    {'selector': 'edge', 'style': {'width': 5, 'line-
    color': 'data(color)', 'font-size': '10px', 'label': 'data(etiq)',
    'text-valign' : 'top', 'text-margin-y' : '-10px'}},
                    {'selector': 'edge.directed', 'style': {'curve-
    style': 'bezier', 'target-arrow-shape': 'triangle', 'target-arrow-
    color': '#FFB233'}}])

print('\nLa simulacion del grafo es:')
grafo


#CONJUNTO DE LINEAS POSIBLES

#Lineas auxiliares para determinar el conjunto de lineas entre cada par
    de estaciones
lineas_aux1=list()
for i in range(0,n):
    for j in range(i+1,n):
        c=list(nx.all_simple_paths(D,source=i,target=j))
        for k in c:
            lineas_aux1.append(tuple(k))

lineas_aux2=list()
for i in range(n-1,-1,-1):
    for j in range(i-1,-1,-1):
        c=list(nx.all_simple_paths(D,source=i,target=j))
        for k in c:
            lineas_aux2.append(tuple(k))
lineas_aux=lineas_aux1+lineas_aux2

#Del conjunto de lineas auxiliar creado, se toman las lineas para cada
    modo de trasporte.
#Se sabe que una linea debe partir de un terminal y avanzar hasta otro
    terminal o una estacion de giro para regresar a su terminal de
    partida.
```

```
294
295 lineas_mod={i:[] for i in index_modos}
296 for k in index_modos:
297     lineas_mod[k]=list()
298     for i in lineas_aux:
299         if i[0] in Terminales_mod[k]:
300             if i[-1] in Terminales_mod[k] or i[-1] in Est_giro_mod[k]:
301                 lineas_mod[k].append(i)
302     print('\nEl conjunto de lineas del modo {} es:\n{}'.format(k,
    lineas_mod[k]))
303     print('\nLa cantidad de lineas factibles del modo {} es: {}'.format
    (k,len(lineas_mod[k])))
304
305 #ASIGNAMOS UN ID A CADA UNA DE LAS LINEAS DE CADA MODO DE TRANSPORTE
306
307 #indexamos el conjunto de lineas del modo m
308 index_aux_mod={i:{} for i in index_modos}
309 index_lineas_mod={i:[] for i in index_modos}
310
311 for k in index_modos:
312     index_aux_mod[k]=tupledict({lineas_mod[k][i]:i+1 for i in range(0,
    len(lineas_mod[k]))})
313     print('\nAsignamos un ID (indice) a cada una de las lineas del modo
     {} para reconocerla en el modelo:\n{}'.format(k,index_aux_mod[k]))
314     index_lineas_mod[k]=tuplelist([index_aux_mod[k][i] for i in
    lineas_mod[k]]) #sacamos la lista de los id's de las lineas
315
316 #Costos de cada linea.
317 #Estos vienen dados por las distancia recorrida entre el nodo terminal
    y su estacion de giro. Se lo multiplica por el costo de
318 #kilometro y por dos
319
320 #costo kilometro
321 Costo_kilometro_mod=[3,4] #3=costo del kilometro para el modo 1; 4=
    costo del kilometro para el modo 2
322 costos_mod={i:[] for i in index_modos}
323 lcc_mod={i:[]for i in index_modos}
324
325 #Calculamos la distancia recorrida de cada linea
326 for k in index_modos:
327     for i in range(0,len(lineas_mod[k])):
328         cc=0
329         for p in range(0,len(lineas_mod[k][i])-1):
```

```
330             if (lineas_mod[k][i][p],lineas_mod[k][i][p+1]) in E:
331                 cc=cc+distancias[(lineas_mod[k][i][p],lineas_mod[k][i][
    p+1])]
332             elif (lineas_mod[k][i][p+1],lineas_mod[k][i][p]) in E:
333                 cc=cc+distancias[(lineas_mod[k][i][p+1],lineas_mod[k][i
    ][p])]
334         lcc_mod[k].append(cc)
335
336 #Calculamos los costos variables de cada linea
337 for k in index_modos:
338     costos_mod[k]={lineas_mod[k][i]:2*Costo_kilometro_mod[k-1]*lcc_mod[
    k][i] for i in range(0,len(lineas_mod[k]))}
339     print('\nLos costos variables de cada linea del modo {} son:\n{}'.
    format(k,costos_mod[k]))
340
341 #UNIMOS EN UN DICCIONARIO LOS ID DE CADA LINEA CON SU COSTO PARA CADA
    UNO DE LOS MODOS DE TRANSPORTE'
342 costos_lineas_lista_mod={i:[] for i in index_modos}
343 lineas_y_costos_mod={i:{} for i in index_modos}
344
345 for k in index_modos:
346     #indexamos el conjunto de los costos de las lineas
347     costos_lineas_lista_mod[k]=tuplelist([costos_mod[k][i] for i in
    lineas_mod[k]])
348     #juntamos el numero de linea de cada modo con su costo en un
    diccionario para que pueda ser leido en la funcion objetivo
349     lineas_y_costos_mod[k]={index_lineas_mod[k][i]:
    costos_lineas_lista_mod[k][i] for i in range(0,len(lineas_mod[k]))}
350     print('\nJuntamos los id de las lineas del modo {} con su costo
    variable:\n{}'.format(k,lineas_y_costos_mod[k]))
351
352 #Asignamos valores de frecuencias (maximas, minimas) y costos fijos a
    cada linea del modo m
353
354 frec_max_mod=[30,30] #30=frecuencia maxima para lineas del modo uno y
    dos
355 costos_fijos_mod=[20,22] #30=costos fijos para lineas del modo uno y
    dos; 22=costos fijos para lineas del modo dos
356
357 frec_max_lineas_mod={i:{} for i in index_modos}
358 frec_max_lineas_lista_mod={i:[] for i in index_modos}
359 frec_min_lineas_mod={i:{} for i in index_modos}
360 lineas_costos_fijos_mod={i:{} for i in index_modos}
```

```
lineas_costos_fijos_lista_mod={i:[] for i in index_modos}
for k in index_modos:
    #frecuencias maximas para lineas del modo m
    frec_max_lineas_mod[k]=tupledict({index_lineas_mod[k][i]:
    frec_max_mod[k-1] for i in range(0,len(lineas_mod[k]))})
    print('\nLas frecuencias maximas de cada linea del modo {} son:\n{}
    '.format(k,frec_max_lineas_mod[k]))
    #lo guardamos en una lista para que pueda ser leido en la funcion
    objetivo
    frec_max_lineas_lista_mod[k]=tuplelist([frec_max_lineas_mod[k][i]
    for i in range(1,len(lineas_mod[k])+1)])
    #frecuencias minimas para lineas del modo m. Por hipotesis son cero
    frec_min_lineas_mod[k]=tupledict({index_lineas_mod[k][i]:0 for i in
     range(0,len(lineas_mod[k]))})
    print('\nLas frecuencias minimas de cada linea del modo {} son:\n{}
    '.format(k,frec_min_lineas_mod[k]))
    #Costos fijos de las lineas
    lineas_costos_fijos_mod[k]={index_lineas_mod[k][i]:costos_fijos_mod
    [k-1] for i in range(0,len(lineas_mod[k]))}
    print('\nConsideramos los siguientes costos fijos para cada linea
    del modo {}:\n{}'.format(k,lineas_costos_fijos_mod[k]))
    #los guardamos en una lista a estos costos fijos
    lineas_costos_fijos_lista_mod[k]=tuplelist([lineas_costos_fijos_mod
    [k][i] for i in range(1,len(lineas_mod[k])+1)])

#IMPLEMENTACION DEL DCM

#Crear el objeto modelo
m = Model('LPP_Quito')
#Elementos auxiliares para ser entendidos en el modelo:
#tomamos el conjunto de lineas del modo de transporte que tiene una
    mayor cantidad de lineas factibles. Para ellos, definimos la
    funcion longitud():
#Esto se hace para facilitar la implementacion el caso cuando el numero
     de lineas de cada modo de transporte es diferente.

def longitud():
    longitud_lineas_mod=[]
    for k in index_modos:
        longitud_lineas_mod.append(len(index_lineas_mod[k]))

    for k in index_modos:
        if len(index_lineas_mod[k])==max(longitud_lineas_mod):
```

```
            break
    return k

a=longitud()
print('\nEl modo con mayor cantidad de  lineas es: {}'.format(a))
index_lineas_tot=index_lineas_mod[a]

#Diccionario auxiliar con la frecuencia minima para cada una de las lineas l del modo de transporte m.
#Este elemento es necesario para utilizarlo directamente en la defincion de las variables de decision
dic_frec_min_lineas_mod_tot={(i,j):0 for i in index_lineas_tot for j in index_modos}


#Creamos las variables
#VARIABLES f_lm enteras ya acotadas inferiormente por el diccionario creado.
#En esta implementacion, nos resulta mas facil indexar a las variables por l y m, siendo
#f_lm=frecuencia de la linea l del modo m
f = m.addVars(index_lineas_tot,index_modos,vtype = GRB.INTEGER, lb=dic_frec_min_lineas_mod_tot, name="f")
print('\nVariables enteras y positivas f_lm:\n{}'.format(f))
#VARIABLES y_lm binaria (y_lm=1 si se usa la linea l del modo m, y_lm=0 si no)
#Se indexa a estas variables de la misma manera que las f_lm
y = m.addVars(index_lineas_tot,index_modos,vtype = GRB.BINARY, name="y")
print('\nVariables binarias y_lm:\n{}'.format(y))

#Funcion objetivo a minimizar
#Tomamos otros elementos auxiliares para definir en un solo paso a la funcion objetivo
costos_lineas_tot={(i,j):0 for i in index_lineas_tot for j in index_modos}
costos_lineas_fijos_tot={(i,j):0 for i in index_lineas_tot for j in index_modos}
for i in index_lineas_tot:
    for j in index_modos:
        if i in index_lineas_mod[j]:
            costos_lineas_tot[(i,j)]=costos_lineas_tot[(i,j)]+costos_lineas_lista_mod[j][i-1]
```

```
423             costos_lineas_fijos_tot[(i,j)]=costos_lineas_fijos_tot[(i,j
    )]+lineas_costos_fijos_lista_mod[j][i-1]
424         else:
425             costos_lineas_tot[(i,j)]=costos_lineas_tot[(i,j)]+0
426             costos_lineas_fijos_tot[(i,j)]=costos_lineas_fijos_tot[(i,j
    )]+0
427
428 print('\nJuntamos el valor de los costos variables de cada linea de
    cada modo de transporte con la tupla (l,m)'
429       + 'para guardar en un diccionario todos los costos operacionales
    :\n{}'.format(costos_lineas_tot))
430 print('\nJuntamos el valor de los costos fijos de cada linea de cada
    modo de transporte con la tupla (l,m)'
431       + 'para guardar en un diccionario todos los costos fijos:\n{}'.
    format(costos_lineas_fijos_tot))
432
433 #Definimos la funcion objetivo
434 m.setObjective(f.prod(costos_lineas_tot,'*','*')+y.prod(
    costos_lineas_fijos_tot,'*','*'), GRB.MINIMIZE)
435
436 #Restricciones
437
438 #Calculamos el conjunto de lineas del modo m que utilizan la arista e
439
440 lineas_por_modo={i:{} for i in index_modos}
441 index_lineas_por_modo={i:{} for i in index_modos}
442 for p in index_modos:
443     lineas_por_modo[p]={(i,j) : () for (i,j) in E} #Construimos el
    conjunto L_e^m
444     for (i,j) in E:
445         for k in lineas_mod[p]:
446             for t in range(0,len(k)-1):
447                 if (k[t],k[t+1])==(i,j) or (k[t+1],k[t])==(i,j) :
448                     lineas_por_modo[p][(i,j)]=lineas_por_modo[p][(i,j)
    ]+(k,)
449                     break
450                 else:
451                     continue
452     print('\nConstruimos el conjunto (L_e^{}), es decir, el conjunto de
     lineas del modo {} que utilizan cada una de las aristas del grafo
    :\n{}'.format(p,p,lineas_por_modo[p]))
453     #indexamos este conjunto el conjunto construido siguiendo la misma
    estructura
```

```
454     index_lineas_por_modo[p]={(i,j): [] for (i,j) in E}
455     for (i,j) in E:
456         for k in lineas_por_modo[p][(i,j)]:
457             index_lineas_por_modo[p][(i,j)]=index_lineas_por_modo[p][(i
    ,j)]+[index_aux_mod[p][k]]
458     print('\nLe asiganamos el ID de cada una de las lineas del modo {}
    que son utilizadas en cada una de las aristas:\n{}'.format(p,
    index_lineas_por_modo[p]))
459
460 #Restricciones de demanda agregada en cada arista
461
462 #Capacidades de los modos
463 capacidad=[180,210] #180=capacidad de los buses de modo uno; 210=
    capacidad de los buses de modo dos
464
465 #colocamos las restricciones
466 for (i,j) in E:
467     e=0
468     for k in index_modos:
469         e=e+capacidad[k-1]*f.sum((p for p in index_lineas_por_modo[k][(
    i,j)]),k)
470     m.addConstr((e>=demanda[(i,j)]),"demanda_pasajeros[{},{}]".format(i
    ,j))
471
472 #Restricciones de frecuencia maxima
473
474 # Es necesario tambien incluir en un diccionario las frecuencias de
    cada linea de cada modo con su valor.
475 dic_frec_max_lineas_tot={(i,j):0 for i in index_lineas_tot for j in
    index_modos}
476 for i in index_lineas_tot:
477     for j in index_modos:
478         if i in index_lineas_mod[j]:
479             dic_frec_max_lineas_tot[(i,j)]=dic_frec_max_lineas_tot[(i,j
    )]+frec_max_lineas_lista_mod[j][i-1]
480         else:
481             dic_frec_max_lineas_tot[(i,j)]=dic_frec_max_lineas_tot[(i,j
    )]+0
482
483 print('\nJuntamos el valor de las frecuencias maximas de cada linea de
    cada modo de transporte con la tupla (l,m)'
484       + 'para guardar en un diccionario todas estas frecuencias maximas
    :\n{}'.format(dic_frec_max_lineas_tot))
```

```
485
486 #agregamos las restricciones
487 m.addConstrs((f[(i,j)]<=dic_frec_max_lineas_tot[(i,j)]*y[(i,j)] for i
    in index_lineas_tot for j in index_modos), "frec_maxima*y")
488
489 #Condicion adicional
490 #Se necesita ingresar una consideracion adicional para cuando el numero
     de lineas
491 #son diferentes entre los modos de transporte. Esto se hace solo para
    entendimiento en la
492 #implementacion puesto que se indexo a las variables
493 por los indices l y m en su definicion.
494 #Esto no afecta a la formulacion del DCM.
495 #Esta restriccion asegura que para cuando haya modos de transporte con
    diferente numeros de lineas, las lineas "excedentes"
496 #del modo con un menor numero de lineas, no se consideren, y por tanto
    tampoco sus costos fijos.
497 for p in index_modos:
498     if p!=a:
499         m.addConstrs((y[(i,p)]<=f[(i,p)]  for i in range(len(
    index_lineas_mod[p])+1,len(index_lineas_tot)+1)), "y_aux")
500
501 # Escribimos el modelo a un archivo externo para facilitar su revision
502 m.write('LPP.lp')
503
504 #RESOLVEMOS EL DCM
505
506 # Terminar luego de 300 segundos
507 m.Params.TimeLimit = 300
508
509 m.optimize()
510
511 #Mostramos los resultados respecto a las lineas elegidas
512 print('\nLa solucin que se reporta es:')
513 if m.status == GRB.Status.OPTIMAL or m.status==9:
514     # Recuperar los valores de las variables
515     vf=m.getAttr('x', f)
516     vy=m.getAttr('x', y)
517     suma_lineas=0
518     suma_frecuencias=0
519     for l in index_lineas_tot:
520         for p in index_modos:
521             if vf[l,p]>0 and vy[l,p]>=0.99:
```

```
522                 suma_lineas=suma_lineas+1
523                 suma_frecuencias=suma_frecuencias+vf[l,p]
524                 print('La linea {} del modo {} es considerada y su
    frecuencia es: {}'.format(l, p, vf[l,p]))
525
526 #datos resultantes
527 cantidad_lineas=0
528 for k in index_modos:
529     cantidad_lineas=cantidad_lineas+len(lineas_mod[k])
530 print('El numero de estaciones consideradas es: {}'.format(n))
531 print('El numero de aristas consideradas es: {}'.format(len(E)))
532 print('La cantidad de lineas factibles de todos los modos de transporte
     es: {}'.format(cantidad_lineas))
533 print('El valor objetivo obtenido es: {}'.format(m.objval))
534 print('El tiempo de ejecucion es: {}'.format(m.runtime))
535 print('El Gap obtenido es: {}'.format(m.MIPGap))
536 print('El numero de lineas elegidas es: {}'.format(suma_lineas))
537 print('La suma de las frecuencias de las lineas elegidas es: {}'.format
    (suma_frecuencias))
538 print('La cantidad de variables que se utilizaron es: {}'.format(len(vf
    )*2))
539 print('La cantidad de restricciones que se utilizaron es: {}'.format(
    len(E)+len(vf)))
540
541
542 #Implementacion con el metodo heuristico
543
544 #Es necesario definir nuevamente otro modelo con todos sus elementos
    anteriores. Este modelo es la relajacion lineal del modelo entero.
545 rl = Model('LPP_Quito_RL')
546 f_rl = rl.addVars(index_lineas_tot,index_modos, lb=
    dic_frec_min_lineas_mod_tot, name="f_rl")
547 y_rl = rl.addVars(index_lineas_tot,index_modos, name="y_rl") #no se
    especifica el tipo de variable porque por defecto Gurobi las asume
    como continuas
548 rl.setObjective(f_rl.prod(costos_lineas_tot,'*','*')+y_rl.prod(
    costos_lineas_fijos_tot,'*','*'), GRB.MINIMIZE)
549 rl.update()
550
551 for (i,j) in E:
552     e=0
553     for k in index_modos:
554         e=e+capacidad[k-1]*f_rl.sum((p for p in index_lineas_por_modo[k
```

```
    ][(i,j)]),k)
555     rl.addConstr((e>=demanda[(i,j)]),"demanda_pasajeros_rl[{},{}]".
    format(i,j))
556 
557 rl.addConstrs((f_rl[(i,j)]<=dic_frec_max_lineas_tot[(i,j)]*y_rl[(i,j)]
    for i in index_lineas_tot for j in index_modos), "frec_maxima*y_rl"
    )
558 
559 for p in index_modos:
560     if p!=a:
561         rl.addConstrs((y_rl[(i,p)]<=f_rl[(i,p)]  for i in range(len(
    index_lineas_mod[p])+1,len(index_lineas_tot)+1)), "y_aux_rl")
562 
563 #Resolvemos la relajacion lineal
564 rl.optimize()
565 print('\nLa solucion de la Relajacion lineal es:')
566 
567 if rl.status == GRB.Status.OPTIMAL or rl.status==9:
568     # Recuperar los valores de las variables
569     vf_rl=rl.getAttr('x', f_rl)
570     vy_rl=rl.getAttr('x', y_rl)
571     for l in index_lineas_tot:
572         for p in index_modos:
573             if vf_rl[l,p]>0 and vy_rl[l,p]>0:
574                 print('La linea {} del modo {} es considerada y su
    frecuencia es: {}'.format(l, p, vf_rl[l,p]))
575 
576 #Codigo de resolucion iterativa con heuristica
577 
578 #definimos una funcion auxiliar
579 def hay_frac(var1,var2):
580     cont=0
581     for l in index_lineas_tot:
582         for p in index_modos:
583             if math.modf(var1[l,p])[0]==0 and math.modf(var2[l,p])
    [0]==0:
584                 cont=cont+1
585             else:
586                 break
587         if cont==0:
588             break
589         else:
590             continue
```

```
591
592     if cont==len(var1):
593         return False
594     else:
595         return True
596
597 tiempo_inicio_heu=time.time()
598 contar_tiempo_heu=0
599
600 while hay_frac(vf_rl,vy_rl)==True:
601     cont2=0
602     for l in index_lineas_tot:
603         for p in index_modos:
604             if math.modf(vf_rl[l,p])[0]!=0 and math.modf(vy_rl[l,p])
    [0]!=0:
605                 rl.addConstr((f_rl[l,p]==math.ceil(vf_rl[l,p])),"
    aux_iterativa_f[{},{}]".format(l,p))
606                 rl.addConstr((y_rl[l,p]==math.ceil(vy_rl[l,p])),"
    aux_iterativa_y[{},{}]".format(l,p))
607                 cont2=0
608                 break
609             elif  vf_rl[l,p]==0 and vy_rl[l,p]==0:
610                 rl.addConstr((f_rl[l,p]==vf_rl[l,p]),"aux_iterativa_f
    [{},{}]".format(l,p))
611                 rl.addConstr((y_rl[l,p]==vy_rl[l,p]),"aux_iterativa_y
    [{},{}]".format(l,p))
612                 cont2=cont2+1
613             elif math.modf(vf_rl[l,p])[0]==0 and math.modf(vy_rl[l,p])
    [0]!=0:
614                 rl.addConstr((f_rl[l,p]==vf_rl[l,p]),"aux_iterativa_f
    [{},{}]".format(l,p))
615                 rl.addConstr((y_rl[l,p]==math.ceil(vy_rl[l,p])),"
    aux_iterativa_y[{},{}]".format(l,p))
616                 cont2=0
617                 break
618             elif math.modf(vf_rl[l,p])[0]!=0 and math.modf(vy_rl[l,p])
    [0]==0:
619                 rl.addConstr((f_rl[l,p]==math.ceil(vf_rl[l,p])),"
    aux_iterativa_f[{},{}]".format(l,p))
620                 rl.addConstr((y_rl[l,p]==vy_rl[l,p]),"aux_iterativa_y
    [{},{}]".format(l,p))
621                 cont2=0
622                 break
```

```
623         if cont2==0:
624             break
625         else:
626             continue
627     rl.optimize()
628     contar_tiempo_heu=contar_tiempo_heu+rl.runtime
629     if rl.status == GRB.Status.OPTIMAL:
630         vf_rl=rl.getAttr('x', f_rl)
631         vy_rl=rl.getAttr('x', y_rl)
632 
633 tiempo_fin_heu=time.time()
634 
635 print('\nLa solucion que se obtine de la heuristica es:')
636 if rl.status == GRB.Status.OPTIMAL or rl.status==9:
637     # Recuperar los valores de las variables
638     vf_rl=rl.getAttr('x', f_rl)
639     vy_rl=rl.getAttr('x', y_rl)
640     suma_lineas=0
641     suma_frecuencias=0
642     for l in index_lineas_tot:
643         for p in index_modos:
644             if vf_rl[l,p]>0 and vy_rl[l,p]>=0.99:
645                 suma_lineas=suma_lineas+1
646                 suma_frecuencias=suma_frecuencias+vf_rl[l,p]
647                 print('La linea {} del modo {} es considerada y su
    frecuencia es: {}'.format(l, p, vf_rl[l,p]))
648 
649 #datos resultantes
650 print('El valor objetivo obtenido es: {}'.format(rl.objval))
651 print('El tiempo de ejecucion es: {}'.format(contar_tiempo_heu))
652 print('El numero de lineas elegidas es: {}'.format(suma_lineas))
653 print('La suma de las frecuencias de las lineas elegidas es: {}'.format
    (suma_frecuencias))
654 
655 
656 #Modelo entero con solucion heuristica como solucion inicial
657 
658 #Definimos nuevamente otro modelo con todos los elementos anteriores
659 # para que se cargue la solucion de la heuristica en sus variables
660 
661 ss = Model('LPP_Quito')
662 fff = ss.addVars(index_lineas_tot,index_modos,vtype = GRB.INTEGER, lb=
    dic_frec_min_lineas_mod_tot, name="fff")
```

```
yyy = ss.addVars(index_lineas_tot,index_modos,vtype = GRB.BINARY, name="yyy")
ss.setObjective(fff.prod(costos_lineas_tot,'*','*')+yyy.prod(costos_lineas_fijos_tot,'*','*'), GRB.MINIMIZE)
ss.update()

for l in index_lineas_tot:
    for p in index_modos:
        #aqui cargamos la solucion heuristica al modelo
        fff[(l,p)].start=vf_rl[(l,p)]
        yyy[(l,p)].start=vy_rl[(l,p)]

for (i,j) in E:
    e=0
    for k in index_modos:
        e=e+capacidad[k-1]*fff.sum((p for p in index_lineas_por_modo[k][(i,j)]),k)
    ss.addConstr((e>=demanda[(i,j)]),"demanda_pasajeros[{},{}]".format(i,j))

ss.addConstrs((fff[(i,j)]<=dic_frec_max_lineas_tot[(i,j)]*yyy[(i,j)] for i in index_lineas_tot for j in index_modos), "frec_maxima*y")

for p in index_modos:
    if p!=a:
        ss.addConstrs((yyy[(i,p)]<=fff[(i,p)]  for i in range(len(index_lineas_mod[p])+1,len(index_lineas_tot)+1)), "y_aux")


#Resolvemos el modelo

# Terminar luego de 300 segundos
ss.Params.TimeLimit = 300

ss.optimize()

print("La solucion que se reporta con la solucion heuristica como solucion inicial es: ")
if ss.status == GRB.Status.OPTIMAL or ss.status==9:
    # Recuperar los valores de las variables
    vfff=ss.getAttr('x', fff)
    vyyy=ss.getAttr('x', yyy)
    suma_lineas=0
```

```
699     suma_frecuencias=0
700     for l in index_lineas_tot:
701         for p in index_modos:
702             if vfff[l,p]>0 and vyyy[l,p]>=0.99:
703                 suma_lineas=suma_lineas+1
704                 suma_frecuencias=suma_frecuencias+vfff[l,p]
705                 print('La linea {} del modo {} es considerada y su
    frecuencia es: {}'.format(l, p, vfff[l,p]))
706 #datos resultantes
707 print('El valor objetivo obtenido es: {}'.format(ss.objval))
708 print('El tiempo de ejecucion es: {}'.format(ss.MIPGap))
709 print('El numero de lineas elegidas es: {}'.format(suma_lineas))
710 print('La suma de las frecuencias de las lineas elegidas es: {}'.format
    (suma_frecuencias))
```