Trabajo Fin de Máster
Máster en Organización Industrial y Gestión de Empresas

# Diseño y análisis de métodos basados en caminos mínimos para la construcción del pool de líneas candidatas en problemas de diseño de redes de transporte público

Autor: Carolina del Carmen Muñoz Delgado
Tutor: José David Canca Ortiz

**Dpto. Organización Industrial y Gestión de Empresas I**
**Escuela Técnica Superior de Ingeniería**

Sevilla, 2022

Trabajo Fin de Máster
Organización Industrial y Gestión de Empresas

# Diseño y análisis de métodos basados en caminos mínimos para la construcción del pool de líneas candidatas en problemas de diseño de redes de transporte público

Autor:
Carolina del Carmen Muñoz Delgado

Tutor:
José David Canca Ortiz
Catedrático de Universidad

Dpto. Organización Industrial y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Trabajo Fin de Máster: Diseño y análisis de métodos basados en caminos mínimos para la construcción del pool de líneas candidatas en problemas de diseño de redes de transporte público

Autor: Carolina del Carmen Muñoz Delgado

Tutor: José David Canca Ortiz

El tribunal nombrado para juzgar el Trabajo arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

*A mi familia y amigos*

*A mis maestros*

# Agradecimientos

A mi tutor y profesor David Canca, por su total entrega y disponibilidad a lo largo de todo el máster, y por ser un gran docente que deja huella en la formación de sus alumnos.

A mis padres, por darnos mutuamente el apoyo incondicional que nos ayuda a superar los retos que se nos presentan.

A Álvaro, por creer en mí y ayudarme a superarme día a día.

A todos mis amigos, por su paciencia y apoyo, y muy en especial a Sara, por los consejos, motivación y comprensión que he recibido en todo momento.

A María, Ana y Mayte, por formar un gran equipo de estudio durante todo el máster, compartiendo los momentos de agobio y de celebración tras cada logro.

*Carolina del Carmen Muñoz Delgado*

*Sevilla, 2022*

# Resumen

El presente Trabajo Fin de Máster se centra en el estudio del problema de planificación de líneas de transporte público (TNDP) y en su resolución mediante un modelo de optimización que permita obtener el diseño óptimo de la red de transporte, minimizando el tiempo total de viaje en la red priorizando dar un servicio de calidad al usuario.

Debido al gran tamaño que puede llegar a alcanzar el modelo de optimización para la planificación de líneas de una ciudad como Sevilla, este trabajo se focaliza en el desarrollo de métodos de generación de un pool de líneas candidatas como una etapa previa a la resolución del modelo, el cual seleccione un subconjunto de líneas candidatas factibles, que posteriormente serán introducidas como input al modelo, de forma que sea posible reducir el tamaño del modelo y, a su vez, obtener soluciones de buena calidad en términos de demanda atendida y tiempo total de viaje.

Para ello, se desarrollan diferentes métodos, los cuales siguen distintos criterios para la selección de dicho subconjunto, y se lleva a cabo un análisis y comparativa de los resultados que cada uno de ellos permite obtener en el TNDP.

# Abstract

This Master Thesis focuses on the study of the public transport line planning problem (TNDP) and its resolution by means of an optimization model that allows obtaining the optimal design of the transport network and prioritizing a quality service to the user.

Due to the large size that can reach the optimization model for the planning of lines in a city like Seville, this work focuses on the development of methods for the generation of a pool of candidate lines as a previous stage to the generation of a pool of candidate line as a previous stage to the resolution of the model, which selects a subset of feasible candidate lines, which will later be introduced as input to the model, so that it is possible to reduce the size of the model and, in turn, obtain good quality solutions in terms of demand served and total travel time.

For this purpose, different methods are developed, which follow different criteria for the selection of such subset, and an analysis and comparison of the results that each of them allows to obtain in the TNDP is carried out.

# Índice

# ÍNDICE DE FIGURAS

# 1 INTRODUCCIÓN Y OBJETO DEL TRABAJO

En este primer capítulo, se pretende contextualizar la problemática abordada en el presente trabajo, exponiendo la situación de partida del problema estudiado, así como describir los objetivos que se persiguen mediante la realización del mismo. Además, se define la estructura del presente documento para su mejor comprensión.

## 1.1 Introducción

La creciente preocupación por el cambio climático y los elevados niveles de contaminación que actualmente se alcanzan en los núcleos urbanos, han llevado a potenciar el uso del transporte público, con el objetivo de que se convierta en el principal modo de desplazamiento habitual tanto urbano como interurbano, sobre todo en ciudades de mediano y gran tamaño con un alto flujo de tránsito.

Para alcanzar este objetivo, es imprescindible implantar un sistema de transporte público eficiente y atractivo tanto para el usuario como para las compañías operadoras, lo cual supone un gran reto, dada la complejidad que supone obtener un diseño de la red de transporte público, debido a los numerosos factores que influyen y los distintos intereses de las partes implicadas.

El desarrollo de un modelo de optimización para la planificación de líneas de transporte público tiene grandes ventajas, ya que permite determinar de forma óptima la infraestructura y recursos necesarios para garantizar que se satisface la demanda de pasajeros que desean desplazarse de un punto a otro de la red, de forma que se consiga minimizar tanto el tiempo de trayecto como los costes operacionales, de forma que el uso de la red de transporte público cumpla con los requisitos mínimos tanto desde el punto de vista del usuario como de la empresa operadora, aumentando la probabilidad del éxito de su implementación.

Sin embargo, la resolución del problema de planificación de líneas mediante un modelo de optimización tiene una elevada complejidad debido al volumen de datos que debe considerar, convirtiéndose en un problema de gran tamaño con numerosas variables y restricciones, las cuales se incrementan exponencialmente en redes de mayor envergadura.

En base a esta problemática, diversos autores han planteado estrategias para la reducción del tamaño del problema de planificación de líneas, estableciendo métodos de generación de líneas candidatas como una etapa inicial al planteamiento y resolución del modelo de optimización. Sin embargo, hasta el momento no se ha conseguido obtener un método que permita reducir el tamaño del problema garantizando al mismo tiempo que se obtienen soluciones de buena calidad.

## 1.2 Objeto del Trabajo

El objetivo del presente trabajo es la definición de un método de generación de líneas candidatas, capaz de seleccionar en una fase inicial un subconjunto del total de líneas candidatas factibles (pool de líneas) que, al ser introducidas como input al modelo de optimización usado para resolver el TNDP, permita obtener soluciones de calidad en términos de demanda atendida y tiempo de viaje de los usuarios al problema de diseño de una red de transporte público, reduciendo a su vez el coste computacional requerido para la resolución del problema.

Para ello, se proponen tres métodos diferentes para la generación de un pool inicial de líneas, los cuales se ejecutan como parte inicial del modelo de planificación considerando diferentes escenarios, con objeto de

analizar los resultados que permite obtener cada uno de ellos y realizar un estudio comparativo entre los mismos para determinar el más adecuado.

## 1.3 Estructura del Trabajo

El presente trabajo se ha estructurado en diferentes capítulos. En primer lugar, en el capítulo 2, se describe en qué consiste el problema de planificación de redes de tránsito, así como los diferentes enfoques adoptados hasta ahora para su planteamiento y resolución.

En el capítulo 3, se realiza una revisión bibliográfica de las publicaciones realizadas en los últimos años sobre los métodos de generación de líneas candidatas, una etapa previa al planteamiento y resolución del problema de planificación de líneas, con objeto de analizar los resultados obtenidos y detectar posibles mejoras como punto de partida para el desarrollo del presente trabajo.

En el siguiente capítulo, en primer lugar, se describen en líneas generales las características y requisitos del modelo de optimización para la resolución del problema de planificación de líneas. Aunque este trabajo se centra en la determinación de conjuntos de líneas candidatas (pool de líneas) --que posteriormente se introducen como inputs a un modelo de optimización (planificación de líneas) para conseguir un diseño final de la red de transporte público--, para valorar la calidad del conjunto incicial de líneas candidatas (inputs) resulta imprescindible ejecutar el modelo de planificación y observar el resultado final (outputs). Resulta por tanto necesario exponer, al menos brevemente, el cometido de este modelo. Una vez contextualizado el modelo, se procede a describir en detalle los diferentes métodos desarrollados en el presente trabajo para la generación de un pool inicial de líneas candidatas.

En el capítulo 5, se describen los diferentes escenarios y consideraciones generales establecidas para la validación de los métodos propuestos, y en el capítulo 6 se muestran los resultados obtenidos para cada uno de ellos, realizando comparativas y extrayendo conclusiones a partir de los mismos.

Por último, en el capítulo 7, se resumen las conclusiones generales que se pueden extraer tras la realización del presente trabajo.

# 2 EL PROBLEMA DE PLANIFICACIÓN DE LÍNEAS

El resultado de un problema de diseño de una red de tránsito, en general, consiste en diseñar un conjunto de líneas que debe ofrecer una buena conectividad, entendida como la posibilidad de viajar entre los puntos de mayor demanda de la red, así como una buena accesibilidad geográfica, garantizando la cobertura del espacio considerado.

La red diseñada tiene una influencia directa sobre los usuarios y operadores de transporte. Por un lado, los usuarios esperan que la red de tránsito disponga del mayor número de líneas posible, de forma que facilite el desplazamiento directo entre cualquier par de puntos, minimizando la necesidad de realizar transbordos entre líneas, y que considere unas altas frecuencias para las líneas, minimizando los tiempos de espera en las paradas (o andenes, en el caso de ferrocarriles) y los tiempos requeridos para realizar transbordos entre líneas. En caso de que no se logren estos objetivos, la calidad percibida del servicio de la red de tránsito será baja, lo que conllevará que los potenciales pasajeros opten por medios de transporte alternativos, disminuyendo así los ingresos obtenidos por la empresa operadora. Por otro lado, un mayor número de líneas y frecuencias más altas implican un mayor coste operacional.

Estos objetivos opuestos requieren que las compañías operadoras de servicios de transporte a alcanzar un equilibrio entre la calidad del servicio ofrecido y el coste de operación de la red.

Como se verá más adelante, ambos aspectos se han tenido en cuenta de diferentes maneras por parte de muchos investigadores. En la literatura disponible, se muestran los distintos enfoques adoptados para la resolución del problema de diseño de una red de transporte público (transit network design problema o TNDP), los cuales pueden clasificarse en cuatro grupos diferentes.

En las primeras etapas, algunos investigadores propusieron procedimientos heurísticos que permitían generar las rutas y determinar las frecuencias de las líneas de una forma práctica (Lampkin & Saalmans, 1967; Mandl, 1980; Furth & Wilson, 1981; Ceder, 1984; Ceder & Wilson, 1986; Shih & Mahmassani, 1994).

Tras ello, se distingue un segundo grupo, formado por trabajos que modelan el problema como un problema de programación matemática, partiendo de un conjunto de líneas candidatas definidas a priori (Bussieck et al., 1997, 2004; Goossens et al., 2006; Guan et al., 2006).

El tercer grupo presta atención a la formulación de modelos de diseño desde cero. Es decir, a partir de una red subyacente, los modelos propuestos construyen líneas mediante la unión de aristas. Para ello, se utilizan variables binarias para determinar si las aristas pertenecen o no a las líneas. Además, estos modelos precisan de varias restricciones que garanticen que las aristas seleccionadas para definir una línea mantienen la estructura adecuada, es decir, la conexión entre aristas consecutivas, la ausencia de bucles y cumplen los límites de longitud total de la línea. Este enfoque da lugar a modelos realmente complejos, los cuales son bastante difíciles de resolver mediante procedimientos exactos (Szeto & Wu, 2011; Szeto & Jiang, 2014; Canca et al., 2017, 2019).

A principios de los años noventa, comienzan a aparecer las primeras aplicaciones de técnicas metaheurísticas al problema de planificación de líneas, como por ejemplo los trabajos de (Baaj & Mahmassani, 1991; Chakroborty et al., 1995, 1998; Chakroborty & Wivedi, 2002). Las aportaciones realizadas en este grupo suelen abordar el problema de planificación de líneas tomando como partida un pequeño conjunto inicial de líneas candidatas, y aprovechan la capacidad de la metaheurística para generar nuevas líneas candidatas a partir de las inicialmente definidas, y para evaluar objetivos más complejos, difíciles de formular de manera explícita.

Las funciones objetivo consideradas generalmente en los problemas de diseño de redes de transporte urbano pueden estar orientadas al pasajero y/o al operador (Guihaire & Hao, 2008). Desde el punto de vista de los pasajeros, la red diseñada deberá cubrir la mayor superficie posible del escenario en estudio, el conjunto de

líneas obtenido deberá permitir el máximo número de viajes directos (es decir, minimizar el número de transbordos) y garantizar el mínimo tiempo de viaje (Jiang et al., 2013; Feng et al., 2018). Desde el punto de vista del operador, el principal objetivo es reducir al máximo los costes de operación y de construcción de la red, sobre todo en el caso de una red de ferrocarriles urbanos, objetivo que se alcanza reduciendo al máximo el número de líneas, sus longitudes y sus frecuencias.

La siguiente figura muestra un esquema de la clasificación de los enfoques adoptados para la resolución del problema de planificación de líneas:

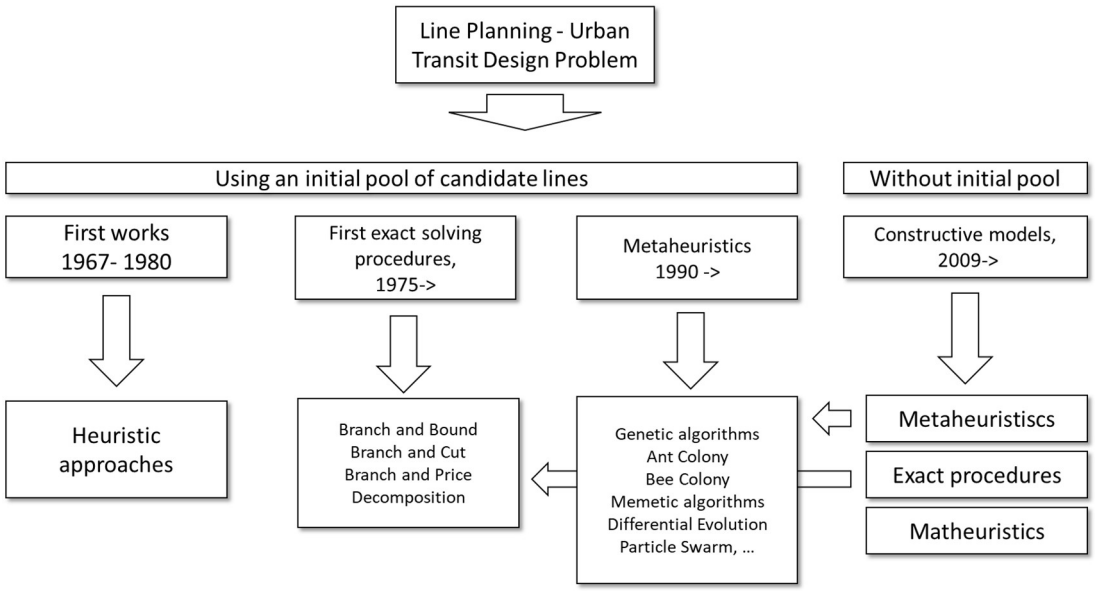


Figura 1. Enfoques para la resolución de problemas de diseño de redes de transporte público.

# 3 ESTADO DEL ARTE

Tal y como se ha indicado anteriormente, el objetivo del presente proyecto consiste en la estudio de métodos de generación de un conjunto inicial de líneas candidatas para posteriormente alimentar un modelo de optimización que, seleccionando un subconjunto de ellas proponga un diseño final de una red de tránsito.

Con la finalidad de conocer los principios y características de los métodos elaborados hasta ahora para obtener conjuntos de líneas candidatas, se ha realizado una revisión bibliográfica, que servirá como punto de partida para el desarrollo del presente trabajo.

## 3.1 Problema de Diseño de una Red de Tránsito (TNDP)

El Problema de Diseño de una Red de Tránsito (TNDP) tiene como objetivo la definición de un conjunto de líneas y sus frecuencias óptimas para satisfacer cierta demanda con un doble objetivo: minimizar el tiempo de viaje y de espera de los usuarios y minimizar los recursos necesarios, es decir, minimizar la flota y el número de líneas.

Para encontrar la solución óptima del TNDP, la mayoría de los trabajos existentes parten de una solución inicial y, mediante el uso de metaheurísticas, la transforman de forma iterativa hasta alcanzar una buena solución en un tiempo no demasiado elevado. Sin embargo, este tipo de enfoque de resolución del problema no permite evaluar la bondad de la solución en términos de optimalidad.

La resolución del TNDP mediante un modelo de optimización presentaría grandes ventajas en cuanto a la calidad de la solución, pero su elevada complejidad provoca que en la actualidad no se suela optar por este tipo de planteamiento, ya que requiere de un coste computacional elevado, debido a que parte de un conjunto muy extenso de posibles líneas candidatas, de entre las que se seleccionará el subconjunto que proporcione el mejor valor de la función objetivo.

Con objeto de reducir dicha complejidad, en este trabajo se propone realizar un paso previo a la resolución del modelo de optimización del TNDP, mediante la definición de un método de generación del conjunto inicial de líneas candidatas (pool), de forma que dicho pool de líneas se usará como conjunto potencial de líneas en el modelo para la selección del mejor subconjunto de ellas, reduciendo considerablemente la carga computacional y el tiempo de resolución requeridos.

## 3.2 Método de generación del pool de líneas candidatas

Para que el modelo de optimización, a partir del pool de líneas seleccionado previamente, consiga encontrar una solución del TNDP buena en términos de optimalidad, es de vital importancia establecer un método que genere un pool de líneas candidatas de calidad, cuyo tamaño sea suficientemente amplio como para no descartar ninguna línea interesante, pero sin que ello suponga tratar con grandes volúmenes de datos, lo que haría que el modelo resultante fuera intratable.

Para ello, es imprescindible que para la formación del pool de líneas candidatas, se tengan en cuenta los principales factores que influyen en la calidad de la solución, como pueden ser la longitud del trayecto o la demanda directa que satisface cada trayecto, entre otros.

En base a ello, se ha realizado una revisión bibliográfica de los distintos métodos desarrollados hasta ahora para la generación del un conjunto inicial de líneas candidatas para la resolución del problema de diseño de una red de tránsito.

## 3.3 Algoritmo de generación de rutas (RGA)

Baaj y Mahmassani (Baaj & Mahmassani, 1995) desarrollaron un algoritmo para la generación de rutas. Para ello, en primer lugar, se establece un esqueleto inicial de trayectos, formado por un número determinado de pares de nodos, seleccionando aquellos capaces de satisfacer una mayor demanda de forma directa, es decir, sin realizar transbordos.

Una vez establecido dicho esqueleto, éste se expande mediante la incorporación de nodos que se encuentren a una distancia de una arista hasta alguno de los nodos incluidos en el esqueleto, seleccionando aquellos más favorables, hasta llegar a formar una ruta que cumpla las restricciones definidas, relativas a la satisfacción de una demanda mínima y a que la longitud de la ruta se encuentre entre un valor mínimo y un valor máximo.

Para la selección e incorporación de los nodos durante la fase de expansión del esqueleto, estos autores contemplaron cuatro estrategias diferentes:

- Inserción de máxima demanda (MD): se selecciona aquel nodo cuya incorporación a la ruta suponga un mayor aumento de la demanda directa total satisfecha.
- Inserción de máxima demanda por tiempo mínimo (MDMT): se selecciona el nodo cuya incorporación a la ruta suponga el mayor ratio entre el aumento de la demanda total satisfecha y el aumento del tiempo de trayecto total de la ruta.
- Inserción de máxima demanda por incremento de longitud de ruta mínima (MDML): se selecciona el nodo cuya incorporación a la ruta suponga el mayor ratio entre el aumento de la demanda total satisfecha y el aumento de la longitud total de la ruta.
- Inserción de máxima demanda por coste mínimo (MDMC): sigue el mismo criterio que en la estrategia MDMT pero, además, incorporando la diferencia del coste que supone tanto al usuario como al operador la incorporación del nodo a la ruta.

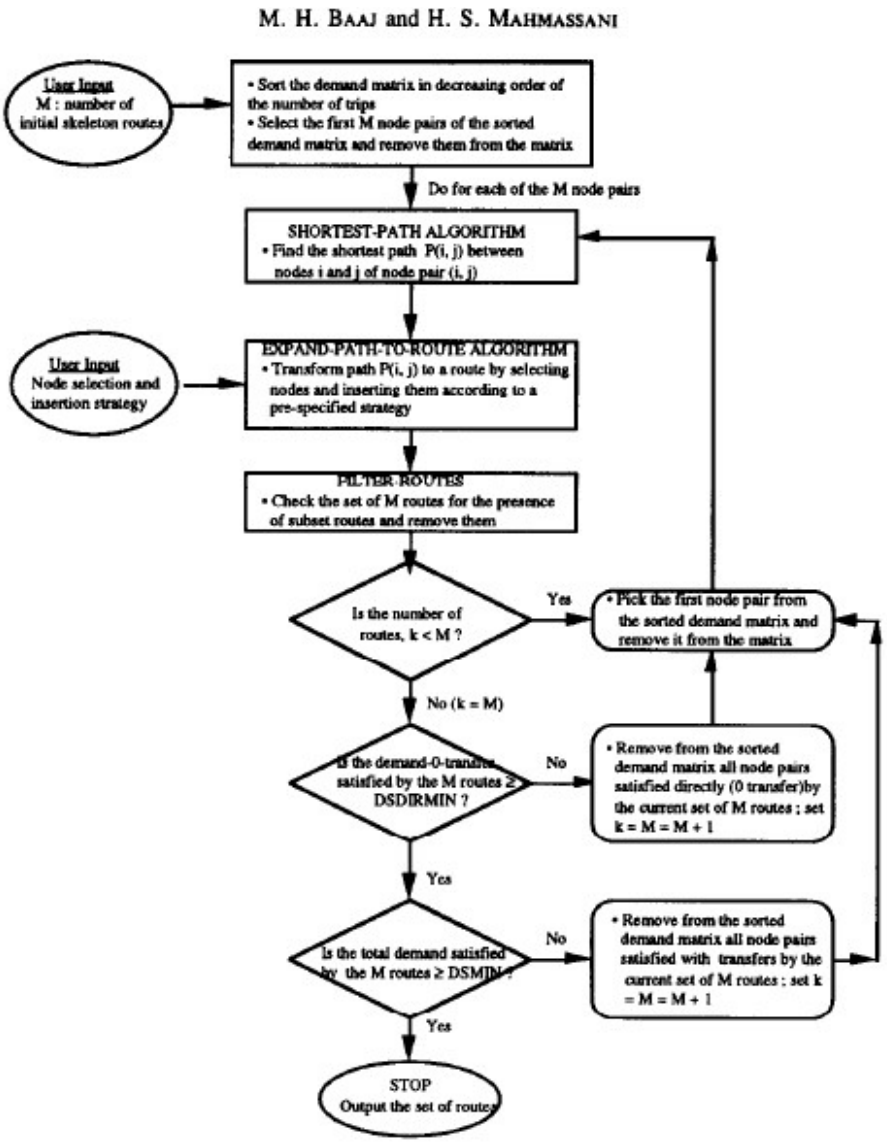


Figura 2. Route generation algorithm (RGA). Fuente: (Baaj & Mahmassani, 1995)

Cabe destacar que este procedimiento garantiza la factibilidad de la ruta durante su construcción, ya que durante la expansión se comprueba que el nodo seleccionado permita el cumplimiento de todas las restricciones del problema (frecuencia, capacidad, carga máxima del vehículo, etc) y, a su vez, comprueba que no existan líneas contenidas en otras para, en caso de que las haya, eliminarlas y reducir así el tamaño del problema.

Del mismo modo, este modelo permite que cada nodo sea atravesado por más de una ruta, de forma que la inclusión de un nodo en una ruta no lo excluye de su inserción en otra ruta, de forma que la satisfacción de la demanda de un nodo pueda ser repartida entre varias rutas.

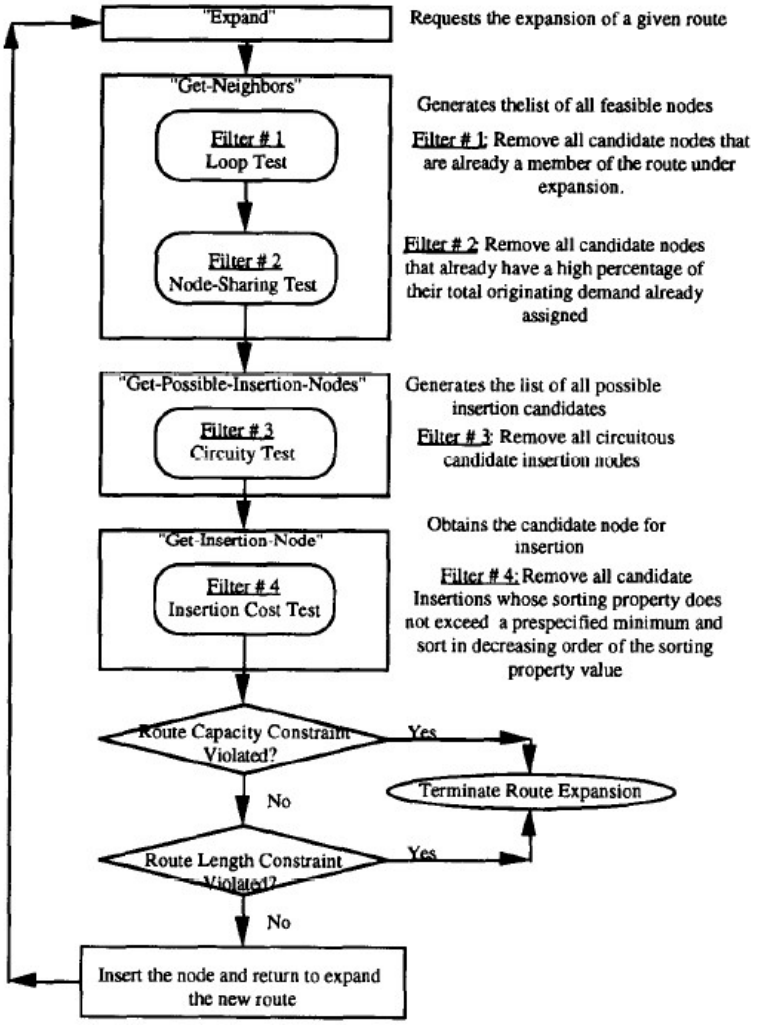


Figura 3. Factibilidad de los nodos durante la expansión de la ruta. Fuente: (Baaj & Mahmassani, 1995)

## 3.4 Algoritmo de inserción de pares (PIA)

Mauttone y Urquhart (Mauttone & Urquhart, 2009), definieron un procedimiento para el diseño de rutas de transporte público de autobús, con un objetivo equilibrado para favorecer tanto al usuario como al operador. De esta forma, se busca determinar el conjunto de rutas más cortas que supongan el menor tiempo de trayecto para el usuario, teniendo en cuenta tanto el tiempo de viaje como el tiempo de espera, y, a su vez, optimice la flota de vehículos necesaria y el número de rutas activas para dar cobertura a la demanda con la menor infraestructura posible, suponiendo los menores costes operacionales.

Para ello, propusieron el llamado Algoritmo de Inserción de Pares (PIA), el cual se basa en el método RGA definido por Baaj y Mahmassani, explicado anteriormente. En el caso del algoritmo PIA, la expansión del esqueleto inicial de rutas se lleva a cab o mediante la inserción de pares de nodos, a diferencia del RGA que inserta nodos de forma individual que se encuentren a una distancia máxima de una arista respecto a los nodos pertenecientes al esqueleto. De esta forma, los nodos candidatos a ser incluidos en la ruta pueden encontrarse a una distancia superior a una arista respecto a los nodos del esqueleto.

Para la expansión de la ruta, se evalúa cada par de nodos no incluido en el esqueleto, con objeto de conocer la diferencia entre el coste de crear una nueva ruta formada por el camino más corto entre ambos nodos y el incremento del coste que supondría la incorporación de este par de nodos a la ruta ya existente, en términos de tiempo de viaje, para seleccionar la opción de menor coste. Además, al igual que en el modelo anterior, antes de la inserción de nodos, se comprueba que se continúa cumpliendo la restricción de longitud total de la ruta y que no existen rutas contenidas en otras, de forma que se asegura la factibilidad de la solución obtenida.

## 3.5 Algoritmo de concentración de flujos (HRGA)

En el modelo diseñado por Cipriani, Gori y Petrelli (Cipriani et al., 2012), se trata de resolver el problema de diseño de la red de tránsito de una red multimodal con demanda elástica, mediante un conjunto de heurísticas. En primer lugar, mediante un algoritmo heurístico basado en la concentración de flujos, se generan un conjunto de rutas factibles y, tras ello, se incorporan en un algoritmo genético que permite seleccionar un subconjunto óptimo de éstas junto con una frecuencia asociada, ya que, según estos autores, es imprescindible que el problema de ajuste de frecuencias se resuelva de forma simultánea a la búsqueda de la ruta óptima.

En una primera etapa, un algoritmo heurístico genera tres conjuntos diferentes y complementarios de rutas factibles:

- Conjunto de rutas tipo A: formado por los trayectos más cortos que conectan los nodos con mayor demanda y no están incluidos en la red de transporte ferroviario. Para ello, se simplifica la matriz de demandas eliminando la demanda satisfecha por el sistema ferroviario y aquellos trayectos que no cumplan la restricción de longitud máxima.

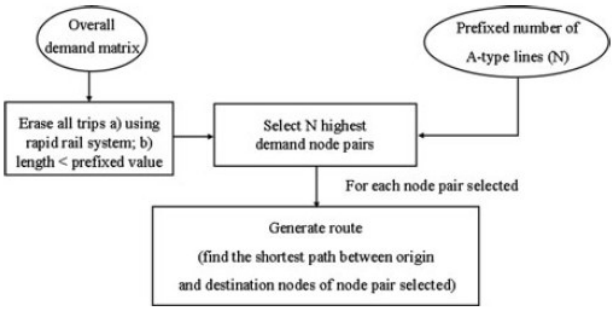


Figura 4. Generación de rutas tipo A. Fuente: (Cipriani et al., 2012)

- Conjunto de rutas tipo B: este segundo conjunto de rutas tiene como objetivo establecer una red formada por aquellas rutas que conectan los principales nodos de tránsito, normalmente en los que se encuentran estaciones de ferrocarril, de manera que sean enlaces, entendidos como el trayecto que une dos nodos, que transporten el mayor número de pasajeros posible. Para la generación de este tipo de rutas, se sigue el siguiente método, propuesto por los mismos autores en publicaciones anteriores:
    - Identificación del esqueleto de la red: consiste en un procedimiento iterativo de asignación de la demanda todo o nada a los trayectos disponibles en la red.

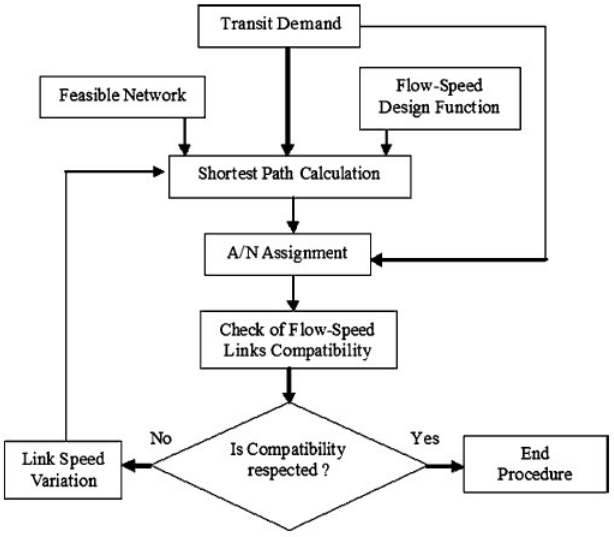


Figura 5. Identificación del esqueleto de rutas tipo B. Fuente:(Cipriani et al., 2012)

    - Generación de rutas: A partir del esqueleto obtenido, se definen las rutas mediante una estrategia de selección e inserción de enlaces según el volumen de pasajeros de los mismos,

siguiendo para ello los siguientes pasos:

- Paso 1: Identificación del punto de partida de cada ruta, que puede ser definida por el planificador de rutas identificados automáticamente según el enlace con mayor volumen de pasajeros.
- Paso 2: Selección del primer enlace a incorporar en la ruta, que será aquel que posea un mayor volumen de pasajeros.
- Paso 3: Ampliación de las rutas insertando otros enlaces, conectados con el primer enlace incorporado en la ruta.

Es importante destacar que, durante la generación de rutas, se garantice la factibilidad de la solución, ya que se deben cumplir restricciones referidas a un valor mínimo de volumen de pasajeros que debe transportar y que cada nodo incorporado en la ruta debe estar más cerca del destino que el nodo que lo precede.

- Conjunto de rutas tipo C: una vez generadas las rutas tipo A y tipo B, se añaden al conjunto de las rutas de autobuses ya existentes en la realidad, sobre la que parte este modelo de mejora, las cuales forman las rutas tipo C, con el objetivo de verificar que todas ellas en conjunto cumplen las restricciones establecidas para el modelo.

## 3.6 Algoritmo de árbol de mínima expansión (MST)

Gattermann, Harbering y Schöbel (Gattermann et al., 2017) definieron un modelo para la generación del pool inicial de líneas candidatas, como primera fase para la resolución del problema TNDP mediante un modelo de optimización.

En su modelo, buscan seleccionar un pool formado por un conjunto de K líneas, que no contengan bucles, de forma que su tamaño (K) garantice la factibilidad del modelo de planificación de líneas y que tengan una alta influencia en la calidad de la solución.

Para ello, establecen un modelo constructivo de generación de líneas, de manera que un algoritmo genera trayectorias entre nodos de forma iterativa creando árboles de mínima expansión (MST) y añadiéndolos al pool, hasta que encuentra una línea ya incluida en el pool y se detiene.

El procedimiento para la generación de líneas es el siguiente:

1. Identificar los nodos terminales, que son aquellos nodos por los que pasa un mayor número de aristas.
2. Determinar el árbol de mínima expansión entre las hojas
3. Generar líneas que conecten dos hojas, dos terminales o terminal-hoja.

Para la realización del MST, a cada trayecto se le asigna su capacidad y un peso, que corresponde a su longitud, a excepción de los trayectos de mayor demanda, a los cuales se les asigna un coste 0, favoreciendo así su selección.

En cada iteración, se comprueba si el conjunto de líneas es factible en cuanto a restricciones de capacidad. En caso afirmativo, el método se detiene. En caso contrario, se disminuyen las capacidades de los trayectos que suponen la no factibilidad y se les otorga peso 0 para que sean seleccionados en el siguiente MST.

Una vez obtenido el primer MST, se pasa a un segundo algoritmo para la generación de líneas. Para ello, se añaden aristas adicionales a las líneas que no eran factibles hasta alcanzar el valor mínimo exigido para la longitud de la ruta que comienza o finaliza en un terminal.

Debe tenerse en cuenta que una línea sólo se añadirá al pool si ésta contiene al menos un trayecto de alta demanda y si no se satura demasiado la capacidad del resto de los trayectos que lo forman.

Se construirán sucesivos árboles de expansión hasta que se hayan incluido todos los trayectos de alta demanda, o no queden líneas disponibles que se puedan añadir.

Para validar el método propuesto de generación de pool, realizan varios experimentos con diferentes tipos de

redes y se estudian los resultados obtenidos en el modelo de planificación de rutas tras la aplicación del método de generación de pool propuesto.

En primer lugar, utilizan 100 redes en forma de estrella con diferentes tamaños, eligiendo de forma aleatoria el número de nodos y aristas de cada una de ellas entre cuatro rangos, que van desde 0 hasta un número máximo (de hasta 500 nodos y aristas), con objeto de evaluar la calidad de la solución óptima obtenida y el tiempo de ejecución requerido. Los resultados obtenidos muestran que se puede alcanzar una buena aproximación a la solución óptima del problema incluyendo sólo una parte del total de las líneas que forman el pool, lo que permite reducir el tiempo de ejecución necesario, obteniéndose niveles de calidad y tiempos de ejecución aceptables en los casos de las redes de mayor tamaño.

Tras ello, con objeto de valorar la aplicabilidad de este método en un entorno real, realizaron experimentos utilizando una red parecida a la red de ferrocarriles alemanes, formada por 250 nodos y 326 aristas, estableciendo diferentes parámetros, tales como la distancia mínima entre nodos origen y destino, el número mínimo y máximo de líneas que pasan por cada nodo no terminal, el número mínimo y máximo de trayectos que debe formar cada línea o el porcentaje de trayectos de mayor demanda incluidos, entre otros. Para ello, probaron un total de 624 combinaciones, obteniéndose en cada una de ellas un pool de líneas diferente.

En la siguiente figura, se muestran los valores que permite obtener cada pool generado para cada objetivo del modelo, que son minimizar costes y maximizar número de viajes directos, mostrando los parámetros considerados para su generación:

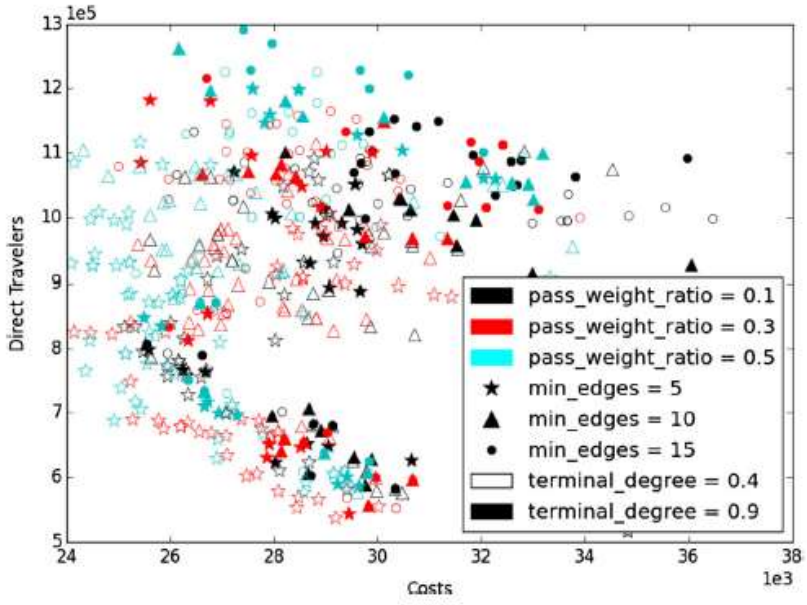


Figura 6. Resultados del modelo de planificación de líneas para cada pool generado

Los resultados obtenidos permiten extraer las siguientes conclusiones:

- Aumentar el número mínimo de trayectos que debe contener cada línea, permite realizar un mayor número de viajes directos.
- Aumentar el número de líneas que debe pasar por cada trayecto entre nodos, reduce el número de viajes directos y aumenta los costes.
- Permitir únicamente líneas entre nodos que se encuentren a una distancia mínima permite mejorar el número de viajes directos.
- Un menor número de nodos terminales permite obtener trayectos más largos, por lo que mejora el número de viajes directos y reduce los costes.
- Los parámetros del porcentaje de uso de líneas preferidas y el número máximo de líneas que deben pasar por un trayecto, no tienen efectos en los valores objetivo, aunque sí influyen en la configuración

de las líneas.

- Un pool de líneas formado por un mayor número de líneas, no necesariamente permite obtener líneas de mejor calidad en cuanto a optimalidad de la solución. Sin embargo, los parámetros seleccionados para la generación del pool y su valor, tienen una gran influencia en la calidad de la solución.

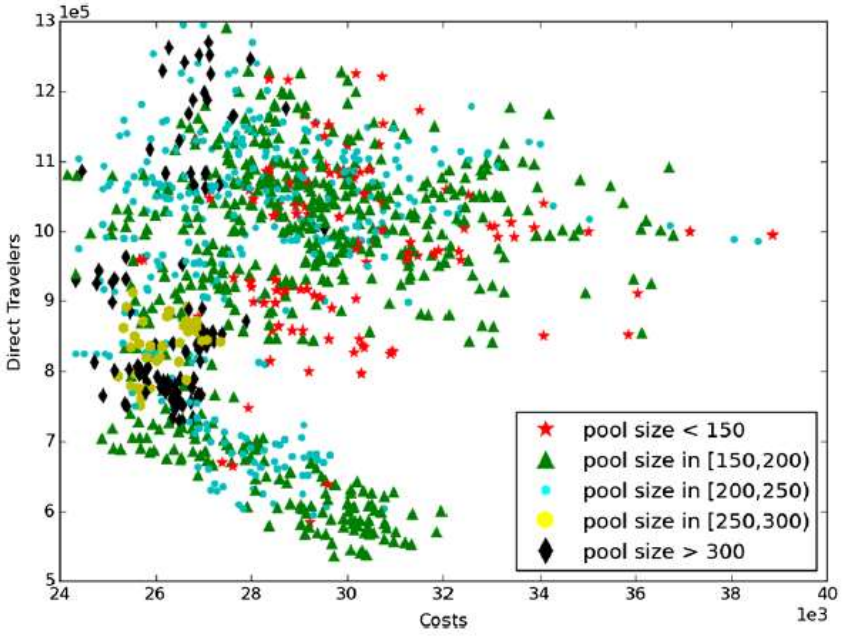


Figura 7. Resultados obtenidos para cada pool en función de su tamaño (número de líneas)

## 3.7 Conclusiones

Tras realizar la revisión de la literatura expuesta anteriormente, se puede observar que, pese a lo esfuerzos realizados desde hace décadas para la reducción del tamaño del problema de planificación de líneas, mediante la selección de un conjunto de líneas candidatas a introducir en el modelo de optimización, con objeto de facilitar su resolución, no se ha llegado a establecer un procedimiento que, en un tiempo de ejecución aceptable, permita obtener resultados de calidad en cuanto a optimalidad de la solución, por lo que continúa siendo un problema abierto.

# 4 MÉTODOS PROPUESTOS

En este capítulo, en primer lugar, se describen los objetivos y la estructura general del modelo de optimización para el problema TNDP utilizado en el presente trabajo, sus inputs, variables de decisión, restricciones definidas y los resultados que permite obtener.

Una vez contextualizado el modelo, se procede a describir detalladamente cada uno de los métodos de generación de pool de líneas candidatas desarrollados, los cuales serán introducidos en el modelo para que, a partir del mismo, obtenga la solución óptima del problema del TNDP.

Cabe destacar que, para que el pool obtenido como resultado de cada uno de los métodos definidos pueda ser introducido en el modelo, se requiere que tenga una determinada estructura, por lo que, por último, se describen las características y estructura que deberá tener el pool de líneas candidatas obtenido en cada método para que pueda ser utilizado como datos de partida para el modelo matemático de optimización de resolución del TNDP.

## 4.1 Estructura del Modelo TNDP

El modelo matemático para la resolución del problema de planificación de líneas estudiado en el presente trabajo, tiene el principal objetivo de diseñar una red de autobuses que permita que el mayor número de pasajeros pueda desplazarse a través de la misma en el mínimo tiempo de viaje posible. Se trata de un modelo bimodal, ya que se considera que los desplazamientos se podrán realizar a pie y/o a través de la red de líneas de autobuses. De este modo, todos los trayectos pueden ser recorridos por los peatones, y un subconjunto de ellos (que podría coincidir con el total), además, podrán ser recorridos utilizando las líneas de autobuses. De este modo, la red peatonal se utilizará como base para seleccionar aquellos trayectos que formarán parte de la red de autobuses.

Para ello, se define un modelo con una estructura multicapa bimodal, en la que la red peatonal se encontrará en la capa base y, a partir de ella, se irán definiendo múltiples capas, de forma que cada una de ellas contenga una línea diferente del conjunto incicial de líneas candidatas (de entre las que finalmente se seleccionarán sólo algunas, formando la red final de líneas de transporte público). Además, estas capas deberán estar unidas entre sí para permitir los transbordos de pasajeros entre los diferentes modos de transporte y entre las diferentes líneas para realizar el desplazamiento deseado, todo ello conformando una supercapa que almacenará el conjunto completo de líneas de la red de autobuses.

Para que el modelo pueda calcular los transbordos entre las diferentes líneas y considere la realización de desplazamientos con trayectos a pie y trayectos en bus, es necesario que pueda diferenciar si los nodos y arcos forman parte de la red peatonal o de las diferentes líneas de la red de autobús. Por ello, los elementos de cada capa (nodos y arcos) tendrá una identificación diferente, aunque se refieran a la misma ubicación y al mismo trayecto. La red peatonal mantendrá la identificación original de nodos y arcos, pero las distintas líneas de la red de autobús deberán ser almacenadas con una identificación distinta para sus nodos y arcos. Así, cada nodo y arco de cada capa tendrá un código único que representa una copia del nodo o arco original y defina la capa a la que pertenece. A este proceso se le denomina codificación.

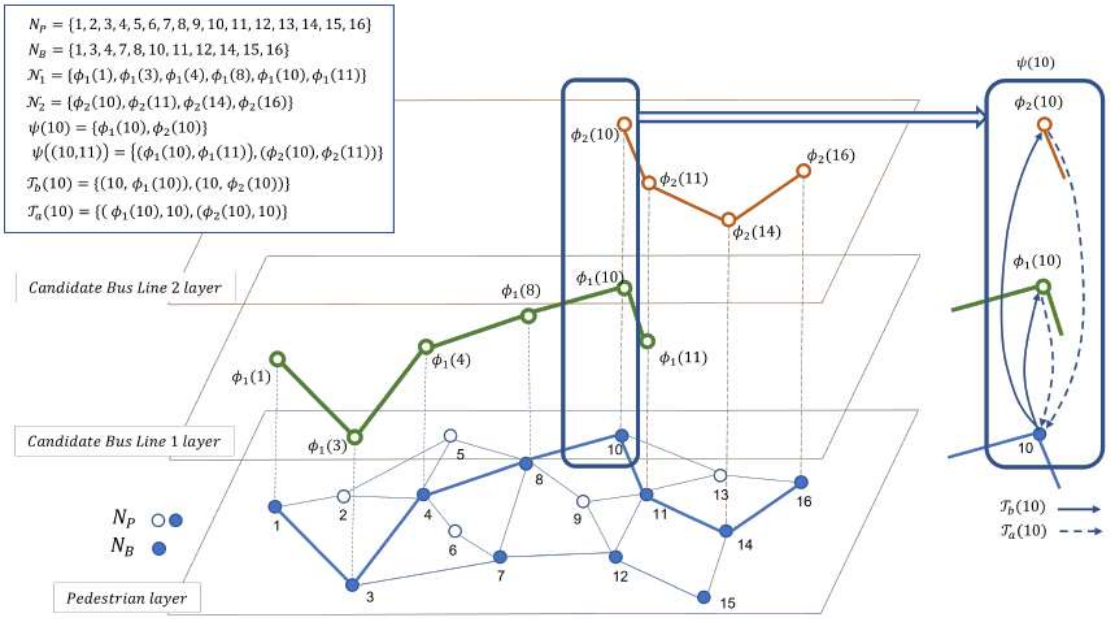


Figura 8. Estructura de red multicapa bimodal. Fuente:(Canca et al., 2022)

De este modo, el modelo matemático pretende hallar la configuración de las líneas de autobuses y la frecuencia óptima de cada una de ellas que permita minimizar el tiempo total de viaje de los desplazamientos realizados a través de la red, por lo que considera las siguientes variables:

- $x_{ij}^{w}$: Variables de flujo en cada arco, tanto de autobús como peatonales. Se tendrá una variable para cada arco $(i, j)$ y cada par origen-destino $w$.
- $y_l$: Variables binarias que representan si cada línea candidata $l$ es seleccionada o no como línea de la red. Se tendrá una variable para cada línea candidata a ser seleccionada.
- $f_l$: Variable entera que representa la frecuencia de cada línea candidata, medida en autobuses por hora. Se tendrá una variable para cada línea candidata.
- $h_l$: Variable entera que representa el headway de cada línea candidata, medido en minutos. Se tendrá una variable para cada línea candidata.

Las restricciones que debe cumplir la solución óptima son las siguientes:

- Restricciones de balance en cada par origen-destino, que garantizan que los pasajeros que inician su viaje en un origen deben ser los mismos que llegan a un destino y restricciones de balance nulo en nodos.
- La frecuencia de cada línea deberá estar entre un valor mínimo y un valor máximo. El primero está relacionado con la calidad del servicio percibida por los usuarios, que probablemente esperarán un servicio al menos cada 10 o 15 minutos. El segundo suele ser fijado por los operadores, en función de la flota disponible y de la demanda media, variando habitualmente entre 20 y 30 autobuses/hora.
- Máximo número de líneas que se desea diseñar en la red, en función de los recursos disponibles por parte del operador.
- Restricciones de capacidad en cada arco: la frecuencia de cada línea vendrá determinada según la capacidad del volumen de pasajeros que los autobuses podrán transportar, de forma que se pueda garantizar que se satisface toda la demanda.

Como se puede observar, el tamaño del problema aumenta considerablemente a medida que la red incluye más nodos, dado que el número de variables depende del número de líneas candidatas a formar parte de la red, por lo que la resolución de un problema de planificación de líneas en escenarios grandes supone unos costes computacionales muy elevados.

Por ello, mediante el presente trabajo se pretende desarrollar un método de generación de un pool inicial de

líneas candidatas, que permita introducir al modelo matemático únicamente las más interesantes para el objetivo buscado, reduciendo de esta forma el número de variables y con ello el tamaño y la complejidad del problema, sin que ello suponga una merma de la calidad de la solución obtenida en términos de optimalidad.

## 4.2 Métodos de generación de pool de líneas candidatas

Como se ha indicado anteriormente, en el presente trabajo se busca la definición de un método de generación de un pool inicial de líneas candidatas que sirva como punto de partida para la resolución del problema TNDP mediante un modelo de optimización, que permita reducir su tamaño y obtener de una solución de buena calidad en términos de optimalidad.

Para ello, se han diseñado tres métodos de generación de líneas candidatas, con el objetivo de evaluar en distintos escenarios los resultados obtenidos para el problema TNDP mediante el modelo de optimización establecido, recibiendo como punto de partida el conjunto de líneas candidatas inicial obtenido a partir de cada uno de los métodos diseñados, y realizar un estudio comparativo entre ellos.

La principal diferencia entre estos tres métodos reside en la estrategia seguida para la selección de las líneas candidatas más interesantes a incluir en el pool de entre todas las lineas posibles, la cual dependerá de los criterios considerados para la priorización de las líneas y para la eliminación de las que, a priori, parecen menos beneficiosas para el objetivo de optimización del problema.

A continuación, se describen en detalle cada uno de los métodos desarrollados.

### 4.2.1 Método 1. Poda simple

En primer lugar, se crean los grafos de la red peatonal y de la red de bus, que permitirá calcular un determinado número de los caminos más cortos que unen cada origen con cada destino de la red, siempre que dichos nodos se encuentren a una distancia mínima. Los pares que se encuentren muy próximos entre sí no interesan para diseñar la red de buses, ya que se considera que los usuarios van a ir andando siempre, por lo que no se tendrán en cuenta estos desplazamientos.

De esta forma, para cada par de nodos de la red, se obtiene un conjunto de los caminos más cortos para desplazarse desde un origen a un destino, pasando por otros nodos que forman parte de la red. Se guardarán únicamente aquellos caminos cuyo tiempo de trayecto total se encuentran entre un valor mínimo y un valor máximo, en base a la velocidad media establecida para el recorrido en autobús, permitiendo descartar aquellas rutas demasiado cortas o demasiado largas, las cuales no serán rentables para el operador ni para el usuario.

Una vez se dispone del conjunto de todas aquellas posibles rutas entre cada origen y cada destino de la red, se procede a aplicar los criterios de poda establecidos para el Método 1, los cuales tienen el objetivo de descartar aquellas rutas que puedan estar contenidas en otras que a priori sean más interesantes.

Para ello, se han definido dos posibles métodos de poda:

- PRUNEMETHOD 1: Usando este método de poda, se busca eliminar aquellos subconjuntos de nodos que ya se encuentren contenidos en otra línea más larga.
- PRUNEMETHOD 2: Mediante este método de poda, se busca eliminar aquellas subcadenas de nodos contenidas en otra línea más larga, manteniendo el mismo orden en la sucesión de dichos nodos.

A su vez, ambos métodos de poda pueden aplicarse a la totalidad del conjunto de líneas candidatas, o bien exceptuando a un pequeño porcentaje de ellas, con el objetivo de comparar sus resultados y analizar si se están eliminando rutas que, aunque cumplan los requisitos de poda establecidos para su eliminación, quizás pueda ser interesante su incorporación al pool.

Es por ello por lo que se realizarán distintas pruebas, excluyendo de la fase de poda a diferentes porcentajes del total de líneas generadas, de forma aleatoria, con objetivo de analizar los resultados obtenidos y extraer conclusiones al respecto.

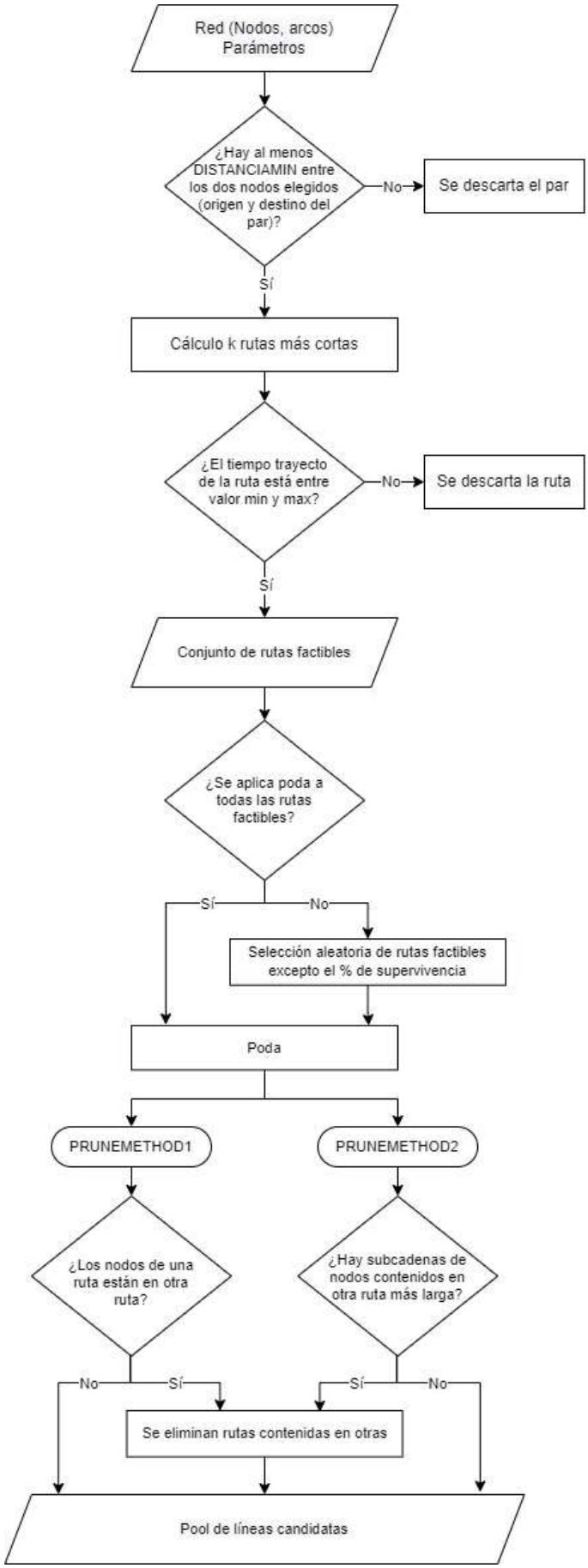


Figura 9. Esquema Método 1. Poda simple

Cabe destacar que este método no tiene en cuenta la demanda directa que satisface cada línea para realizar la poda. Únicamente considera el tiempo de trayecto de cada línea, de forma que se incluyan en el pool de líneas candidatas aquellos caminos más cortos que permitan desplazarse desde cada origen a cada destino, evitando la duplicidad de líneas. Sin embargo, es posible que con este método se estén descartando líneas que puedan satisfacer un gran volumen de demanda, de forma que, entre los nodos de mayor demanda, las líneas activas puedan ir muy saturadas. Por ello, en los siguientes métodos se incluye la demanda como uno de los criterios para la selección de líneas candidatas a incluir en el pool.

### 4.2.2 Método 2. Poda por demanda basada en ordenación de nodos

Este método se caracteriza principalmente por considerar la demanda directa satisfecha por cada línea candidata como criterio determinante para la selección de las líneas a incluir en el pool de líneas candidatas, el cual servirá como punto de partida para la resolución del modelo de optimización establecido para la obtención de la solución óptima del problema de planificación de rutas.

De este modo, una vez que se han calculado los caminos más cortos entre cada par origen-destino de la red de transporte, se eliminan aquellos que excedan el tiempo máximo de viaje establecido, así como los que sean demasiado cortos, al igual que se realiza en el Método 1 explicado anteriormente.

Además, se determinó que sería interesante evitar líneas que incluyan muchas paradas. Los métodos diseñados únicamente consideran el tiempo de recorrido entre las paradas para el descarte de líneas a incluir en el pool, pero no tienen en cuenta el tiempo necesario para la carga y descarga de pasajeros en cada una de las paradas, lo cual aumenta considerablemente el tiempo total de viaje de cada línea. Por ello, en el Método 2, se limita el número máximo de nodos que debe tener cada línea.

Una vez descartadas las líneas que no cumplan con las limitaciones del tiempo de trayecto y del número máximo de nodos, las líneas se ordenan de mayor a menor de acuerdo con la demanda directa que podrían atender en el caso de ser seleccionadas para su incorporación al pool, considerando para ello todos los viajes que los pasajeros desean realizar entre cada par de nodos que forma parte de la línea, incluyendo ambas direcciones. Las líneas con una alta demanda acumulada son líneas candidatas interesantes, ya que pueden mover de forma directa un alto número de pasajeros sin necesidad de transbordos, por lo que se priorizarán a la hora de ser seleccionadas para su incorporación en el pool de líneas candidatas.

Cabe destacar que, para definir un tamaño de pool adecuado, es necesario garantizar que todos los nodos están suficientemente representados en la red de transporte, y que por cada uno de ellos pasan diferentes líneas que permitirán su conexión con suficientes puntos de la red.

Para ello, de forma simultánea a la ordenación de líneas de mayor a menor demanda, se contabilizarán y almacenarán el número de líneas que atraviesan cada nodo de la red, con el objetivo de evitar la eliminación de una línea si el valor más bajo del contador de líneas que atraviesan cada nodo de la línea cae por debajo de un determinado valor predefinido. De este modo, todos los nodos de la red estarán atravesados por al menos un número determinado de líneas candidatas del pool final, considerando por lo tanto que todos los nodos están suficientemente representados.

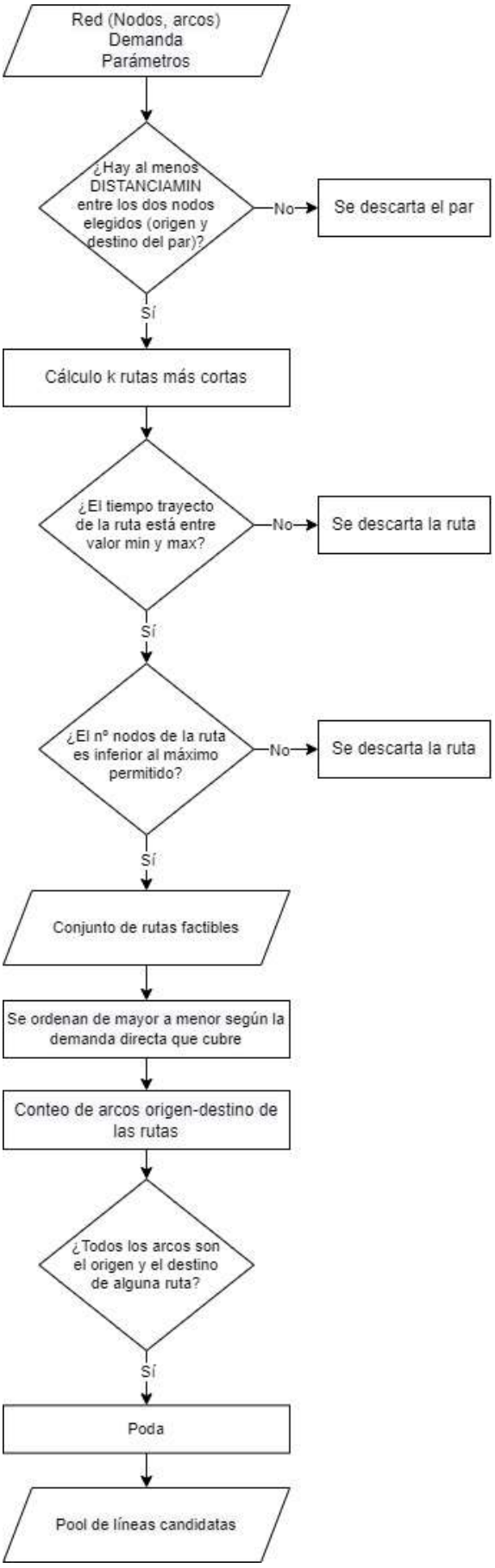


Figura 10. Esquema Método 2. Poda por demanda basada en ordenación de nodos

Cabe destacar que es posible que determinados nodos precisen estar más representados que otros, sobre todo aquellos con un mayor volumen de pasajeros. Sin embargo, este método establece un mismo valor mínimo de líneas que deben contener cada nodo, sin distinguir de qué nodo se trata ni qué volumen de demanda soporta, aunque estas condiciones particulares podrían ser implementedas de forma inmediata en el código desarrollado.

### 4.2.3 Método 3. Poda por demanda basada en ordenación de pares OD

El Método 3 sigue la misma estructura que el Método 2. La principal diferencia reside en que, en lugar de evaluar la representación de cada nodo en el pool para determinar el tamaño del pool, contabilizando el número de líneas candidatas que incluye dicho nodo, se opta por contabilizar la representación de cada par origen-destino en el pool.

Con ello, se busca garantizar que al menos una línea del pool realiza el trayecto entre cada par origen-destino de la red de forma directa. Es decir, que todos los trayectos de la red están incluidos en al menos una de las líneas candidatas que forman el pool a introducir en el modelo de optimización para la resolución del problema de planificación de rutas.

Este método permite que todos los pareas origen-destino de la red sean evaluados en el modelo de optimización, asegurando que no se descarta ninguno en la fase previa de generación del pool que finalmente pueda proporcionar resultados de buena calidad.

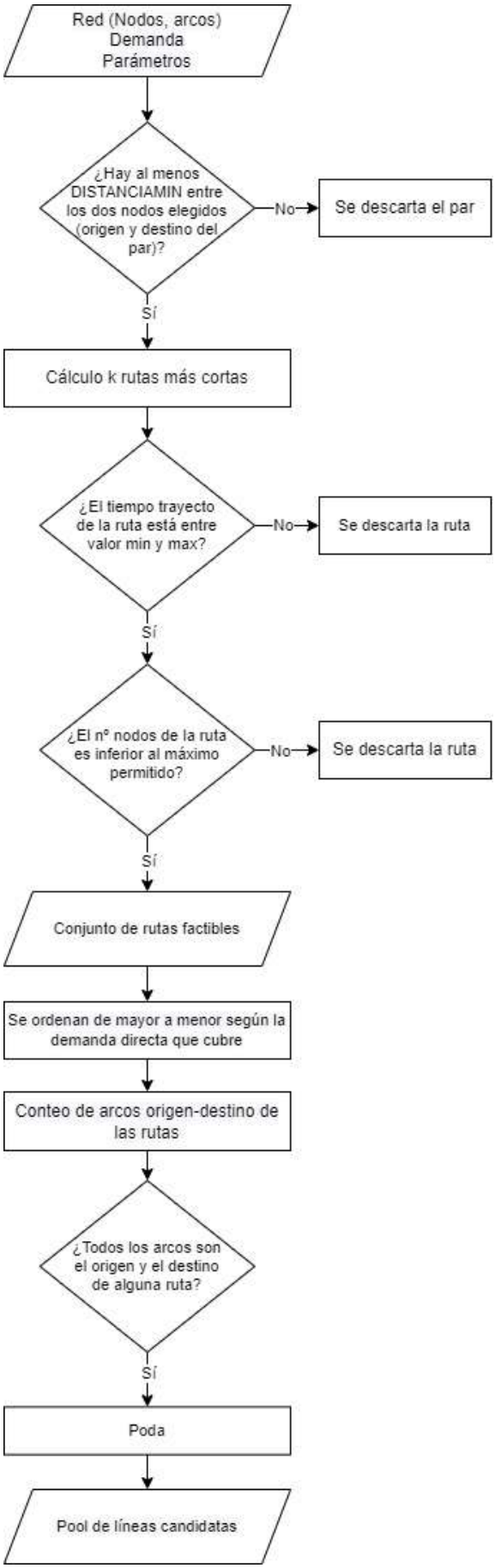


Figura 11. Esquema Método 3. Poda por demanda basada en ordenación de pares OD

## 4.3 Preparación del pool como input al Modelo

Para que el pool obtenido como resultado de los métodos anteriores pueda ser introducido en el modelo matemático y servir como punto de partida para la resolución del problema de planificación de líneas, será necesario disponer de la siguiente información relativa a cada una de las líneas candidatas incluidas en el mismo:

- Nodos por los que pasa
- Trayectos entre pares de nodos origen-destino que la forman
- Tiempo total de trayecto, calculado en función de la velocidad establecida como parámetro del modelo.

Asimismo, será necesario contabilizar el número de líneas incluidas en el pool que tengan el mismo origen y el mismo destino.

Para que el modelo matemático definido pueda disponer de dicha información asociada a las líneas candidatas y, a partir de la misma, obtener la solución óptima del problema de planificación de líneas, el pool obtenido como resultado de los diferentes métodos propuestos deberá ser almacenado siguiendo la estructura descrita a continuación:

- Diccionario Nodos-Líneas:
    - Clave: tupla origen-destino y nº identificación de la línea
    - Valor: lista formada por los nodos por los que pasa la línea

```
D_NodosLineasBus = {((1, 2), 2):[1, 5, 6, 7, 3, 2]
                    ((1, 2), 3):[1, 5, 9, 10, 6, 2] …}
```

Figura 12. Ejemplo Diccionario Nodos-Líneas

- Diccionario Arcos-Líneas:
    - Clave: tupla origen-destino y nº identificación de la línea
    - Valor: lista formada por todos pares de nodos origen-destino que forman la línea, sólo en un sentido.

```
D_ArcosLineasBus = {((1, 2), 2):[(1, 5), (5, 6), (6, 7), (7, 3), (3, 2)]
                    ((1, 2), 3):[(1, 5), (5, 9), (9, 10), (10, 6), (6, 2)] …}
```

Figura 13. Ejemplo Diccionario Arcos-Líneas

- Diccionario Longitud-Líneas:
    - Clave: tupla origen-destino y nº identificación de la línea
    - Valor: longitud total de la línea, entendida como tiempo de trayecto, en función de la velocidad establecida como parámetro del modelo.

```
D_LongLineasBus = {((1, 2), 2):5399.646001610554
                   ((1, 2), 3):7726.503594387588 …}
```

Figura 14. Ejemplo Diccionario Longitud-Líneas

- Diccionario Número de Líneas:
    - Clave: tupla origen-destino
    - Valor: número de líneas del pool que tienen dicho origen y destino

```
D_NumeroLineasBus = {(1, 2):8
                     (1, 3):1 …}
```

Figura 15. Ejemplo Diccionario Número de Líneas

Tal y como se ha explicado al principio de este apartado, el modelo utilizado para el diseño de la red es bimodal,

ya que considera que los trayectos se podrán realizar tanto a pie como a través de la red de autobús, por lo que sigue una estructura multicapa, en la que la capa base representa la red peatonal y se generará una capa para cada línea incluida en el conjunto de líneas bus candidatas. En cada una de las capas, se asignará una identificación diferente a cada nodo y a cada arco, con objeto de facilitar la modelización y la correcta resolución del problema TNDP. Por este motivo, el pool de líneas candidatas de la red de autobús obtenido como resultado de los métodos desarrollados, será codificado y posteriormente almacenado, siguiendo la estructura anteriormente descrita, de forma que el modelo de optimización reciba correctamente la información contenida en el pool de líneas, necesaria para la resolución del problema TNDP.

# 5 CASOS DE ESTUDIO

Con objeto de evaluar la capacidad de los métodos propuestos en el presente trabajo para la selección de un subconjunto de líneas candidatas que permita reducir el tamaño del problema de planificación de líneas y obtener soluciones de buena calidad, se procede a realizar pruebas mediante su aplicación en diferentes escenarios, de forma que se realice un análisis comparativo de los resultados obtenidos en cada uno de ellos y sea posible extraer conclusiones sobre los mismos, valorando cuál de ellos permite obtener mejores resultados y, en general, si finalmente la utilización de estos métodos en una fase inicial realmente permite mejorar la resolución del problema de planificación de líneas.

A continuación, en primer lugar, se describen los escenarios establecidos para evaluar el modelo de resolución del TNDP aplicando cada uno de los métodos de generación de pool propuestos, así como las consideraciones de partida establecidas y, por último, se describen los resultados obtenidos para el problema TNDP con cada uno de los métodos en los diferentes experimentos, realizando una comparativa final que facilitará la extracción de conclusiones sobre la idoneidad del uso de los mismos.

## 5.1 Definición de Escenarios

Para evaluar los métodos propuestos, se han establecido diferentes escenarios que puedan asemejarse a las características de una red de autobuses urbana que precise cubrir los desplazamientos entre diferentes puntos del centro urbano de una ciudad de tamaño mediano, como podría ser Sevilla.

Para ello, se ha considerado una red de partida formada por 16 nodos equidistantes entre sí, que forma una malla cuadrada de 4x4 nodos, que abarca una superficie de unos 4.000 $m^2$. En esta red, cada nodo está conectado a su sucesor en dirección vertical y horizontal, obteniéndose 240 pares origen-destino.

A partir de esta red inicial, se han generado diferentes redes, cuya distancia entre sus nodos varía, eliminando la equidistancia existente en la red original. Para ello, se han generado de forma aleatoria 10 redes diferentes, mediante el desplazamiento de las coordenadas originales de cada nodo alrededor de un determinado radio, que corresponde a la mitad de la distancia establecida inicialmente entre los nodos.

En la siguiente figura, se muestra a modo de ejemplo una de las redes definidas, indicando en color azul la ubicación de los nodos de la red original, y en naranja la nueva localización de los nodos de la red considerada para la generación de escenarios de validación.

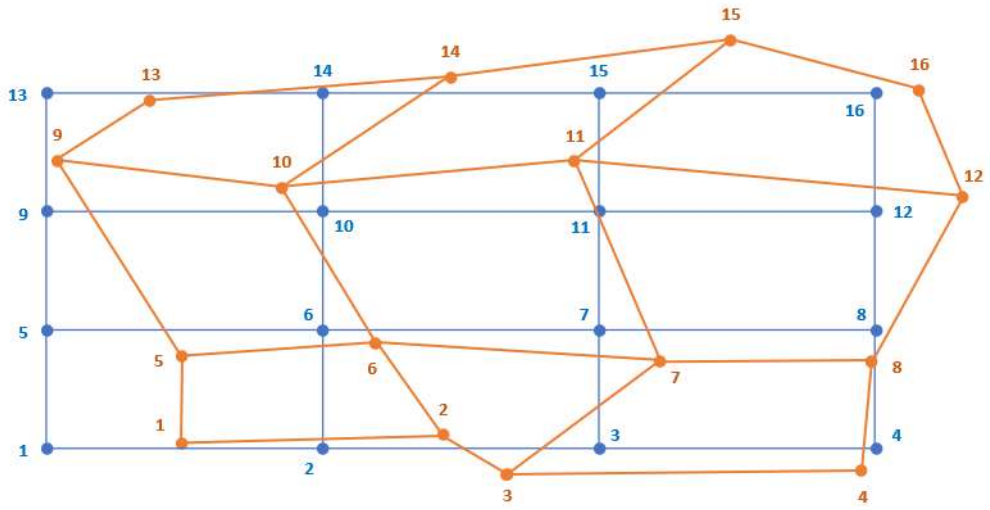


Figura 16. Generación de redes de nodos para la definición de escenarios

Por otro lado, se han definido diferentes matrices de demanda, las cuales recogen el número de pasajeros que desean desplazarse entre todos los pares origen-destino de la red. Para ello, se ha definido una matriz de demanda inicial y, a partir de ella, se ha escalado con aleatoriedad para obtener 10 matrices de demanda que representan un volumen de desplazamientos de entre entre 3.500 y 7.500 desplazamientos, el cual se considera acorde al tamaño de la red y a la superficie que cubre.

Combinando las 10 redes y las 10 matrices de demanda origen-destino distintas, se han generado un total de 100 escenarios diferentes, los cuales se incluirán como datos de entrada para la generación del pool de líneas candidatas con cada uno de los métodos desarrollados, y a continuación, se procederá a la resolución del modelo matemático considerado para el problema de planificación de líneas. Los resultados obtenidos en cada una de las pruebas se almacenarán para su posterior análisis y extracción de conclusiones sobre los métodos desarrollados.

## 5.2 Consideraciones

Antes de iniciar las pruebas para evaluar los resultados obtenidos a partir de cada método desarrollado en cada uno de los diferentes escenarios diseñados, se indican a continuación los parámetros establecidos, tanto para la generación del pool inicial, como para la resolución del TNDP mediante el modelo de optimización, los cuales se deben tener en cuenta para una mayor comprensión de los resultados que se obtengan.

### 5.2.1 Parámetros establecidos para la generación de pools

A continuación, se exponen los principales parámetros y supuestos que se han establecido para cada uno de los métodos de generación de pool propuestos en el presente trabajo:

Método 1.

- Los nodos origen y destino de cada par OD deberán estar a una distancia de al menos 400m para que dicho par OD sea considerado en el problema. En aquellos pares OD en los que la distancia del origen al destino sea menor a ese límite se considera que el usuario realizará el desplazamiento a pie, por lo que se eliminan.
- Para cada par OD, se calcularán los 10 caminos más cortos para ir desde el nodo origen hasta el nodo destino, en términos de longitud. Estos caminos limitarán las posibilidades de desplazamiento de los usuarios sobre la red. El cálculo de caminos se lleva a cabo para evitar que el número de variables del problema se dispare, dificultando su resolución.
- Cada línea deberá tener una longitud total mayor a 3.000m y menor a 65.000m (solo en una dirección) para que pueda ser incluida en el pool.

Método 2.

- Para cada par OD, se calcularán los 10 caminos más cortos para ir desde el nodo origen hasta el nodo destino, en términos de tiempo, por lo que dependerá de la velocidad establecida (parámetro del modelo).
- Cada línea del pool deberá contener como máximo 9 nodos, para evitar tener líneas con muchas paradas y deberá suponer un tiempo total de trayecto de 60 minutos como máximo.
- Cada nodo de la red deberá estar contenido en al menos 25 líneas del pool. Una vez alcanzado este valor para todos los nodos, no se incluirán más líneas en el pool.

Método 3:

- Al igual que en el Método 2, para cada par OD, se calcularán los 10 caminos más cortos para ir desde el nodo origen hasta el nodo destino, en términos de tiempo, por lo que dependerá de la velocidad establecida (parámetro del modelo).
- De la misma forma que para el Método 2, cada línea del pool deberá contener como máximo 9 nodos, para evitar tener líneas con muchas paradas y deberá suponer un tiempo total de trayecto de 60 minutos como máximo.

Cada par OD de la red deberá estar contenido en al menos 1 línea del pool. Una vez alcanzado este valor para todos los pares OD, no se incluirán más líneas en el pool.

### 5.2.2 Parámetros establecidos para la resolución del problema TNDP

Para la resolución del modelo de planificación de líneas, se han establecido las siguientes consideraciones principales:

- Se ha considerado una velocidad media para los desplazamientos a pie de 2 km/h y en bus de 60 km/h.
- Se establece un límite de frecuencia de 20 autobuses/hora en cada línea y de 60 autobuses/hora para aquellos arcos por los que pasan varias líneas.
- Se considera que los usuarios podrán circular desde el nodo origen al nodo destino de cada par a través de un subgrafo formado por los arcos pertenecientes a los 10 caminos mínimos que unen el origen y destino de par.

Se busca obtener el diseño de una red de autobús formada por 4 líneas que minimice el tiempo de trayecto de los usuarios.

# 6 RESULTADOS

A partir de los casos de estudio definidos en el anterior apartado, se procede a obtener y analizar los resultados obtenidos en cada una de las pruebas realizadas. Para ello, se realizan distintas comparativas, analizando los principales aspectos que determinarán si la red diseñada finalmente es beneficiosa tanto para los usuarios como para la empresa operadora.

Para ello, en primer lugar, se analizarán las características de los pools de líneas candidatas obtenidos como resultado de los métodos estudiados, que determinarán el tamaño del modelo de optimización a resolver. Tras ello, se estudiarán los resultados del problema de planificación de líneas obtenidos a partir de los pools anteriores, es decir, qué líneas del pool inicial el modelo ha seleccionado como parte de la solución óptima de la red de autobús, así como su frecuencia, teniendo como objetivo principal minimizar el tiempo total de desplazamiento de los usuarios sobre la red.

## 6.1 Resultados de la generación de pools

Tal y como se ha explicado anteriormente, cada uno de los métodos de generación de líneas candidatas propuestos en este trabajo siguen criterios diferentes, tanto para la generación del total de líneas factibles como para la selección de las mejores para formar parte del pool inicial. De este modo, con cada método se obtendrá un pool de líneas candidatas, cuyas características y tamaño diferirán de los obtenidos a partir de otro método, afectando al tamaño del modelo de optimización para la resolución del modelo TNDP y a la calidad de la solución óptima obtenida.

Para comparar los pools obtenidos como resultado de cada uno de los métodos, se deben tener en cuenta los siguientes aspectos:

- En el Método 1, no se tiene en cuenta la demanda para la generación de líneas candidatas factibles entre cada par origen-destino de la red. Una vez se tienen todas las líneas factibles, se pueden seguir dos criterios de poda (PRUNEMETHOD 1 y PRUNEMETHOD 2), por lo que en cada caso se obtendrán pools diferentes.
- El Método 2 y Método 3 siguen los mismos criterios para la generación de líneas candidatas factibles, teniendo en cuenta para ello tanto el tiempo de trayecto como la demanda entre cada par origen-destino. En cuanto a la poda, en ambos métodos se comprueba que el pool represente adecuadamente los elementos que forman la red original, contando los nodos contenidos en las líneas en el caso del Método 2 y los pares OD en el Método 3.

Para analizar la idoneidad de cada uno de los métodos desarrollados, así como sus efectos en los resultados del problema de diseño de líneas, se han definido 100 escenarios, a partir de 10 redes con coordenadas aleatorias de 16 nodos y de 10 matrices de demanda entre cada par origen-destino.

En la siguiente gráfica, se muestra el tamaño del pool obtenido con cada método para un mismo escenario:

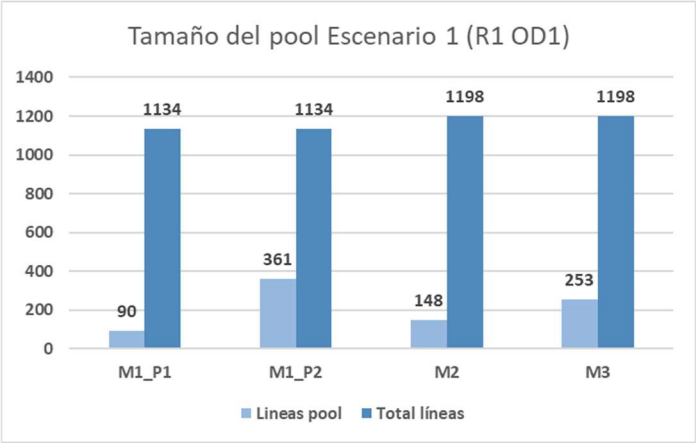


Figura 17. Tamaño de pool Escenario 1

Como se puede observar en dicha gráfica, el Método 1 genera un total de 1134 líneas factibles, un número menor que en los métodos 2 y 3, ya que en estos métodos se garantiza que todos los pares OD (o todos los nodos) están representados en el pool de líneas candidatas, mientras que en el método 1 sólo se procede a eliminar líneas candidatas incluidas en otras líneas candidatas.

En cuanto al Método 1, cabe destacar que, una vez generadas todas las líneas candidatas factibles, el criterio de poda P1 descarta la incorporación al pool de todas aquellas líneas cuyos nodos estén contenidos en otra línea más larga, sin considerar el orden, mientras que el criterio de poda P2 descarta incluir aquellas líneas idénticas a alguna sucesión de nodos incluidos en otra línea más larga, manteniendo el mismo orden. Por tanto, el P2 es más restrictivo a la hora de descartar líneas candidatas, por lo que genera pools de mayor tamaño que con el P1, tal y como se puede observar en el gráfico anterior.

Por otro lado, dado que el Método 3 asegura que todos los pares origen-destino estén incluidos en alguna de las líneas del pool, descarta un menor número de líneas candidatas que el Método 2, por lo que se obtienen pools de mayor tamaño, como muestra el ejemplo.

La matriz de demanda OD1 representa una situación de baja demanda. Con objeto de comparar la influencia de la demanda, a continuación, se muestra la gráfica anterior de tamaño del pool obtenida en el Escenario 10, en el cual se considera la misma red que en el Escenario 1 pero con una matriz OD de situación de alta demanda:

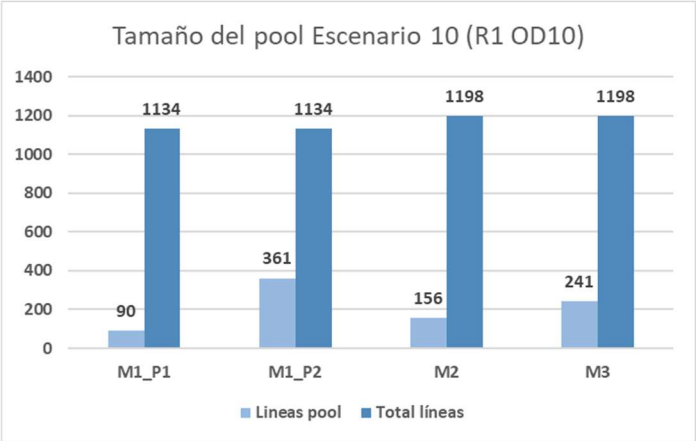


Figura 18. Tamaño de pool Escenario 10

Dado que el criterio seguido para generar las líneas candidatas únicamente depende de la distancia entre los nodos de la red y los criterios de poda del Método 1 no consideran la demanda para el descarte de líneas candidatas, en el Escenario 10, el Método 1 obtiene el mismo pool de líneas candidatas que en el Escenario 1, al tratarse de la misma red (R1).

Sin embargo, los Métodos 2 y 3 sí consideran la demanda para la selección de las líneas candidatas a incluir en el pool, por lo que varía el tamaño del mismo respecto al obtenido en el Escenario 1.

Con objeto de analizar los efectos de la red, se procede a comparar el tamaño de los pools anteriores con el

obtenido para el Escenario 100, el cual considera una red cuyos nodos tienen una localización diferente (R10) y la misma matriz de demanda que en el Escenario 10.

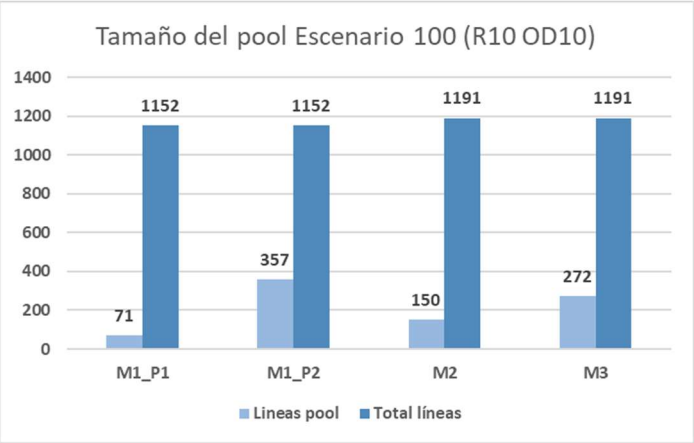


Figura 19. Tamaño de pool Escenario 100

En este caso, al tratarse de una red diferente a la de los ejemplos anteriores, la distancia entre nodos varía, afectando tanto al proceso de generación de líneas candidatas factibles como al descarte de las mismas para su incorporación al pool, en cualquiera de los métodos propuestos.

Para disponer de una visión más generalizada, la siguiente gráfica recoge el tamaño promedio de los pools obtenidos en todos los escenarios estudiados con cada uno de los métodos:

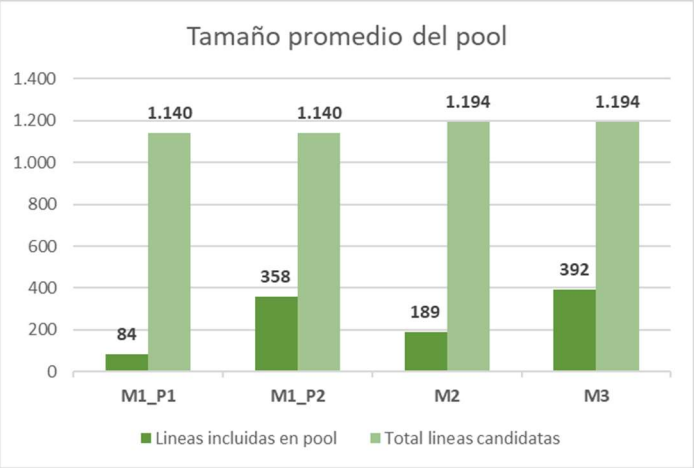


Figura 20. Tamaño promedio de pool

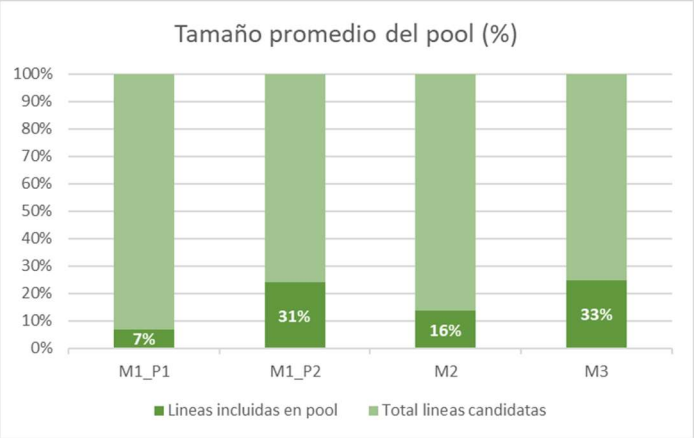


Figura 21. Tamaño promedio de pool en %

En la Figura 21, se puede observar que los métodos M1 con criterio de poda P1 y M2 son los que consiguen una

mayor reducción del tamaño del problema TNDP a resolver, ya que permiten obtener un pool inicial formado por menos del 20% del total de las líneas candidatas.

Sin embargo, es preciso comprobar que las líneas incluidas en el pool permitan obtener soluciones al problema de planificación de la red de autobús de buena calidad en cuanto a los resultados del modelo, de forma que la reducción del tamaño del problema sea de interés.

## 6.2 Resultados del TNDP a partir del pool inicial

Para evaluar la idoneidad de los métodos desarrollados, se procede a analizar los resultados obtenidos en el modelo de optimización para la resolución del TNDP, introduciendo como dato inicial el pool obtenido con cada uno de los métodos desarrollados. De este modo, el modelo seleccionará las mejores líneas del pool para alcanzar el objetivo establecido, que en este caso es el de minimizar el tiempo de trayecto de los usuarios para desaplazarse desde un nodo origen a un nodo destino a través de la red, considerando tanto desplazamientos directos como con trayectos a pie.

Se considera que un servicio de transporte público de calidad es aquel que permite el desplazamiento desde un punto a otro de la red del mayor número de personas de forma directa en el menor tiempo posible.

Con objeto de analizar si los resultados del modelo permiten obtener el diseño de una red de transporte de calidad a partir de los métodos desarrollados, se procedea estudiar los siguientes aspectos:

- Demanda directa satisfecha mediante la red de autobús.
- Tiempo promedio de viaje, considerando tiempo de trayecto de todos los desplazamientos, tanto en autobús de forma directa como incluyendo trayectos a pie, y la demanda total.

En primer lugar, se va a analizar la demanda satisfecha mediante la red de autobús. Dado que se han considerado 10 matrices de demanda diferentes para la definición de los escenarios, se procede a analizar la demanda directa que permite satisfacer cada una de las redes finalmente obtenidas tras resolver el modelo de planificación de líneas.

En el siguiente gráfico se muestran los resultados obtenidos en cuanto a la demanda directa que la red R1, obtenida a partir de la red base, permite capturar con respecto al total de la demanda, considerando todas las matrices OD definidas, a partir de cada uno de los métodos propuestos:

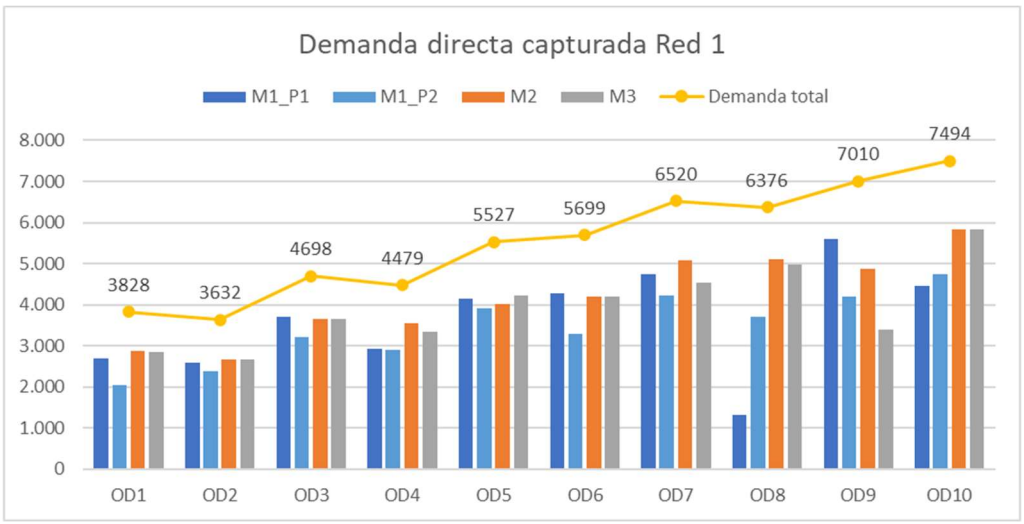


Figura 22. Demanda directa Red 1

En la Figura 22 se puede apreciar que para esta red los mejores resultados de demanda directa satisfecha se obtienen con los métodos M1_P1 y M2.

Calculando los valores promedio obtenidos para todas las redes, se obtienen los siguientes resultados:

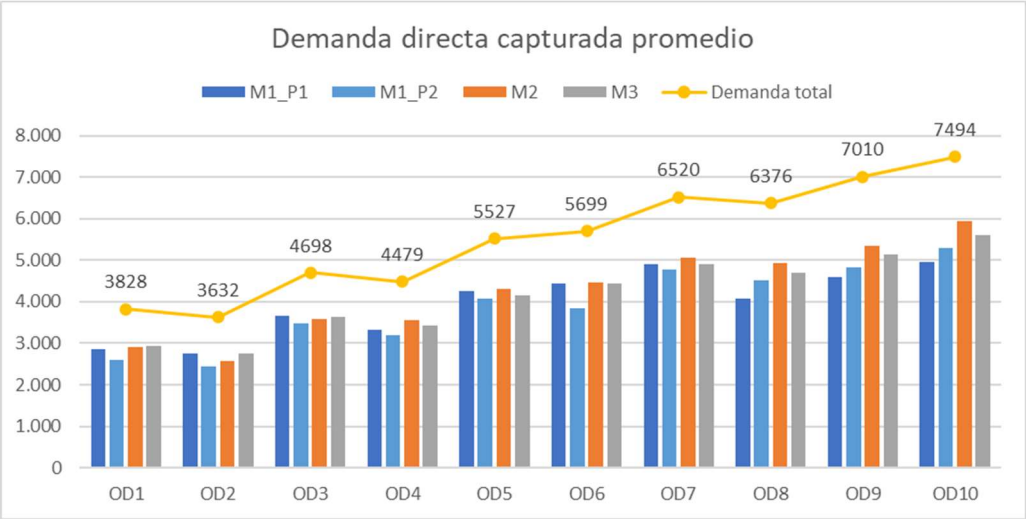


Figura 23. Demanda directa promedio por matriz OD

Analizando los valores promedio obtenidos para la demanda directa en los 100 escenarios estudiados, se obtienen los siguientes resultados, en porcentajes:

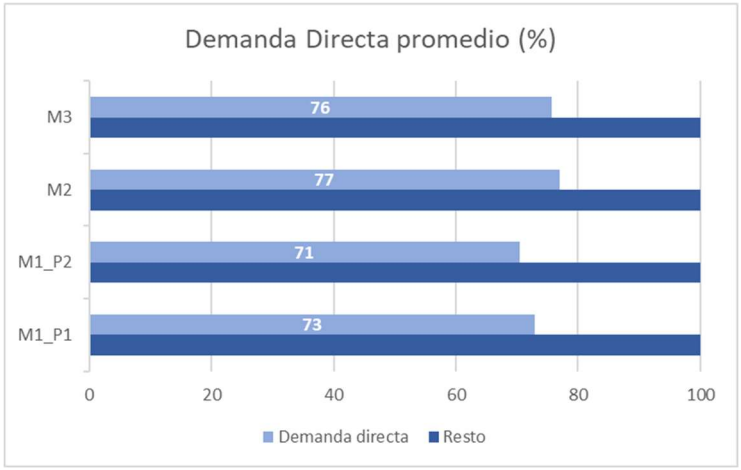


Figura 24. Demanda directa promedio

Finalmente, teniendo en cuenta los valores promedio, con el Método 2 de generación de pool inicial se obtienen los mejores resultados de demanda directa capturada por la red de líneas de autobús, seguido del Método 3.

Sin embargo, se debe analizar qué ocurre con el resto de la demanda que no se cubre de forma directa. Por ello, en el siguiente gráfico, se muestra el porcentaje promedio de la demanda directa satisfecha, teniendo en cuenta los resultados de todos los escenarios estudiados, así como la demanda indirecta satisfecha, que corresponde a aquellos desplazamientos a través de la red de bus que incluyen algún transbordo o trayecto a pie, y la demanda no cubierta, que representa el número de viajes para lo que los usuarios deberán realizar todo el trayecto a pie.

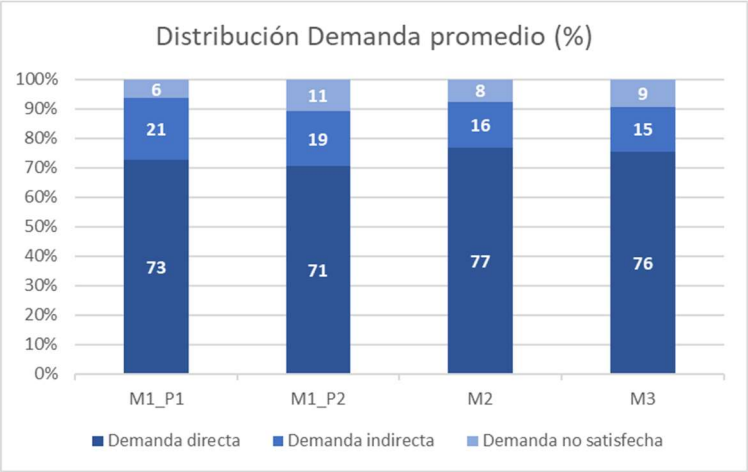


Figura 25. Distribución promedio de la demanda

Tal y como se puede observar en la Figura 24 y la Figura 25, el Método 1 es el que obtiene peores valores en cuanto a demanda directa que puede capturar la red de autobús resultante, lo cual se debe principalmente a que este método no tiene en cuenta la demanda para la selección de las líneas a incluir en el pool inicial, si no que prioriza incorporar las líneas más cortas, por lo que quizás descarta desde el primer momento líneas capaces de satisfacer de forma directa una gran demanda de pasajeros.

Sin embargo, este método (con el criterio de poda P1) es el que permite obtener soluciones al problema que proporcionan cobertura a una mayor proporción de la demanda, aunque más del 20% de los usuarios deba realizar al menos un transbordo.

Teniendo en cuenta que la realización de transbordos es un aspecto negativo desde el punto de vista del usuario, a priori el Método 2, seguido del Método 3, serán los más adecuados según los resultados de demanda, ya que permiten obtener soluciones que capturan un mayor porcentaje de demanda de forma directa, aunque un pequeño porcentaje de la demanda no pueda ser atendida mediante las redes de autobús obtenidas al resolver el modelo de planificación.

Una vez comparados los resultados obtenidos en cuanto a la demanda que permite satisfacer la red, se procede al análisis del tiempo promedio de viaje de las soluciones con cada uno de los métodos desarrollados.

Cabe destacar que el tiempo promedio de viaje incluye todos los desplazamientos realizados por los usuarios según la demanda total, sin distinguir entre los desplazamientos realizados a través de la red de autobús o por la red peatonal, ni si los desplazamientos en autobús se realizan de forma directa o incluyendo algún transbordo.

Dado que el tiempo de viaje depende de la longitud de la red y la distancia entre sus nodos, se van a comparar los resultados obtenidos en cuanto a tiempo de viaje según la red estudiada.

En la siguiente gráfica se muestra el tiempo promedio de viaje (en minutos) para la Red 1, considerando todas las matrices de demanda, obtenido como resultado del problema TNDP a partir de los pools obtenidos con los distintos métodos desarrollados:

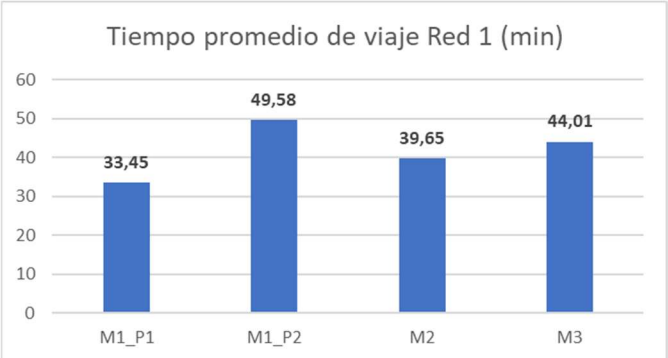


Figura 26. Tiempo promedio viaje Red 1

Tal y como se puede observar, con el Método 1 P1 se obtienen los menores tiempos de trayecto promedio para

esta red. Esto se debe principalmente a que este método prioriza que las líneas que se incluyen en el pool sean los trayectos más cortos entre cada par de nodos origen-destino, evitando que haya líneas contenidas en otras más largas. Sin embargo, los Métodos 2 y 3 priorizan que las líneas incluidas en el pool sean las que mayor demanda directa satisfacen, pudiendo descartar líneas más cortas que contengan nodos o pares OD que ya estén suficientemente representados en el pool en líneas de mayor demanda.

A continuación, se muestran los resultados para el resto de las redes estudiadas. Se puede apreciar que en todas las redes se mantienen las observaciones obtenidas para la Red 1:

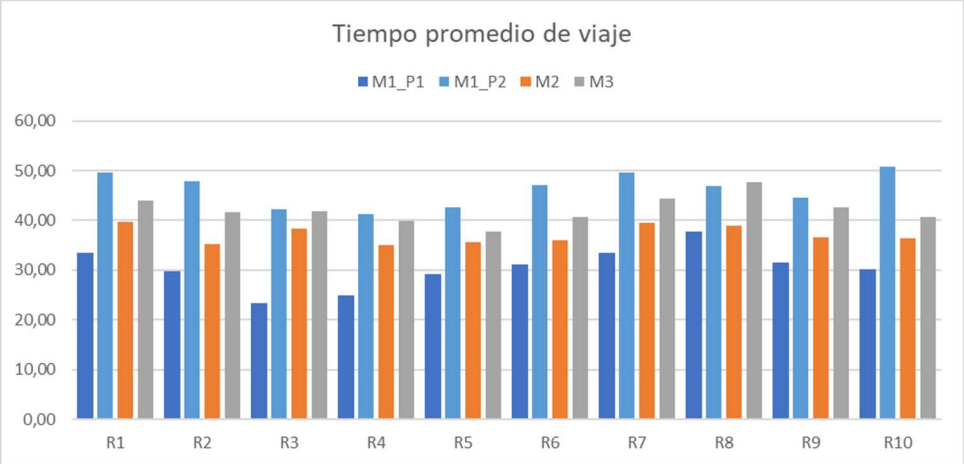


Figura 27. Tiempo promedio viaje por red

Para disponer de una visión general, en el siguiente gráfico se muestra el tiempo promedio de viaje considerando todos los escenarios estudiados, diferenciando los resultados obtenidos a partir de cada método:

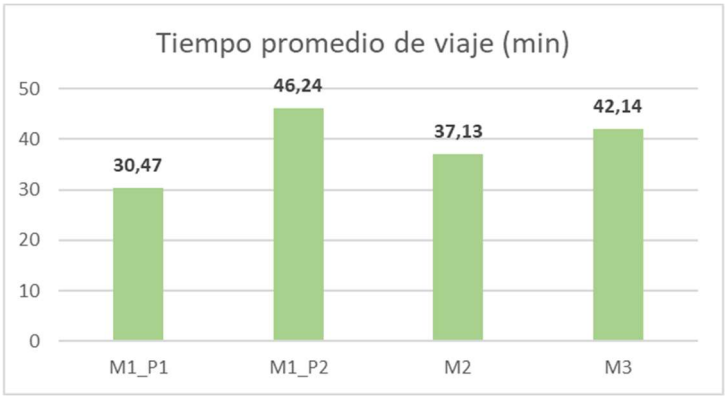


Figura 28. Tiempo promedio viaje

Como se puede observar en el anterior gráfico, los métodos que permiten obtener un menor tiempo promedio de viaje para todos los desplazamientos de los usuarios son el Método 1 P1 y el Método 2.

Cabe destacar que el tiempo promedio de viaje es inferior en líneas más cortas y directas, ya que e calcula como el ratio entre el tiempo de todos los trayectos y la demanda total, sin distinguir entre los que se realizan de forma directa o con transbordos. De este modo, con el Método 1 se obtiene un tiempo promedio de viaje muy inferior al resto, pero da servicio a una menor demanda de forma directa.

Como se indicaba anteriormente, se considera que un servicio de tranporte público de calidad es aquel que permite el desplazamiento de mayor número de pasajeros de forma directa en el menor tiempo posible. Por ello, en base a los resultados, destacan los obtenidos para el Método 2, ya que, aunque el tiempo promedio de viaje es más elevado que con el Método 1, permite dar cobertura a una mayor proporción de demanda de forma directa a través de la red de autobús.

No obstante, dado que inicialmente se establecieron determinados valores para los diferentes parámetros de cada método, no se tiene la certeza de que los valores establecidos sean los más adecuados. Por ello, a continuación, se van a realizar diferentes pruebas modificando algunos de estos parámetros, con objeto de analizar la influencia

de los mismos en los resultados obtenidos y si es posible obtener alguna mejora.

### 6.2.1 Prueba 1. Repetición de nodos en las líneas del pool

Los resultados mostrados para el Método 2 se han obtenido con la suposición inicial de que cada nodo de la red debe estar contenido en al menos 25 líneas del pool y que, una vez alcanzado este valor para todos los nodos, no se incluyan más líneas en el pool, tal y como se indicó en el capítulo 5 de la presente memoria. Con objeto de analizar la influencia de este parámetro y si este valor es el más favorable, se han obtenido los resultados del problema modificando dicho valor, probando con 10, 15, 20, 30 y 35 repeticiones de cada nodo en las líneas del pool.

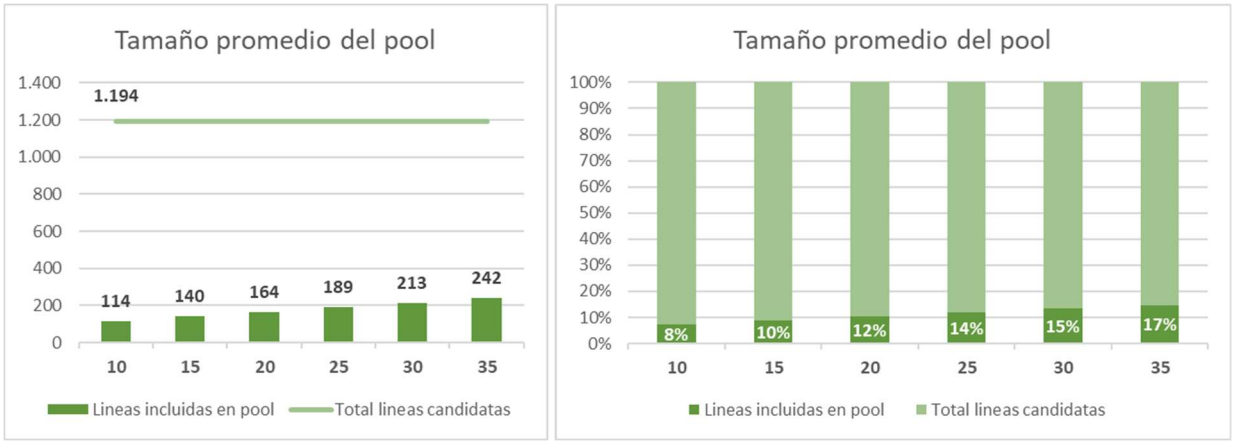


Figura 29. Tamaño promedio del pool, con distintos valores de repetición de nodos en sus líneas

Tal y como se puede observar, al incrementar el número de veces que debe aparecer cada nodo en las líneas del pool, aumenta el tamaño del mismo, ya que el método comenzará a descartar líneas una vez se alcance un valor mayor en el conteo de cada nodo.

Una vez ejecutado el modelo del TNDP, se obtienen los siguientes resultados promedio en cuanto a la demanda:

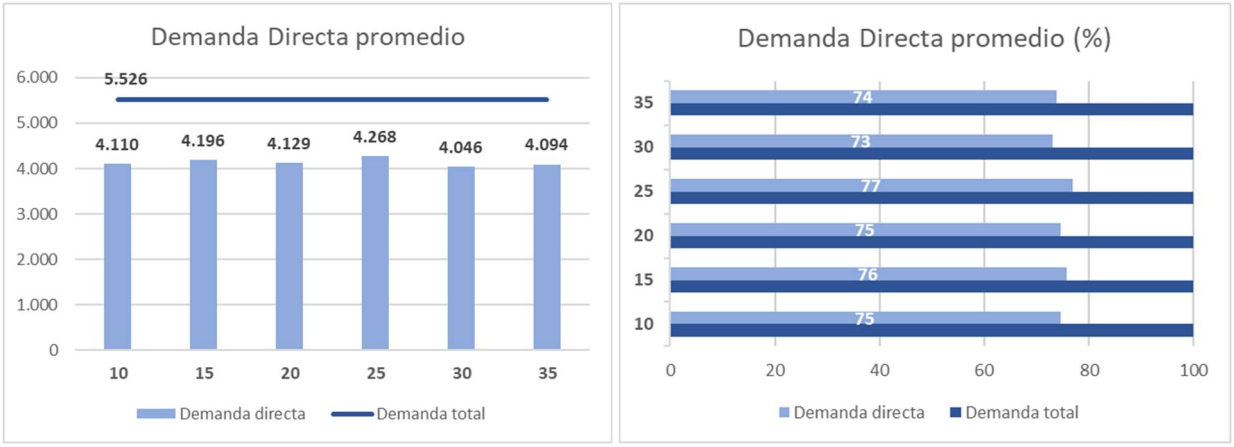


Figura 30. Demanda directa promedio, con distinta representación de cada nodo en el pool inicial

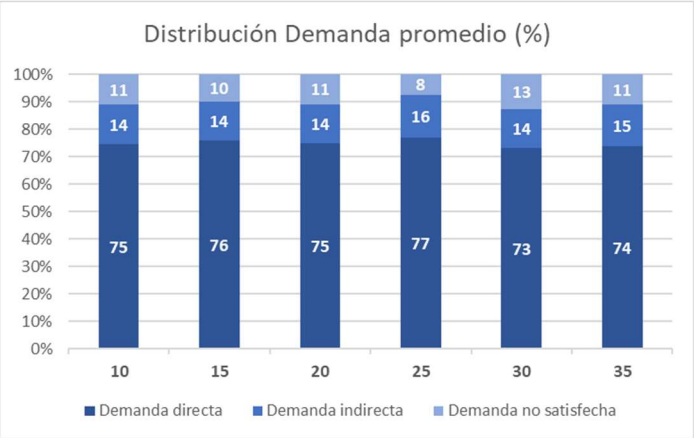


Figura 31. Distribución de la demanda, con distinta representación de cada nodo en el pool inicial

Aunque a priori lo lógico sería que un pool de mayor tamaño permitiera obtener mejores soluciones, las anteriores gráficas muestran que no existe tal relación, ya que para los escenarios estudiados se obtienen peores valores promedio de demanda directa en pools más grandes con una mayor repetición de cada nodo. Esto demuestra que la importancia no está en el tamaño del pool, si no en el proceso de selección de las líneas a incluir en el mismo, de forma que éstas sean las de mayor influencia en la obtención de buenos resultados.


Figura 32. Tiempo promedio de viaje, con distinta representación de cada nodo en el pool inicial

En cuanto al tiempo promedio de viaje, a partir de pools con una menor repetición de nodos (más pequeños) se obtienen menores valores para el tiempo promedio, ya que se incluyen aquellas líneas más cortas que satisfacen una mayor demanda, aumentando su longitud a medida que se exigen más repeticiones.

### 6.2.2 Prueba 2. Número de caminos posibles entre cada par OD

El modelo de optimización considerado en el presente trabajo para el problema de planificación de líneas incluye un parámetro que restrige el número de posibles caminos por los que se puede ir de un nodo a otro de la red a un valor de 0 caminos, ya sea realizando el trayecto a través de la red de autobús, de la red peatonal o utilizando ambas.

A pesar de que este parámetro no influye en la generación del pool inicial, se considera conveniente analizar su efecto en los resultados del modelo a partir del pool obtenido para el Método 2.

Para ello, se ha ejecutado el modelo a partir del pool obtenido en el Método 2 (con los parámetros iniciales), estableciendo un mayor número de caminos posibles entre cada par OD (20 caminos), con objeto de estudiar si esta restricción influye en la calidad de los resultados obtenidos.

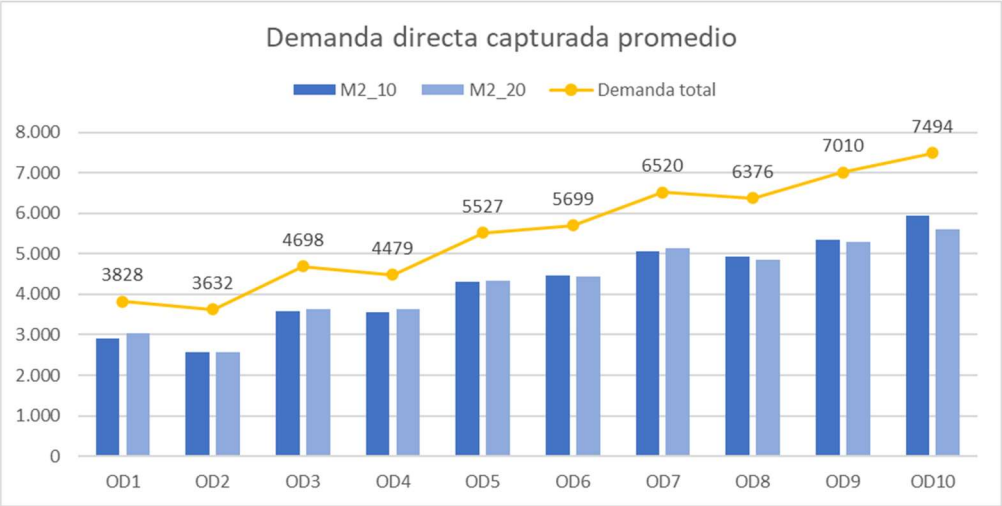


Figura 33. Demanda directa promedio por matriz OD, con diferente nº de caminos permitidos

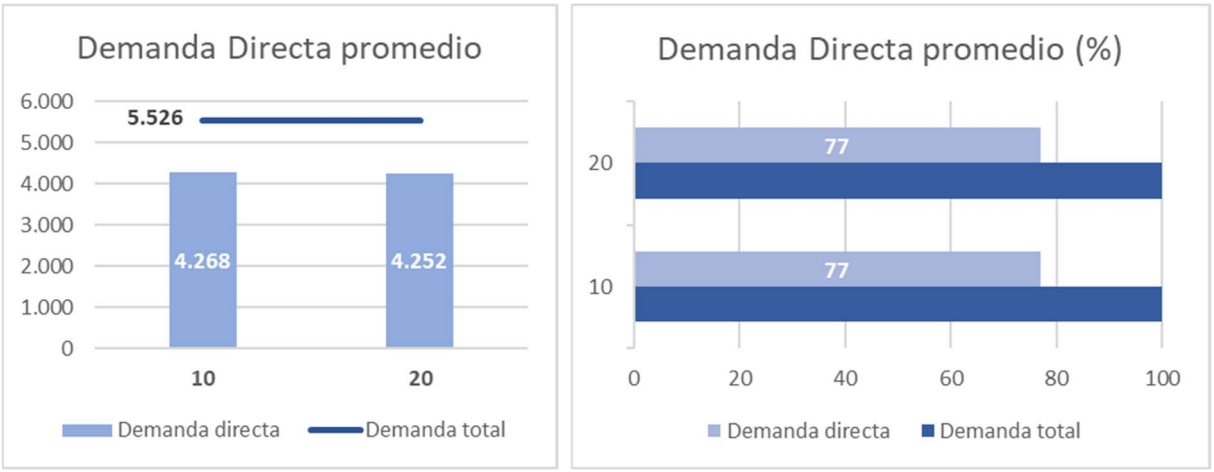


Figura 34. Demanda directa promedio, con diferente nº de caminos permitidos

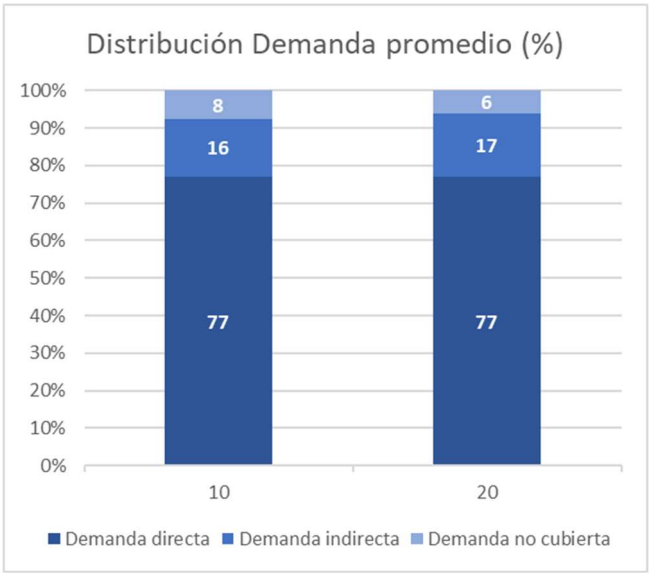


Figura 35. Distribución de la demanda promedio, con diferente nº de caminos permitidos

Los resultados obtenidos en cuanto a demanda directa son muy similares, por lo que muestran que no se obtienen mejores resultados al permitir más posibles caminos para el desplazamiento entre dos puntos de la red. Esto se debe a que el objetivo del modelo de optimización es minimizar el tiempo de trayecto, priorizando en todo

momento que el flujo de pasajeros sea capturado por los caminos más cortos. Sin embargo, ampliando el número de posibles caminos, se da cobertura a una mayor parte de la demanda de forma indirecta, por lo que se considera que da un mejor servicio.

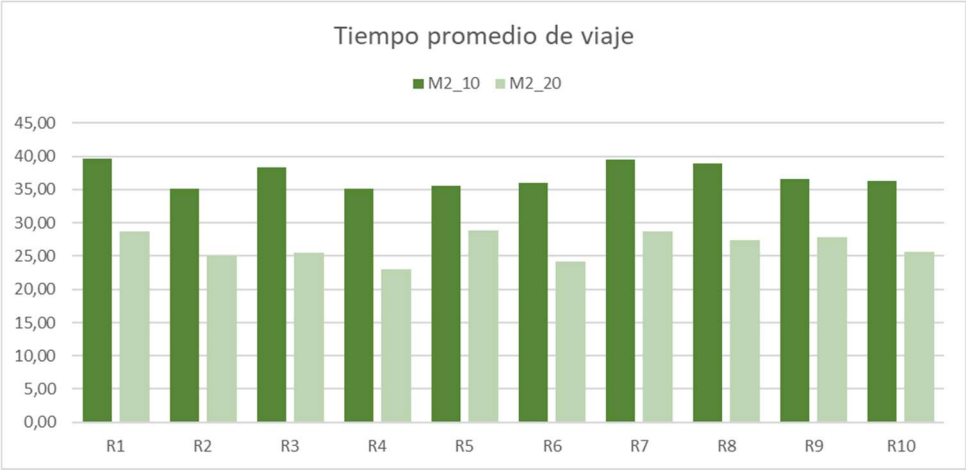


Figura 36. Tiempo promedio de viaje por red, con diferente nº de caminos permitidos

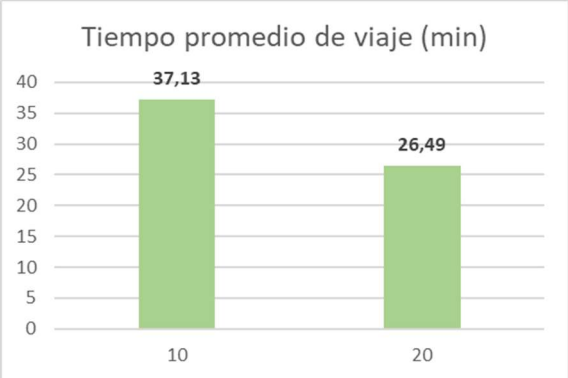


Figura 37. Tiempo promedio de viaje, con diferente nº de caminos permitidos

Por otro lado, si se consideran más posibles caminos para realizar el desplazamiento desde un nodo a otro de la red, se detecta que el modelo es capaz de encontrar mejores soluciones en cuanto a tiempo promedio de viaje. Esto se debe, por un lado, a que se está dando cobertura a una mayor demanda de forma indirecta, como se ha indicado anteriormente, y, por otro lado, a que el hecho de relajar esta restricción, puede llevar a recomendar trayectos con una geometría menos lógica. Esto quiere decir que, al ampliar los posibles caminos para ir desde un nodo a otro, quizás se puede reducir el tiempo total del trayecto si se va andando hasta un nodo más lejano por el que pase una línea de autobús que permita alcanzar el nodo deseado, por ejemplo.

En base a los resultados obtenidos en esta prueba, se considera que se podrían obtener soluciones que brinden un mejor servicio si se permite un mayor número de posibles caminos entre cada par de nodos de la red. Sin embargo, el aumento de este valor eleva considerablemente el coste computacional para la resolución del TNDP. Durante estas pruebas, se ha requerido el doble de tiempo de ejecución para obtener los resultados con valor 20, por lo que sería impensable su aplicación en un entorno real de mayor tamaño. Aunque se obtienen resultados algo mejores, se debe valorar si realmente compensa el coste computacional de dicha modificación.

### 6.2.3 Prueba 3. Número de nodos por línea

El Método 2 incluye un parámetro para limitar el número de nodos por los que pasa cada línea del pool, con objeto de descartar desde el principio aquellas líneas demasiado largas, las cuales a priori no resultan interesantes para el usuario ni para el operador del servicio. En el método original, se consideró que serían factibles aquellas líneas que pasaran por un máximo de 9 nodos, por lo que se ha probado a modificar este parámetro a 7 y 11 nodos.

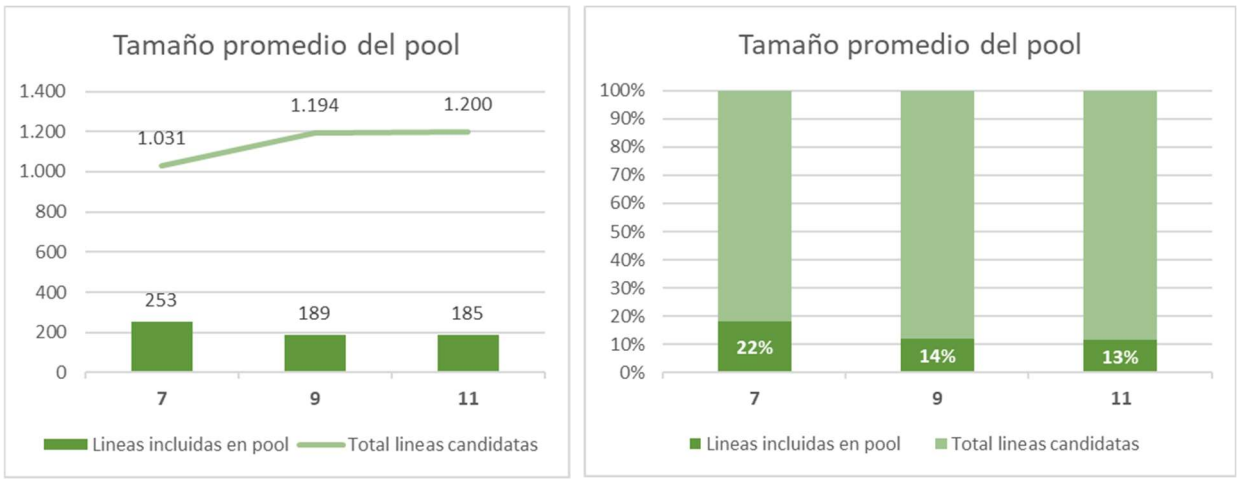


Figura 38. Tamaño promedio del pool, con distinto nº máximo de nodos por línea

En las anteriores gráficas se puede observar que, al limitar a que las líneas deban contener 7 nodos como máximo, se elimina casi un 20% del total de líneas candidatas desde la fase inicial de generación de líneas, sin llegar a la fase de poda, en la que se estudia la demanda directa a la que podría dar servicio.

Por otro lado, no se detectan muchas diferencias en el pool al aumentar dicho límite a 11 nodos por línea. Esto se debe a que la gran mayoría de las líneas candidatas, de forma natural, no pasan por más de 9 nodos, ya que superarían la restricción de máxima duración total de la línea establecida.

A partir de los diferentes pools obtenidos, a continuación, se muestran los resultados obtenidos en el modelo TNDP en cuanto a demanda y tiempo promedio de viaje:

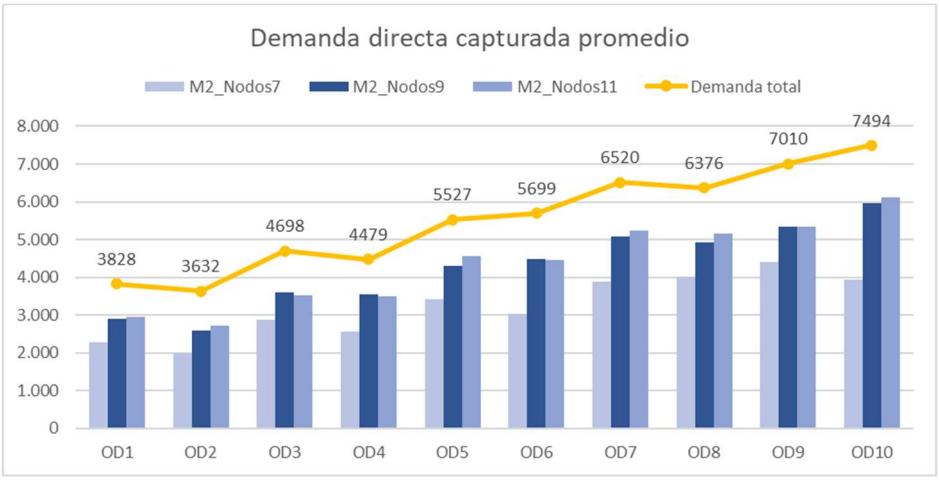


Figura 39. Demanda directa promedio por matriz OD, con diferente nº de nodos por línea

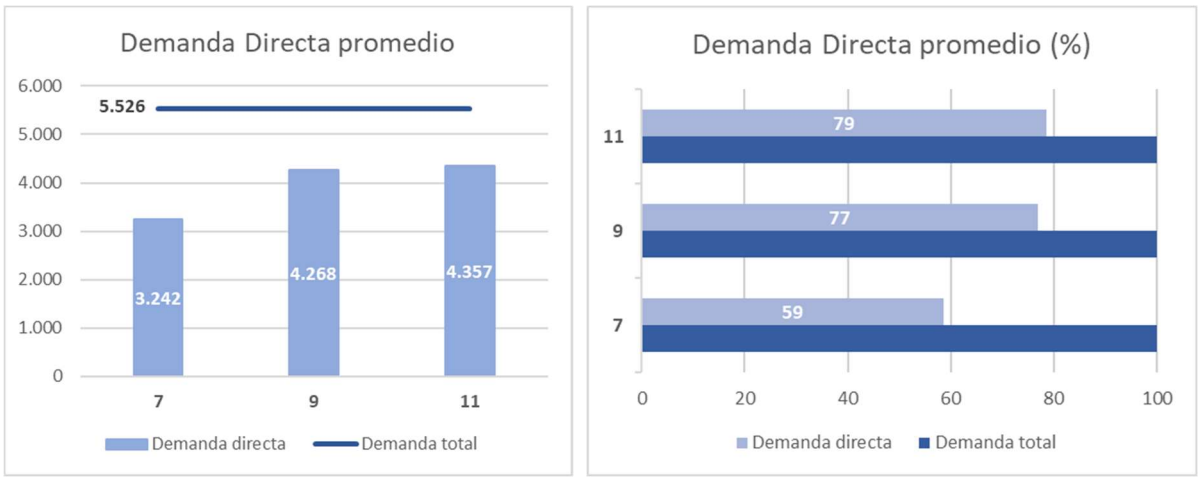


Figura 40. Demanda directa promedio, con diferente nº de nodos por línea

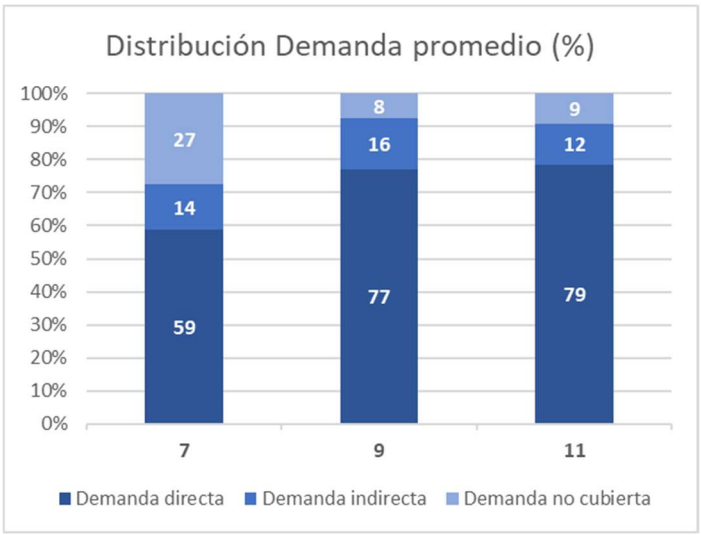


Figura 41. Distribución de la demanda promedio, con diferente nº de nodos por línea

En las anteriores gráficas se puede observar que, al permitir que se generen líneas más largas, se pueden obtener soluciones al problema de planificación de líneas que proporcionen un mejor servicio en cuanto a demanda directa. Además, se vuelve a mostrar que un pool de líneas candidatas de mayor tamaño no tiene por qué proporcionar mejores resultados, si no que el éxito reside en la selección de las líneas que tengan un mayor potencial.

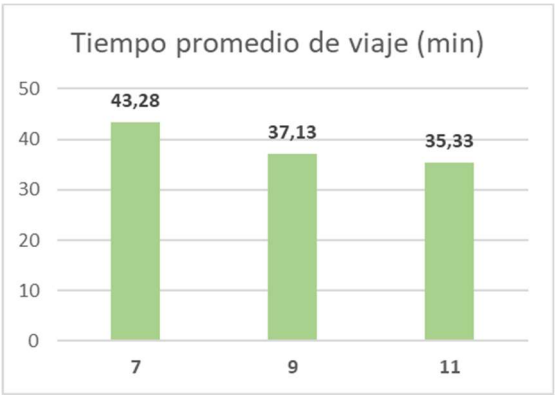


Figura 42. Tiempo promedio de viaje, con diferente nº de nodos por línea

Dado que las líneas que pasan por más nodos pueden capturar una mayor proporción de demanda directa, también permiten obtener mejores resultados en cuanto al tiempo promedio de viaje.

# 7 CONCLUSIONES

En el presente Trabajo Fin de Máster se ha llevado a cabo el desarrollo de diferentes métodos para la generación de un conjunto de líneas candidatas (pool) en una etapa previa al planteamiento y resolución de un modelo de optimización para el diseño de una red de transporte público. El principal objetivo perseguido con la realización de este trabajo es el de reducir el tamaño del problema de planificación de líneas que posteriormente se debe resolver mediante un modelo de optimización. La utilización de un conjunto de líneas candidatas de gran tamaño supone un importante inconveniente si se desea aplicar el procedimiento de optimización a redes reales. Por otra parte, se desea garantizar la obtención de buenas soluciones para el problema de diseño de una red de transporte público, por lo que la utilización de un pool de pequeño tamaño puede ser negativo, ya que el modelo de optimización debe seleccionar algunas de las líneas de entre el conjunto de líneas candidatas, y si este contiene pocas líneas es habitual que la solución final sea de muy mala calidad en términos de la demanda directa capturada y de los tiempos de viaje sobre la red. Por tal motivo, es necesario que el pool tenga un tamaño razonable y que además esté compuesto de líneas candidatas de calidad. La consecución de ambos objetivos no es sencilla, sobre todo porque la bondad de las líneas candidatas sólo puede medirse tras resolver el modelo de optimización.

Se han desarrollado tres métodos para construir el pool de líneas candidatas, los cuales se diferencian entre sí por los criterios seguidos durante la generación y selección de las líneas candidatas más interesantes a incluir en el pool. Para valorar la idoneidad de dichos métodos, se han diseñado distintos escenarios de prueba, combinando distintas redes de 16 nodos y diferentes matrices de demanda entre cada par origen destino de la misma, con el objetivo de ejecutar el modelo de optimización de planificación de líneas (TNDP) utilizando como datos de partida el pool obtenido con cada método, y realizar posteriormente un estudio comparativo de los resultados obtenidos con cada uno de ellos.

En base a los resultados globales obtenidos para todos los escenarios, se ha llegado a la conclusión de que el Método 2 es el que mejores resultados ha proporcionado para la batería de experimentos realizada, ya que ha permitido dar cobertura a aproximadamente un 77% de la demanda total de forma directa y a un 17% más si se incluyen trayectos a pie, alcanzando un valor de promedio de tiempo de trayecto de unos 37 minutos, considerando todos los trayectos, ya sean a través de la red de autobús, de la red peatonal o de ambas.

Dado que durante la definición de los métodos se establecieron valores fijos para determinados parámetros, se consideró conveniente realizar varias pruebas modificando algunos de dichos parámetros, con la finalidad de analizar la influencia de los mismos sobre los resultados obtenidos para el Método 2. En estas pruebas, se pudo comprobar que el hecho de aumentar el tamaño del pool no conlleva a un incremento de la calidad de las soluciones encontradas por el modelo, pero sí aumenta considerablemente el coste computacional de la resolución del problema TNDP. Esto demuestra que la clave para alcanzar el objetivo del trabajo es la correcta priorización y selección de las líneas que lo forman.

No obstante, a día de hoy, sigue sin existir un método claramente dominante para la generación de un pool inicial de líneas candidatas, ya que todos los métodos estudiados presentan ventajas e inconvenientes, en función de su ámbito de aplicación. Además, son numerosos los factores que influyen en el planteamiento y resolución del problema de planificación de líneas, lo que dificulta ampliamente la definición de un método de generación de

líneas candidatas previo que garantice la calidad de los resultados, dada la gran cantidad de diferentes posibilidades y combinaciones que se deberían estudiar. Por tanto, se sigue considerando un problema de difícil resolución para el que no hay una solución única, aunque sí se ha podido demostrar que el pool tiene una influencia directa en los resultados del problema de diseño de la red de tránsito.

# REFERENCIAS

Baaj, M. H., & Mahmassani, H. (1991). An AI-based approach for transit route system planning and design. *Journal of Advanced Transportation*, *25*(2), 187-210.

Baaj, M. H., & Mahmassani, H. S. (1995). Hybrid route generation heuristic algorithm for the design of transit networks. *Transportation Research Part C*, *3*(1), 31-50. https://doi.org/10.1016/0968-090X(94)00011-S

Bussieck, M. R., Kreuzer, P., & Zimmermann, U. T. (1997). Optimal lines for railway systems. *European Journal of Operation Research*, *96*, 54-63.

Bussieck, M. R., Lindner, T., & Lubbecke, M. E. (2004). A fast algorithm for near cost optimal line plans. *Mathematical Methods of Operations Research*, *59*, 205-220.

Canca, D., De-Los-Santos, A., Laporte, F., & Mesa, J. A. (2017). An adaptive neighborhood search metaheuristic for the integrated railway rapid transit network design and line planning problem. *Computers & Operations Research*, *78*, 1-14.

Canca, D., De-Los-Santos, A., Laporte, G., & Mesa, J. A. (2019). Integrated railway rapid transit network design and line-planning problem with maximum profit. *Transportation Research E: Logistics and Transportation Review*, *127*, 1-30.

Canca, D., De-Los-Santos, A., Zarzo, A., & Villa, A. (2022). *A multilayer line planning model for the design/expansión of public transportation networks*. EWGLA XXVII, Aveiro, Portugal.

Ceder, A. (1984). Bus frequency determination using passenger count data. *Transportation Research Part A: General*, *18*(5–6), 439-453.

Ceder, A., & Wilson, N. H. M. (1986). Bus network design. *Transportation Research Part B: Methodological*, *20*(4), 331-344.

Chakroborty, P., Deb, K., & Srinivas, B. (1998). Network-wide optimal scheduling of transit systems using genetic algorithms. *Computer-Aided Civil and Infrastructure Engineering*, *13*(5), 363-376.

Chakroborty, P., Deb, K., & Subrahmanyam, P. S. (1995). Optimal scheduling of urban transit system using genetic algorithms. *ASCE Journal of Transportation Engineering*, *121*(6), 544-553.

Chakroborty, P., & Wivedi, T. (2002). Optimal route network design for transit systems using genetic algorithms. *Engineering Optimization*, *34*(1), 83-100.

Cipriani, E., Gori, S., & Petrelli, M. (2012). A bus network design procedure with elastic demand for large urban areas. *Public Transport*, *4*(1), 57-76. https://doi.org/10.1007/s12469-012-0051-7

Feng, X., Zhu, X., Qian, X., Jie, Y., Ma, F., & Niu, X. (2018). A new transit network design study in consideration of transfer time composition. *Transportation Research Part D: Transport and Environment*, *66*, 85-94. https://doi.org/10.1016/j.trd.2018.03.019

Furth, P. G., & Wilson, N. H. M. (1981). Setting frequencies on bus routes: Theory and practice. *Transportation Research Record*, *818*, 1-7.

Gattermann, P., Harbering, J., & Schöbel, A. (2017). Line pool generation. *Public Transport*, *9*(1-2), 7-32. https://doi.org/10.1007/s12469-016-0127-x

Goossens, J.-W., van Hoesel, S., & Kroon, L. (2006). On solving multi-type railway line planning problems. *European Journal of Operational Research*, *168*(2), 403-424.

Guan, J. F., Yang, H., & Wirasinghe, S. C. (2006). Simultaneous optimization of transit line configuration and passenger line assignment. *Transportation Research Part B: Methodological*, *40*(10), 885-902.

Guihaire, V., & Hao, J.-K. (2008). Transit network design and scheduling: A global review. *Transportation Research Part A: Policy and Practice*, *42*(10), 1251-1273.

Jiang, Y., Szeto, W., & Ng, T. (2013). Transit network design: A hybrid enhanced artificial bee colony approach and a case study. *International Journal of Transportation Science and Technology*, *2*(3), 243-260. https://doi.org/10.1260/2046-0430.2.3.243

Lampkin, W., & Saalmans, P. D. (1967). The design of routes, service frequencies, and schedules for a municipal bus undertaking: A case study. *OR*, *18*(4), 375-397.

Mandl, C. E. (1980). Evaluation and optimization of urban public transportation networks. *European Journal of*

*Operational Research*, *5*(6), 396-404.

Mauttone, A., & Urquhart, M. E. (2009). A route set construction algorithm for the transit network design problem. *Computers and Operations Research*, *36*(8), 2440-2449. https://doi.org/10.1016/j.cor.2008.09.014

Shih, M.-C., & Mahmassani, H. S. (1994). *A design methodology for bus transit networks with coordinated operations* (SWUTC/94/60016-1). Center for Transportation Research. University of Texas at Austin.

Szeto, W. Y., & Jiang, Y. (2014). Transit route and frequency design: Bi-level modeling and hybrid artificial bee colony algorithm approach. *Transportation Research Part B: Methodological*, *67*, 235-263.

Szeto, W. Y., & Wu, Y. (2011). A simultaneous bus route design and frequency setting problem for Tin Shui Wai, Hong Kong. *European Journal of Operational Research*, *209*(2), 141-155.

# ANEXO. CÓDIGO

En este anexo, se muestra el código elaborado en Python para cada uno de los métodos de generación de líneas candidatas propuestos en el presente trabajo, los cuales se han evaluado posteriormente mediante la ejecución de un modelo de optimización para la resolución del problema de planificación de líneas, utilizando para ello el optimizador Gurobi.

Cabe destacar que los Métodos 2 y 3 se han desarrollado con programación orientada a objetos (POO), con el objetivo de reducir su extensión.

## 7.1 Código Método 1

```
from typing import Set, Any, Tuple

import matplotlib.pyplot as plt
import matplotlib.image as im
import networkx as nx
import numpy
from IOfunctionsExcel5 import *
from MyPlotandNetworkFunctions5 import *
from gurobipy import *
from heapq import heappush, heappop
from itertools import count
import json as js
import io
import ast
import random
import gurobipy
from random import random


# Funciones para codificar y descodificar la red de bus

SumaCode1 = 10
SumaCode2 = 10


def code(Capa, Linea, Nodo):
    return (SumaCode1 * SumaCode2 * Capa + SumaCode2 * Linea + Nodo)


def decode(Code):
    Resto = Code % (SumaCode1 * SumaCode2)
    Capa = (Code - Resto) / (SumaCode1 * SumaCode2)
    Resto2 = Resto % SumaCode2
    Linea = (Resto - Resto2) / SumaCode2
    Nodo = Resto2
    return int(Capa), int(Linea), int(Nodo)
```

```
def decodeCapa(Code):
    a, b, c = decode(Code)
    return int(a)


def decodeLinea(Code):
    a, b, c = decode(Code)
    return int(b)
def decodeNodo(Code):
    a, b, c = decode(Code)
    return int(c)

#
*************************************************************************
*******
#
*************************************************************************
*******
#                           Definición de escenarios de trabajo
#                           Definición de parámetros
#
*************************************************************************
*******
#
*************************************************************************
*******

DEBUG=0  # 1 para que imprima resultados intermedios, a 0 para que no imprima

print(">>> Lectura de datos de redes")
# por ahora la red peatonal coincide con la red potencial para bus, aunque se
pueden usar diferentes

name1 = "Redes.xlsx"
name2 = "Redes.xlsx"
# Descripción de dónde se encuentran los datos de la red base peatonal
exp1 = {
    'Nodos': ['B2', 'B17'],
    'Coordenadas': ['B1', 'D17'],
    'Arcos': ['I2', 'I25'],
    'DArcos': ['I1', 'L25']
}

# Descripción de dónde se encuentran los datos de la red base para bus
exp2 = {
    'Nodos': ['B2', 'B17'],
    'Coordenadas': ['B1', 'D17'],
    'Arcos': ['I2', 'I25'],
    'DArcos': ['I1', 'L25']
}

# ******    PARAMETROS  **************

DISTANCIAMIN = 400  # Distancia mínima entre nodos para que sea factible el
par OD
VMedia = 60 * (1000 / 60)  # (m/min) Se considera que bus avanza a 60 km/h
VmediaPea = 2 * (1000 / 60)  # (m/min) Se considera que peatón avanza a 2km/h

FMAX = 20  # autobus/hora (1 autobus cada 2 mins)
FMAX2 = 60  # Frecuencia máxima autobuses/hora para una distintas lineas en
```

```
cada arco (1 autobus cada 2 mins)

Length_min = 3000 # longitud mínima de la línea para que pertenezca al pool
Length_max = 65000  # longitud máxima de la línea para que pertenezca al pool

PRUNEMETHOD = 1
# 1: elimina líneas contenidas en otras que tengan subconjuntos de nodos
repetidos
# 2: elimina líneas contenidas en otras que tengan subconjuntos de nodos
ORDENADOS repetidos
RANDOMIZE = 0 # 1 si se incluye aleatoriedad
SURVIVALPERCEN = 0.7 # si RANDOMIZE = 1, % líneas factibles sin poda

ALTERNATIVEROUTELINES = 10  # k caminos calculados para cada par OD
MAX_LINES_4 = 4  # Se usa en el modelo para restricción de máximo número de
líneas a implantar en el TNDP
NUMEROCAMINOS = 10  #subconjunto de nodos y arcos del grafo por los que se
permite que circulen los pasajeros desde un nodo a otro (zona de la red)

STOP_TIME = 400

#
*************************************************************************
*******
#
*************************************************************************
*******
#                           Lectura de datos de los escenarios
#                           (Combinaciones de 10 redes y 10 Matrices OD)
#
*************************************************************************
*******
#
*************************************************************************
*******
print(">>> Carga de datos de escenarios")

nameMat = "MATOD.xlsx"

EscenarioRed=[i for i in range(10)]
EscenarioMat=[i for i in range(10)]

HojasRed=['RED1', 'RED2', 'RED3', 'RED4','RED5', 'RED6','RED7','RED8','RED9',
'RED10']
HojasMatriz=['OD1', 'OD2', 'OD3', 'OD4','OD5', 'OD6','OD7', 'OD8','OD9',
'OD10']

METODO='M1'

for esc in EscenarioRed:
    sheet1 = HojasRed[esc]
    sheet2 = HojasRed[esc]
    for mat in EscenarioMat:
        FileLinePool = ''
        sheetMat = HojasMatriz[mat]
        DD_MatOD = Read_Excel_to_NesteDic(nameMat, sheetMat, 'D8', 'T24')
        FileLinePool= FileLinePool+ 'Pool_' + METODO + '_' + HojasRed[esc] +
'_' + HojasMatriz[mat] + '.txt'

        ################ Lectura de la red peatonal ####################
        # Lectura de nodos peatonal
        L_Nodos1 = Read_Excel_to_List(name1, sheet1, exp1['Nodos'][0],
```

```
exp1['Nodos'][1])
        # Leer coordenadas x e y de la red de nodos peatonal
        D_Coordenadas1 = Read_Excel_to_NesteDic(name1, sheet1,
exp1['Coordenadas'][0], exp1['Coordenadas'][1])
        # Leemos el arco, el origen, destinto y factor
        D_Arcos1 = Read_Excel_to_NesteDic_TupleInt(name1, sheet1,
exp1['DArcos'][0], exp1['DArcos'][1])

        ################ Lectura de la red base BUS ####################
        # Lectura de nodos bus
        L_Nodos2 = Read_Excel_to_List(name2, sheet2, exp2['Nodos'][0],
exp2['Nodos'][1])
        # Leer coordenadas x e y de la red de soporte para bus
        D_Coordenadas2 = Read_Excel_to_NesteDic(name2, sheet2,
exp2['Coordenadas'][0], exp2['Coordenadas'][1])
        # Leemos el arco, el origen, destinto y factor
        D_Arcos2 = Read_Excel_to_NesteDic_TupleInt(name2, sheet2,
exp2['DArcos'][0], exp2['DArcos'][1])

        #
*************************************************************************
*******
        #
*************************************************************************
*******
        #               Estructuras de datos para guardar escenarios.
        #               Duplicar arcos. Cálculo de longitudes de arcos.
Conversion a unidades adecuadas aplicando factor de escala a cada red (No
confundir con el escalado de la matriz de demanda) aquí se escala el grafo a
unidades reales.
        #
        #
*************************************************************************
*******
        #
*************************************************************************
*******
        print(">>> Duplicando arcos y escalando la red", FileLinePool)

        D_ArcosPeat={}
        for (i,j) in D_Arcos1:
            D_ArcosPeat[(i,j)]=D_Arcos1[(i,j)]
            D_ArcosPeat[(j,i)]={}
        # Duplica arcos (doble sentido)
        for (i,j) in D_Arcos1:
            D_ArcosPeat[(j,i)]['ORIGEN']=D_Arcos1[(i,j)]['DESTINO']
            D_ArcosPeat[(j, i)]['DESTINO']=D_Arcos1[(i,j)]['ORIGEN']
            D_ArcosPeat[(j,i)]['FACTOR']=D_Arcos1[(i,j)]['FACTOR']

        D_ArcosBus={}
        for (i,j) in D_Arcos2:
            D_ArcosBus[(i,j)]=D_Arcos2[(i,j)]
            D_ArcosBus[(j,i)]={}
        # Duplica arcos (doble sentido)
        for (i,j) in D_Arcos2:
            D_ArcosBus[(j,i)]['ORIGEN']=D_Arcos2[(i,j)]['DESTINO']
            D_ArcosBus[(j, i)]['DESTINO']=D_Arcos2[(i,j)]['ORIGEN']
            D_ArcosBus[(j,i)]['FACTOR']=D_Arcos2[(i,j)]['FACTOR']

        D_LongitudPeat = {}
        for i in D_ArcosPeat:
            a = D_ArcosPeat[i]['ORIGEN']
```

```
            b = D_ArcosPeat[i]['DESTINO']
            c = D_ArcosPeat[i]['FACTOR']
            D_LongitudPeat[(a, b)] = numpy.sqrt((D_Coordenadas1[a]['X'] -
D_Coordenadas1[b]['X']) ** 2 + (D_Coordenadas1[a]['Y'] -
D_Coordenadas1[b]['Y']) ** 2) * c

        D_LongitudBus={}
        for i in D_ArcosBus:
            a = D_ArcosBus[i]['ORIGEN']
            b = D_ArcosBus[i]['DESTINO']
            c = D_ArcosBus[i]['FACTOR']
            D_LongitudBus[(a, b)] = numpy.sqrt((D_Coordenadas2[a]['X'] -
D_Coordenadas2[b]['X']) ** 2 + (D_Coordenadas2[a]['Y'] -
D_Coordenadas2[b]['Y']) ** 2) * c

        ###############  Aplicando factor de escala a la red peatonal
*********************************
        numerador=4000  # Se supone que la red tiene una superficie de 4x4 Km
        denominador=1500    # Ancho de la red definida (1 nodo cada 500m)
        factorescala1 = numerador / denominador
        print("---------- Factor Escala red peatonal ->", factorescala1)

        for i in D_ArcosPeat:
            a = D_ArcosPeat[i]['ORIGEN']
            b = D_ArcosPeat[i]['DESTINO']
            c = D_ArcosPeat[i]['FACTOR']
            D_LongitudPeat[(a, b)] = D_LongitudPeat[(a, b)] * c *
factorescala1

        ###############  Aplicando factor de escala a la red BUS
************************************
        factorescala2=factorescala1
        print("---------- Factor Escala red Bus ->", factorescala2)

        for i in D_ArcosBus:
            a = D_ArcosBus[i]['ORIGEN']
            b = D_ArcosBus[i]['DESTINO']
            c = D_ArcosBus[i]['FACTOR']
            D_LongitudBus[(a, b)] = D_LongitudBus[(a, b)] * c * factorescala2

        #
*************************************************************************
*******
        #
*************************************************************************
*******
        # Escalado de la matriz de demanda y puesta a cero de los elementos
de la matriz tales que la distancia entre ellos es menor que  DISTANCIAMIN
*************************************************************************
*******
        #
*************************************************************************
*******
        print(">>> Escalado y lectura de matrices de demanda que cumplen
distancia mínima")

        for i in L_Nodos1:
            for j in L_Nodos1:
                DD_MatOD[i][j] = DD_MatOD[i][j]*Escala
        D_Dist = {}
        for i in L_Nodos1:
            for j in L_Nodos1:
```

```
                D_Dist[i, j] = numpy.sqrt((D_Coordenadas1[i]['X'] -
D_Coordenadas1[j]['X']) ** 2 + (D_Coordenadas1[i]['Y'] -
D_Coordenadas1[j]['Y']) ** 2) * factorescala1
                if D_Dist[i, j] < DISTANCIAMIN:
                    DD_MatOD[i][j] = 0.0 # Poner a 0 la demanda entre nodos
muy cercanos, ya que se considera que los usuarios irán a pie

        SumMatrix = {}
        for i in L_Nodos1:
            SumMatrix[i] = sum(DD_MatOD[i][j] for j in L_Nodos1)
        Sumatotal = sum(SumMatrix[i] for i in L_Nodos1)
        SumMatrix.clear()
        print("---------- Demanda total Matriz", sheetMat, "->", Sumatotal)
        D_Coordenadas1.clear()
        #
*************************************************************************
*******
        #
*************************************************************************
*******
        #               Calculando Pool de Líneas candidatas (POOL ENHANCED)
        #
*************************************************************************
*******
        #
*************************************************************************
*******
        # PRIMERO se crean los grafos de la red peatonal y de bus, a partir
de listas de tuplas (origen, destino, tiempo trayecto)
        # SEGUNDO se crea el pool, eliminando los arcos que no se encuentran
entre longmin y longmax
        # TERCERO se poda, eligiendo el método PRUNEMETHOD deseado (=1 o 2)

        print(">>> Construyendo Pool de Líneas candidatas")

        # PRIMERO
        # Conjunto de tuplas de arcos con tiempo de trayecto (según longitud
y velocidad) para grafo networkx G1 de la red peatonal
        L_ArcosPeat = []
        for i in D_ArcosPeat:
            a = D_ArcosPeat[i]['ORIGEN']
            b = D_ArcosPeat[i]['DESTINO']
            c = D_ArcosPeat[i]['FACTOR']
            aux = []
            aux.append(a)
            aux.append(b)
            aux.append(D_LongitudPeat[(a, b)]/VmediaPea)   # se incluye en la
tupla el TIEMPO (minutos) de recorrido a pie de cada arco
            L_ArcosPeat.append(tuple(aux))  # esta lista incluye tuplas con
origen, destino y tiempo recorrido a pie
        if DEBUG: print("Lista de arcos de la red Peatonal", L_ArcosPeat)

        # Conjunto de tuplas de arcos con tiempo de trayecto (según longitud
y velocidad) para grafo networkx G2 la red bus
        L_ArcosBus = []
        for i in D_ArcosBus:
            a = D_ArcosBus[i]['ORIGEN']
            b = D_ArcosBus[i]['DESTINO']
            c = D_ArcosBus[i]['FACTOR']
            aux = []
            aux.append(a)
            aux.append(b)
```

```
            aux.append(D_LongitudBus[(a, b)]/VMedia)
            L_ArcosBus.append(tuple(aux))  # esta lista incluye tuplas con
origen, destino y TIEMPO (min) recorrido en bus
        if DEBUG: print("Lista de arcos de la red de Buses", L_ArcosBus)

        # Se borran algunas estructuras de datos que ya no se necesitan
        D_Arcos1.clear()
        D_Arcos2.clear()
        D_ArcosBus.clear()

        # Creamos el grafo G1 de la red peatonal usando Networkx
        G1 = nx.DiGraph()
        G1.add_nodes_from(L_Nodos1)              # Nodos de la red peatonal
        G1.add_weighted_edges_from(L_ArcosPeat) # Arcos de la red peatonal
        #
*************************************************************************
*******
        #
*************************************************************************
*******
        #               Calculando pares OD que se deberán usar en el modelo
        #               Para cada pareja de nodos en la red peatonal (que se
consideran origen y destino de los desplazamientos de las personas sobre la
red) se va a calcular un conjunto de caminos que permiten viajar
        #
*************************************************************************
*******
        #
*************************************************************************
*******

        print('>>> Calculando conjuntos (m,n) de pares OD activos en la red
(a partir de red peatonal)')

        """
        En la red peatonal los caminos que unen cada par de nodos, siempre
que la distancia que los separa sea >= que DISTANCIAMIN
        se almacenan en los diccionarios
        """
        D_NodosPathPeat = {}
        D_ArcosPathPeat = {}
        Contador_Paths = 0
        for m in L_Nodos1:
            for n in L_Nodos1:
                 if m != n:
                     if D_Dist[m, n] >= DISTANCIAMIN:
                         aux = []
                         try:
                             aux = nx.dijkstra_path(G1, m, n,
weight='weight')
                             if DEBUG: print('De %s a %s Path %s'%(m,n,aux))
                             D_NodosPathPeat[(m, n)] = aux
                             if DEBUG:  print("Añadiendo Path peatonal de %s
a %s" % (m, n))
                             Contador_Paths += 1
                         except:
                            pass

        for (m, n) in D_NodosPathPeat:
            D_ArcosPathPeat[(m, n)] = Devuelve_arcos(D_NodosPathPeat[(m, n)])
        G1.clear()
        D_Dist.clear()
```

```
        """
        Hasta aqui tenemos el camino peatonal más corto que une cada origen
con cada destino, siempre que la distancia del origen al destino sea
        mayor que la DISTANCIAMIN. Los pares que se encuentran muy próximos a
menos de DISTANCIAMIN no interesan para diseñar la red de buses, pues
suponemos
        que los usuarios van a ir andando siempre.
        """


        #
*************************************************************************
*******
        #
*************************************************************************
*******
        #               Calculando Pool de Líneas candidatas (POOL ENHANCED)
        #
        #
*************************************************************************
*******
        #
*************************************************************************
*******

        # Creamos el grafo G2 de la red BUS usando Networkx
        L_NodosBus=[i for i in L_Nodos2]
        G2 = nx.DiGraph()
        G2.add_nodes_from(L_NodosBus)
        G2.add_weighted_edges_from(L_ArcosBus)

        # SEGUNDO
        Contador_lineasEn = 0

        # Diccionarios con líneas de bus con mismo código que red original
        # todos ellos tienen de clave cada par de nodos O-D
        D_NodosLineasBusEnNoCode = {}   # Valor: Lista de rutas (k caminos
más cortos) entre ellos (lista de listas)
        D_LongLineasBusEnNoCode = {}    # Valor: Lista de longitudes de las
rutas anteriores
        D_ArcosLineasBusEnNoCode = {}   # Valor: Lista de pares de nodos (en
dos sentidos) incluidos en las rutas
        D_NumeroLineasBusEnNoCode = {}  # Valor: nº rutas

        # Diccionarios con líneas de bus con nodos CODIFICADOS
        D_NodosLineasBusEn = {}
        D_LongLineasBusEn = {}
        D_ArcosLineasBusEn = {}
        D_NumeroLineasBusEn = {}

        for r in L_NodosBus: # Usando RED BUS SIN CODIFICAR
            for s in L_NodosBus:
                if r < s: # Podría ser i < j para no repetir en dos sentidos
                    Paths=[]
                    Lengths=[]
                    try:
                        Paths, Lengths= k2_shortest_paths_lengths(G2, r, s,
ALTERNATIVEROUTELINES, D_LongitudBus, weight='weight')
                        # Paths: Lista de k caminos más cortos (lista de
listas de nodos conectados en el grafo)
                        # Lengths: Lista de la longitud (escalada) de cada
camino se comprueba que cada camino tenga una longitud entre los límites min
```

```
y max y se añade al Dicc
                        for o,p in enumerate(Paths):
                            if Lengths[o] >= Length_min and Lengths[o] <=
Length_max:
                                D_NodosLineasBusEnNoCode[(r,s),o]=p
                                D_LongLineasBusEnNoCode[(r,s),o]=Lengths[o]
                                if DEBUG:  print("Añadiendo lineas de %s a %s
ruta %s longitud %s" % (r, s, o, Lengths[o]))
                                Contador_lineasEn += 1

                                # En estos diccionarios se encuentran los
caminos que cumplen restricción longmin y max
                                # Solo están las líneas en un sentido
                    except:
                        pass
        print("---------- Numero de posibles Líneas (1 sentido)->",
Contador_lineasEn)
        for (r, s),o in D_NodosLineasBusEnNoCode:
            D_ArcosLineasBusEnNoCode[(r,s),o]=
Devuelve_arcos(D_NodosLineasBusEnNoCode[(r, s), o])
            # Función que convierte caminos (lista de nodos) en arcos (pares
de nodos)

         TERCERO
        # Borramos las lineas repetidas que puedan aparecer en las listas
        print(">>> Podando Líneas")

        # Primero se hace una copia del Dicc anterior
        D_NodosLineasBus2EnNoCode = {}
        for (r, s),o in D_NodosLineasBusEnNoCode:
            D_NodosLineasBus2EnNoCode[(r, s),o] =
D_NodosLineasBusEnNoCode[(r, s),o]

        # Para cada línea, comprueba si en otra de las líneas del pool (Dicc)
está contenida la primera
        for (r, s),o in D_NodosLineasBus2EnNoCode:
            for (m, n),p in D_NodosLineasBus2EnNoCode:
                if r != m or s != n:
                    if PRUNEMETHOD == 1:
                        # Utiliza set para no guardar nodos en orden,
comprobando sólo si los nodos están o no en otra ruta más larga
                        aa = set(D_NodosLineasBus2EnNoCode[(r, s),o])
                        bb = set(D_NodosLineasBus2EnNoCode[(m, n),p])
                        if bb.issubset(aa):  # entonces la linea [m,n] está
contenida en [r,s]
                            if DEBUG: print("Eliminando linea (%s,%s), camino
%s" % (m, n, p))
                            if RANDOMIZE==0:
                                D_NodosLineasBusEnNoCode.pop(((m, n), p), 1)
                                D_ArcosLineasBusEnNoCode.pop(((m, n), p), 1)
                                D_LongLineasBusEnNoCode.pop(((m, n), p), 1)
                            else:
                                nr=random()
                                if nr > SURVIVALPERCEN:
                                    D_NodosLineasBusEnNoCode.pop(((m, n), p),
1)
                                    D_ArcosLineasBusEnNoCode.pop(((m, n), p),
1)
                                    D_LongLineasBusEnNoCode.pop(((m, n), p),
1)

                    elif PRUNEMETHOD == 2:
```

```
                        # aa y bb son listas, por lo que comprueba si los
nodos están en otra ruta más larga en el mismo orden
                        aa = D_NodosLineasBus2EnNoCode[(r, s), o]
                        bb = D_NodosLineasBus2EnNoCode[(m, n), p]
                        if RANDOMIZE==0:
                            if len(aa) >= len(bb):
                                flag1 = any(bb == aa[i:i+len(bb)] for i in
range(0,len(aa)))
                                if flag1:
                                    if DEBUG: print("Eliminando linea
(%s,%s), camino %s" % (m, n, p))
                                    D_NodosLineasBusEnNoCode.pop(((m, n), p),
1)
                                    D_ArcosLineasBusEnNoCode.pop(((m, n), p),
1)
                                    D_LongLineasBusEnNoCode.pop(((m, n), p),
1)
                            elif len(aa) < len(bb):
                                flag2 = any(aa == bb[i:i + len(aa)] for i in
range(0, len(bb)))
                                if flag2:
                                    if DEBUG: print("Eliminando linea
(%s,%s), camino %s" % (m, n, p))
                                    D_NodosLineasBusEnNoCode.pop(((r, s), o),
1)
                                    D_ArcosLineasBusEnNoCode.pop(((r, r),o),
1)
                                    D_LongLineasBusEnNoCode.pop(((r, s),o),
1)

                        else:
                            nr = random()
                            if len(aa) >= len(bb):
                                flag1 = any(bb == aa[i:i+len(bb)] for i in
range(0,len(aa)))
                                if flag1:
                                    if nr > SURVIVALPERCEN:
                                        if DEBUG: print("Eliminando linea
(%s,%s), camino %s" % (m, n, p))
                                        D_NodosLineasBusEnNoCode.pop(((m, n),
p), 1)
                                        D_ArcosLineasBusEnNoCode.pop(((m, n),
p), 1)
                                        D_LongLineasBusEnNoCode.pop(((m, n),
p), 1)
                            elif len(aa) < len(bb):
                                flag2 = any(aa == bb[i:i + len(aa)] for i in
range(0, len(bb)))
                                if flag2:
                                    if nr > SURVIVALPERCEN:
                                        if DEBUG: print("Eliminando linea
(%s,%s), camino %s" % (m, n, p))
                                        D_NodosLineasBusEnNoCode.pop(((r, s),
o), 1)
                                        D_ArcosLineasBusEnNoCode.pop(((r,
r),o), 1)
                                        D_LongLineasBusEnNoCode.pop(((r,
s),o), 1)


        for (r,s),o in D_NodosLineasBusEnNoCode:
            D_NumeroLineasBusEnNoCode[(r,s)]=sum(1 for (m,n),o in
```

```
D_NodosLineasBusEnNoCode if m==r and n==s)

        #### Por ahora los nodos de la red peatonal y de red bus tienen los
mismos códigos, aunque la red bus será en principio menor (menos densa) que
la red peatonal.
            #
*************************************************************************
*******
            #
*************************************************************************
*******
            #         Utilizando las funciones de codificación, cada nodo de
cada linea se recodifica de una forma específica.
            #         Si por un nodo de la red peatonal pasan varias líneas,
se hacen varias copias del nodo, cada uno con un código específico. Entonces,
para cada nodo peatonal puede existir un conjunto de nodos partners, tantos
como líneas pasen por ese nodo. Además estos nodos artificales se conectan
con el nodo peatonal por dos arcos transfer, uno en cada dirección.
            #
*************************************************************************
*******
            #
*************************************************************************
*******

        # Para codificar los nodos del pool de lineas de bus

        Num_Lineas_generadas = len(D_NodosLineasBusEnNoCode)
        SumaCode2 = len(L_Nodos1) + 1
        SumaCode1 = Num_Lineas_generadas + 1

        ToNode={}
        ToCode={}
        LineToNumber={}
        NumberToLine={}
        k=0
        Bus=1
        for ((r,s),o) in D_NodosLineasBusEnNoCode:
            k=k+1
            LineToNumber[(r,s),o]=k
            NumberToLine[k]=((r,s),o)
            for i in D_NodosLineasBusEnNoCode[(r,s),o]:
                if (i,k) not in ToCode:
                    ToCode.update({(k,i):code(Bus,k,i)})
        for (k,i) in ToCode:
            ToNode.update({ToCode[(k,i)]:i})

        #Recodificando Lineas
        for ((r,s),o) in D_NodosLineasBusEnNoCode:
            aux=D_NodosLineasBus2EnNoCode[(r,s),o]
            aux2=[]
            for i in aux:
                k=LineToNumber[(r,s),o]
                aux2.append(code(Bus,k,i))
            D_NodosLineasBusEn[(r,s),o]=aux2

        for (r,s),o in D_NodosLineasBusEn:
            D_NumeroLineasBusEn[(r,s)]=sum(1 for (m,n),o in
D_NodosLineasBusEn if m==r and n==s)

        for (r, s),o in D_NodosLineasBusEn:
            D_ArcosLineasBusEn[(r,s),o]=
```

```
Devuelve_arcos(D_NodosLineasBusEn[(r, s), o])

        for ((r, s),o) in D_NodosLineasBusEn:
            D_LongLineasBusEn[(r,s),o]=D_LongLineasBusEnNoCode[(r,s),o]

        #Escribimos en un archivo txt las lineas codificadas y en otro txt
sin codificar
        print('>>> Escribiendo Pool de líneas Enhanced a disco')
        File='CODE_' + FileLinePool
        file = io.open(File, "w")
        file.write('%s\n' % (Num_Lineas_generadas))
        for key, value in D_NodosLineasBusEn.items():
            file.write('%s:%s\n' % (key, value))
        for key, value in D_ArcosLineasBusEn.items():
            file.write('%s:%s\n' % (key, value))
        for key, value in D_LongLineasBusEn.items():
            file.write('%s:%s\n' % (key, value))
        file.write('%s\n'%(len(D_NumeroLineasBusEn.items())))
        for key, value in D_NumeroLineasBusEn.items():
            file.write('%s:%s\n' % (key, value))
        file.close()
        File='NOCODE_'+ FileLinePool
        file = io.open(File, "w")
        file.write('%s\n' % (Num_Lineas_generadas))
        for key, value in D_NodosLineasBusEnNoCode.items():
            file.write('%s:%s\n' % (key, value))
        for key, value in D_ArcosLineasBusEnNoCode.items():
            file.write('%s:%s\n' % (key, value))
        for key, value in D_LongLineasBusEnNoCode.items():
            file.write('%s:%s\n' % (key, value))
        file.write('%s\n' % (len(D_NumeroLineasBusEnNoCode.items())))
        for key, value in D_NumeroLineasBusEnNoCode.items():
            file.write('%s:%s\n' % (key, value))
        file.close()

        D_NodosLineasBusEnNoCode.clear()
        D_LongLineasBusEnNoCode.clear()
        D_ArcosLineasBusEnNoCode.clear()
        D_NumeroLineasBusEnNoCode.clear()
        D_NodosLineasBus2EnNoCode.clear()

        Num_Lineas_generadas = len(D_NodosLineasBusEn)
        SumaCode2=len(L_Nodos1)+1
        SumaCode1=Num_Lineas_generadas+1

        L_NodosBus.clear()
        G2.clear()

        print("---------- Numero de Líneas despues de podar Enhanced-> %s" %
len(D_NodosLineasBusEn))
        print(">>> Terminado Pool de Líneas candidatas", FileLinePool)
```

## 7.2 Código Método 2

```
import networkx as nx
import copy
from IOfunctionsExcel5 import *
from MyPlotandNetworkFunctions5 import *
import io
import math
```

```
ALTERNATIVELINES=10
LIMITE_TIEMPO=60       # Máximo tiempo de trayecto de una línea completa
LIMITE_NODOS=9         # Máximo nº de nodos incluidos en un camino
                       # Esta restricción no se incluye en Método 1

# Funciones para codificar y descodificar la red de bus

SumaCode1 = 10
SumaCode2 = 10

def code(Capa, Linea, Nodo):
    return (SumaCode1 * SumaCode2 * Capa + SumaCode2 * Linea + Nodo)


def decode(Code):
    Resto = Code % (SumaCode1 * SumaCode2)
    Capa = (Code - Resto) / (SumaCode1 * SumaCode2)
    Resto2 = Resto % SumaCode2
    Linea = (Resto - Resto2) / SumaCode2
    Nodo = Resto2
    return int(Capa), int(Linea), int(Nodo)


def decodeCapa(Code):
    a, b, c = decode(Code)
    return int(a)


def decodeLinea(Code):
    a, b, c = decode(Code)
    return int(b)


def decodeNodo(Code):
    a, b, c = decode(Code)
    return int(c)


#-----------------------------------------------------------------------------

class node:
    def __init__(self, Code, Network):
        self.Code=Code
        self.Network=Network
        self.Accounted=0
        self.NRepetitions = 0

#-----------------------------------------------------------------------------

class link:
    def __init__(self, Code, Origin, Destination, Network):
        self.Code=Code
        self.Origin=Origin
        self.Destination=Destination
        self.Network = Network
        self.Longitud=math.sqrt((self.Network.dNodesCoord[Origin]['X']-
self.Network.dNodesCoord[Destination]['X'])**2+
(self.Network.dNodesCoord[Origin]['Y'] -
self.Network.dNodesCoord[Destination]['Y']) **2 ) * self.Network.escala
        self.Time = self.Longitud / self.Network.velocidad
```

```
        self.Carga=0

#-----------------------------------------------------------------------------

class network:

    def __init__(self, wbRed, wbMat, sheet1, sheet2, Data):

        self.wbRed = wbRed  # excel donde están los datos de Redes
        self.wbMat = wbMat  # excel donde están los datos de Matrices OD
        self.sheet1 = sheet1
        self.sheet2 = sheet2
        self.escala=4000/1500
        self.velocidad= 60*(1000/60)
        self.lNodes=[]
        self.dNodesCoord={}
        self.dLinks={}
        self.dDemand={}


        self.dObjNodes = {}
        self.dObjLinks = {}
        self.Totaldemand=0

        self.read(Data)
        self.duplicate_links()


    def read(self,Data):
        self.lNodes = Read_Excel_to_List(self.wbRed, self.sheet1,
Data['Nodos'][0], Data['Nodos'][1])
        self.dNodesCoord = Read_Excel_to_NesteDic(self.wbRed, self.sheet1,
Data['NodosCoord'][0], Data['NodosCoord'][1])

        for i in self.lNodes:
            self.dObjNodes[i] = node(i,self)
        self.dLinks = Read_Excel_to_NesteDic_TupleInt(self.wbRed,
self.sheet1, Data['Arcos'][0], Data['Arcos'][1])

        for (i,j) in self.dLinks:
            self.dObjLinks[(i,j)] = link((i,j),i,j,self)
        self.dDemand = Read_Excel_to_NesteDic(self.wbMat, self.sheet2,
Data['Demanda'][0], Data['Demanda'][1])

        aux=0
        for i in self.lNodes:
            for j in self.lNodes:
                aux+=self.dDemand[i][j]

        self.Totaldemand=aux


    def duplicate_links(self):
        aux={}
        for (i,j) in self.dLinks:
            aux[(j,i)]={'ORIGEN':j, 'DESTINO':i}

        for (i, j) in self.dLinks:
            self.dObjLinks[(j,i)]=link((j,i), j, i, self)

        self.dLinks.update(aux)
```

```
#---------------------------------------------------------------------------

class poolOD:
    def __init__(self, pool, Code, Origin, Destination, Demand):

        self.Code=Code
        self.Origen=Origin
        self.Destino=Destination
        self.Demand=Demand
        self.atendido=0
        self.dLines= {}
        self.Accounted=0
        self.NumberLines=0
        self.NumberRepetitions=0



class poolLine:

    def __init__(self, pool, Code , Origin, Destination, lNodes):

        self.Code=Code
        self.Origin=Origin
        self.Destination=Destination
        self.Pool=pool
        self.Network=pool.Network
        self.Demand=0
        self.NumNodos=0
        self.Longitud=0
        self.lNodes=lNodes
        self.lLinks=[]
        self.lObjPoolODs=[]
        self.NumberPoolODs=0

        self.calculate_demand()
        self.completa()

    def completa(self):
        self.lLinks=Devuelve_arcos(self.lNodes)
        self.NumNodos=len(self.lNodes)
        self.Longitud=sum(self.Network.dObjLinks[(i,j)].Longitud for (i,j) in
self.lLinks)

    def update(self, obj):
        self.lObjPoolODs.append(obj.Code)
        self.NumberPoolODs+=1

    def calculate_demand(self):
        for i in self.lNodes:
            for j in self.lNodes:
                self.Demand+=self.Network.dDemand[i][j]


class pool:

    def __init__(self, network, Code):

        self.Code=Code
        self.Network=network
        self.dDemand=network.dDemand
        self.dObjNodes={}
        self.dObjLines={}
```

```
        self.NumLines=0
        self.dObjODs={}
        self.dObjLinesReduced={}
        self.dRepetitions={}
        self.dRepetitionsNodes={}
        self.CuentaODsConPaths=0
        self.CuentaNodesConPaths=0

        self.generate_ODs()
        self.ordena_ODs()
        self.generate_lines()
        self.duplicate_links()


    def generate_ODs(self):
        for i in self.Network.lNodes:
            for j in self.Network.lNodes:
                if i<j:
                    self.dObjODs[(i,j)]=poolOD(self,(i,j),i,j,
self.dDemand[i][j]+self.dDemand[j][i])
                    # Mete como atributo de pool la demanda ida + demanda
vuelta de cada par de nodos

    def ordena_ODs(self):
        aux=sorted(self.dObjODs.items(), key=lambda x: x[1].Demand,
reverse=True)
        self.dObjODs=dict(aux)
        # Diccionario ordenado de M a m demanda ida+vuelta, de clave el par
nodos y valor corresponde al valor de demanda ida+vuelta

    ##### Esta función hace el pool
    def generate_lines(self):
        # Crear grafo
        ListaNodos=[i for i in self.Network.lNodes]
        ListaArcos=[]
        dLinks={}
        for l in self.Network.dObjLinks.values():
            ListaArcos.append((l.Origin,l.Destination,l.Time))
            dLinks[l.Code]=l.Time
            # Diccionario con clave cada par OD y valor el tiempo de trayecto

        G=nx.DiGraph()
        G.add_nodes_from(ListaNodos)
        G.add_weighted_edges_from(ListaArcos)
        CODIF=1
        for (i,j) in self.dObjODs:
            # self.dRepetitions[(i,j)]=0
            self.dRepetitionsNodes[i]=0
            self.dRepetitionsNodes[j]=0
            try:
                paths, lengths =
k2_shortest_paths_lengths(G,i,j,ALTERNATIVELINES, dLinks, weight='weight')
                # Crear objetos poolLine y añadir al pool

                for o,p in enumerate(paths):
                    if lengths[o] <= LIMITE_TIEMPO and len(p) <=
LIMITE_NODOS:
                        self.dObjLines[(i,j,o)]=poolLine(self, CODIF, i, j,
p)
                        self.NumLines+=1
                        # Para cada nodo del camino y su sucesor, se añade la
demanda ida+vuelta que satisface
```

```
                        for node in p:
                            for node2 in p:
                                if node < node2:
                                    self.dObjLines[(i, j,
o)].update(self.dObjODs[(node, node2)])
                                    self.dObjODs[(node,node2)].NumberLines+=1

self.dObjODs[(node,node2)].dLines[(i,j,o)]=self.dObjLines[(i,j,o)].Demand
                        CODIF+=1

            except:
                pass

        lineasordered=sorted(self.dObjLines.items(), key=lambda x:
x[1].Demand, reverse=True)
        self.dObjLines=dict(lineasordered)
        # Diccionario ordenado de M a m demanda ida+vuelta, con clave
(origen, destino, nº del camino)
        # y valor corresponde al valor de demanda ida+vuelta

        # Ahora se eliminan las peores
        # Se copian las mejores siempre que todos los nodos estén
representados por lo menos con XX Lineas (dar valor "Corte")
        Corte=25
        for m in self.Network.dObjNodes:
            self.dRepetitionsNodes[m] = 0

        for (i,j,k) in self.dObjLines:
            self.dObjLinesReduced[(i,j,k)]=self.dObjLines[(i,j,k)]
            for m in self.dObjLines[(i,j,k)].lNodes:
                self.Network.dObjNodes[m].NRepetitions+=1
                self.dRepetitionsNodes[m] += 1
            Minimo=min(self.dRepetitionsNodes.values())
            if Minimo > Corte: # Todos los nodos estan en CORTE lineas
                break


    def duplicate_links(self):

        for (i,j,k) in self.dObjLinesReduced:
            aux=[]
            for (a,b) in self.dObjLinesReduced[(i,j,k)].lLinks:
                aux.append((b,a))
            self.dObjLinesReduced[(i, j, k)].lLinks =
self.dObjLinesReduced[(i, j, k)].lLinks + aux



    def write_lines_NoCode(self, LINES_FILE_EN):

        Num_Lineas_generadas=len(self.dObjLinesReduced)
        Num_Lineas_total=len(self.dObjLines)
        print('>>> Total líneas candidatas: ', Num_Lineas_total)
        print('>>> Escribiendo Pool de líneas Enhanced NoCode a disco. Son %s
líneas' %(Num_Lineas_generadas))
        file = io.open(LINES_FILE_EN, "w")
        file.write('%s\n' % (Num_Lineas_generadas))
        # Par origen-destino, nº del camino y Nodos que lo forman:
        for (i,j,k) in self.dObjLinesReduced: #Nodos
            file.write('%s:%s\n' % (((i,j),k),
self.dObjLinesReduced[(i,j,k)].lNodes))
        # Par origen-destino, nº del camino y Arcos que lo forman (ida y
```

```
vuelta):
        for (i, j, k) in self.dObjLinesReduced:
            file.write('%s:%s\n' % (((i,j),k),
self.dObjLinesReduced[(i,j,k)].lLinks))
        # Par origen-destino, nº del camino y Tiempo del trayecto:
        for (i, j, k) in self.dObjLinesReduced:
            file.write('%s:%s\n' % (((i,j),k),
self.dObjLinesReduced[(i,j,k)].Longitud))


        file.close()


    def write_lines_Code(self, LINES_FILE_EN):

        Num_Lineas_generadas=len(self.dObjLinesReduced)
        # print('>>> Escribiendo Pool de líneas Enhanced Code a disco. Son %s
líneas' %(Num_Lineas_generadas))
        file = io.open(LINES_FILE_EN, "w")
        file.write('%s\n' % (Num_Lineas_generadas))
        # Par origen-destino, nº del camino y Nodos que lo forman:
        for key, value in D_NodosLineasBusEn.items():
            file.write('%s:%s\n' % (key, value))
        for key, value in D_ArcosLineasBusEn.items():
            file.write('%s:%s\n' % (key, value))
        for key, value in D_LongLineasBusEn.items():
            file.write('%s:%s\n' % (key, value))
        file.write('%s\n'%(len(D_NumeroLineasBusEn.items())))
        for key, value in D_NumeroLineasBusEn.items():
            file.write('%s:%s\n' % (key, value))


#-------------------------------------------------------------------------


Data={'Nodos': ['B2','B17'],
'NodosCoord': ['B1','D17'],
      'Arcos': ['I1','K25'],
       'Demanda':['D8','T24']}

EscenarioRed=[i for i in range(10)]
EscenarioMat=[i for i in range(10)]

HojasRed=['RED1', 'RED2', 'RED3', 'RED4','RED5', 'RED6','RED7','RED8','RED9',
'RED10']
HojasMatriz=['OD1', 'OD2', 'OD3', 'OD4','OD5', 'OD6','OD7', 'OD8','OD9',
'OD10']

METODO='M2'

for esc in EscenarioRed:
    sheetRed = HojasRed[esc]
    for mat in EscenarioMat:
        sheetMat = HojasMatriz[mat]
        Metodo2 = network('Redes.xlsx', 'MATOD.xlsx', sheetRed, sheetMat,
Data)
        P1=pool(Metodo2, 1)


        # Se escribe el pool SIN CODIFICAR
        FileLinePool = ''
        P1.write_lines_NoCode('NOCODE_' + FileLinePool+ 'Pool_' + METODO +
```

```
'_' + HojasRed[esc] + '_' + HojasMatriz[mat] + '.txt')

        # SE CODIFICAN LOS RESULTADOS PARA LECTURA EN MODELO

        # Diccionarios con líneas de bus con nodos CODIFICADOS
        D_NodosLineasBusEn = {}
        D_LongLineasBusEn = {}
        D_ArcosLineasBusEn = {}
        D_NumeroLineasBusEn = {}

         # Para codificar los nodos del pool de lineas de bus
        Num_Lineas_generadas=len(P1.dObjLinesReduced)
        SumaCode2 = len(Metodo2.lNodes) + 1
        SumaCode1 = Num_Lineas_generadas + 1

        ToNode={}
        ToCode={}
        LineToNumber={}
        NumberToLine={}
        k=0
        Bus=1
        for (r,s,o) in P1.dObjLinesReduced:
            k=k+1
            LineToNumber[(r,s),o]=k
            NumberToLine[k]=((r,s),o)
            for i in P1.dObjLinesReduced[(r,s,o)].lNodes:

                if (i,k) not in ToCode:
                    ToCode.update({(k,i):code(Bus,k,i)})
        for (k,i) in ToCode:
            ToNode.update({ToCode[(k,i)]:i})

        #Recodificando Lineas
        for (r,s,o) in P1.dObjLinesReduced:
            aux=P1.dObjLines[(r,s,o)].lNodes
            aux2=[]
            for i in aux:
                k=LineToNumber[(r,s),o]
                aux2.append(code(Bus,k,i))
            D_NodosLineasBusEn[(r,s),o]=aux2

        for (r,s),o in D_NodosLineasBusEn:
            D_NumeroLineasBusEn[(r,s)]=sum(1 for (m,n),o in
D_NodosLineasBusEn if m==r and n==s)

        for (r, s),o in D_NodosLineasBusEn:
            D_ArcosLineasBusEn[(r,s),o]=
Devuelve_arcos(D_NodosLineasBusEn[(r, s), o])

        for ((r, s),o) in D_NodosLineasBusEn:
            D_LongLineasBusEn[(r,s),o]=P1.dObjLinesReduced[(r,s,o)].Longitud

        # Se escribe el pool CODIFICADO
        FileLinePool = ''
        P1.write_lines_Code('CODE_' + FileLinePool+ 'Pool_' + METODO + '_' +
HojasRed[esc] + '_' + HojasMatriz[mat] + '.txt')
```

## 7.3 Código Método 3

```
import networkx as nx
import copy
from IOfunctionsExcel5 import *
from MyPlotandNetworkFunctions5 import *
import io
import math


ALTERNATIVELINES=10
LIMITE_TIEMPO=60       # Máximo tiempo de trayecto de una línea completa
LIMITE_NODOS=9         # Máximo nº de nodos incluidos en un camino
                       # Esta restricción no se incluye en Método 1

# Funciones para codificar y descodificar la red de bus

SumaCode1 = 10
SumaCode2 = 10

def code(Capa, Linea, Nodo):
    return (SumaCode1 * SumaCode2 * Capa + SumaCode2 * Linea + Nodo)




def decode(Code):
    Resto = Code % (SumaCode1 * SumaCode2)
    Capa = (Code - Resto) / (SumaCode1 * SumaCode2)
    Resto2 = Resto % SumaCode2
    Linea = (Resto - Resto2) / SumaCode2
    Nodo = Resto2
    return int(Capa), int(Linea), int(Nodo)


def decodeCapa(Code):
    a, b, c = decode(Code)
    return int(a)


def decodeLinea(Code):
    a, b, c = decode(Code)
    return int(b)


def decodeNodo(Code):
    a, b, c = decode(Code)
    return int(c)


#-----------------------------------------------------------------------------

class node:
    def __init__(self, Code, Network):
        self.Code=Code
        self.Network=Network
        self.Accounted=0
        self.NRepetitions=0

#-----------------------------------------------------------------------------
```

```
class link:
    def __init__(self, Code, Origin, Destination, Network):
        self.Code=Code
        self.Origin=Origin
        self.Destination=Destination
        self.Network = Network
        self.Longitud=math.sqrt((self.Network.dNodesCoord[Origin]['X']-
self.Network.dNodesCoord[Destination]['X'])**2+
(self.Network.dNodesCoord[Origin]['Y'] -
self.Network.dNodesCoord[Destination]['Y']) **2 ) * self.Network.escala
        self.Time = self.Longitud / self.Network.velocidad
        self.Carga=0

#-----------------------------------------------------------------------------

class network:

    def __init__(self, wbRed, wbMat, sheet1, sheet2, Data):

        self.wbRed = wbRed  # excel donde están los datos de Redes
        self.wbMat = wbMat  # excel donde están los datos de Matrices OD
        self.sheet1 = sheet1
        self.sheet2 = sheet2
        self.escala=4000/1500
        self.velocidad= 60*(1000/60)
        self.lNodes=[]
        self.dNodesCoord={}
        self.dLinks={}
        self.dDemand={}


        self.dObjNodes = {} #Network
        self.dObjLinks = {}
        self.Totaldemand=0

        self.read(Data)
        self.duplicate_links()


    def read(self,Data):
        self.lNodes = Read_Excel_to_List(self.wbRed, self.sheet1,
Data['Nodos'][0], Data['Nodos'][1])
        self.dNodesCoord = Read_Excel_to_NesteDic(self.wbRed, self.sheet1,
Data['NodosCoord'][0], Data['NodosCoord'][1])

        for i in self.lNodes:
            self.dObjNodes[i] = node(i,self)

        self.dLinks = Read_Excel_to_NesteDic_TupleInt(self.wbRed,
self.sheet1, Data['Arcos'][0], Data['Arcos'][1])

        for (i,j) in self.dLinks:
            self.dObjLinks[(i,j)] = link((i,j),i,j,self)

        self.dDemand = Read_Excel_to_NesteDic(self.wbMat, self.sheet2,
Data['Demanda'][0], Data['Demanda'][1])
        aux=0
        for i in self.lNodes:
            for j in self.lNodes:
                aux+=self.dDemand[i][j]

        self.Totaldemand=aux
```

```
    def duplicate_links(self):
        aux={}
        for (i,j) in self.dLinks:
            aux[(j,i)]={'ORIGEN':j, 'DESTINO':i}

        for (i, j) in self.dLinks:
            self.dObjLinks[(j,i)]=link((j,i), j, i, self)

        self.dLinks.update(aux)

#----------------------------------------------------------------------------

class poolOD:
    def __init__(self, pool, Code, Origin, Destination, Demand):

        self.Code=Code
        self.Origen=Origin
        self.Destino=Destination
        self.Demand=Demand
        self.atendido=0
        self.dLines= {}
        self.Accounted=0
        self.NumberLines=0
        self.NumberRepetitions=0


class poolLine:

    def __init__(self, pool, Code , Origin, Destination, lNodes):

        self.Code=Code
        self.Origin=Origin
        self.Destination=Destination
        self.Pool=pool
        self.Network=pool.Network
        self.Demand=0
        self.NumNodos=0
        self.Longitud=0
        self.lNodes=lNodes
        self.lLinks=[]
        self.lObjPoolODs=[]
        self.NumberPoolODs=0

        self.calculate_demand()
        self.completa()

    def completa(self):
        self.lLinks=Devuelve_arcos(self.lNodes)
        self.NumNodos=len(self.lNodes)
        self.Longitud=sum(self.Network.dObjLinks[(i,j)].Longitud for (i,j) in
self.lLinks)

    def update(self, obj):
        self.lObjPoolODs.append(obj.Code)
        self.NumberPoolODs+=1

    def calculate_demand(self):
        for i in self.lNodes:
            for j in self.lNodes:
                self.Demand+=self.Network.dDemand[i][j]
```

```
class pool:

    def __init__(self, network, Code):

        self.Code=Code
        self.Network=network
        self.dDemand=network.dDemand
        self.dObjLines={}
        self.NumLines=0
        self.dObjODs={}
        self.dObjLinesReduced={}
        self.dRepetitions={}
        self.dRepetitionsNodes={}
        self.CuentaODsConPaths=0

        self.generate_ODs()
        self.ordena_ODs()
        self.generate_lines()
        self.duplicate_links()


    def generate_ODs(self):
        for i in self.Network.lNodes:
            for j in self.Network.lNodes:
                if i<j:
                    self.dObjODs[(i,j)]=poolOD(self,(i,j),i,j,
self.dDemand[i][j]+self.dDemand[j][i])
                    # Mete como atributo de pool la demanda ida + demanda
vuelta de cada par de nodos

    def ordena_ODs(self):
        aux=sorted(self.dObjODs.items(), key=lambda x: x[1].Demand,
reverse=True)
        self.dObjODs=dict(aux)
        # Dicc ordenado de M a m demanda ida+vuelta, de clave el par nodos
        # y valor corresponde al valor de demanda ida+vuelta


    ##### Esta función hace el pool
    def generate_lines(self):
        # Crear grafo
        ListaNodos=[i for i in self.Network.lNodes]
        ListaArcos=[]
        dLinks={}
        for l in self.Network.dObjLinks.values():
            ListaArcos.append((l.Origin,l.Destination,l.Time))
            dLinks[l.Code]=l.Time
            # Dicc con clave cada par OD y valor el tiempo de trayecto

        G=nx.DiGraph()
        G.add_nodes_from(ListaNodos)
        G.add_weighted_edges_from(ListaArcos)
        CODIF=1
        for (i,j) in self.dObjODs:
            self.dRepetitions[(i,j)]=0
            try:
                paths, lengths =
k2_shortest_paths_lengths(G,i,j,ALTERNATIVELINES, dLinks, weight='weight')
                # Crear objetos poolLine y añadir al pool
```

```
                for o,p in enumerate(paths):
                    if lengths[o] <= LIMITE_TIEMPO and len(p) <=LIMITE_NODOS:
                        self.dObjLines[(i,j,o)]=poolLine(self, CODIF, i, j,p)
                        self.NumLines+=1
                        # Para cada nodo del camino y su sucesor, se añade la
demanda ida+vuelta que satisface
                        for node in p:
                            for node2 in p:
                                if node < node2:
                                    self.dObjLines[(i, j,
o)].update(self.dObjODs[(node, node2)])
                                    self.dObjODs[(node,node2)].NumberLines+=1

self.dObjODs[(node,node2)].dLines[(i,j,o)]=self.dObjLines[(i,j,o)].Demand
                        CODIF+=1

            except:
                pass

        lineasordered=sorted(self.dObjLines.items(), key=lambda x:
x[1].Demand, reverse=True)
        self.dObjLines=dict(lineasordered)
        # Diccionario ordenado de M a m demanda ida+vuelta, con clave
(origen, destino, nº del camino) y valor corresponde la demanda ida+vuelta

        # Ahora se eliminan las peores
        # Se copian las mejores siempre que todos los pares OD estén
representados por lo menos con 1 linea del pool
        Corte=1
        for (i,j,k) in self.dObjLines:
            self.dObjLinesReduced[(i,j,k)]=self.dObjLines[(i,j,k)]
            self.dObjODs[(i,j)].Accounted+=1

            for n1 in  self.dObjLinesReduced[(i,j,k)].lNodes:
                for n2 in self.dObjLinesReduced[(i, j, k)].lNodes:
                    if n1 < n2:
                        self.dRepetitions[(n1, n2)]+=1

            Minimo=min(self.dRepetitions.values())
            if Minimo > Corte: # Todos los pares estan en CORTE lineas
                break


        for (i,j) in self.dObjODs:
            if self.dObjODs[(i,j)].Accounted>1: self.CuentaODsConPaths+=1


    def duplicate_links(self):

        for (i,j,k) in self.dObjLinesReduced:
            aux=[]
            for (a,b) in self.dObjLinesReduced[(i,j,k)].lLinks:
                aux.append((b,a))
            self.dObjLinesReduced[(i, j, k)].lLinks =
self.dObjLinesReduced[(i, j, k)].lLinks + aux


    def write_lines_NoCode(self, LINES_FILE_EN):

        Num_Lineas_generadas=len(self.dObjLinesReduced)
        print('>>> Escribiendo Pool de líneas Enhanced NoCode a disco. Son %s
líneas' %(Num_Lineas_generadas))
```

```
        file = io.open(LINES_FILE_EN, "w")
        file.write('%s\n' % (Num_Lineas_generadas))
        # Par origen-destino, nº del camino y Nodos que lo forman:
        for (i,j,k) in self.dObjLinesReduced: #Nodos
            file.write('%s:%s\n' % (((i,j),k),
self.dObjLinesReduced[(i,j,k)].lNodes))
        # Par origen-destino, nº del camino y Arcos que lo forman (ida y
vuelta):
        for (i, j, k) in self.dObjLinesReduced:
            file.write('%s:%s\n' % (((i,j),k),
self.dObjLinesReduced[(i,j,k)].lLinks))
        # Par origen-destino, nº del camino y Tiempo del trayecto:
        for (i, j, k) in self.dObjLinesReduced:
            file.write('%s:%s\n' % (((i,j),k),
self.dObjLinesReduced[(i,j,k)].Longitud))

        # Corte:
        file.write('%s\n' % (self.CuentaODsConPaths))
        for (i,j) in self.dObjODs:
            if self.dObjODs[(i,j)].Accounted>1:
                file.write('%s:%s\n' % ((i,j),
self.dObjODs[(i,j)].Accounted))
        file.close()


    def write_lines_Code(self, LINES_FILE_EN):

        Num_Lineas_generadas=len(self.dObjLinesReduced)
        print('>>> Escribiendo Pool de líneas Enhanced Code a disco. Son %s
líneas' %(Num_Lineas_generadas))
        file = io.open(LINES_FILE_EN, "w")
        file.write('%s\n' % (Num_Lineas_generadas))
        for key, value in D_NodosLineasBusEn.items():
            file.write('%s:%s\n' % (key, value))
        for key, value in D_ArcosLineasBusEn.items():
            file.write('%s:%s\n' % (key, value))
        for key, value in D_LongLineasBusEn.items():
            file.write('%s:%s\n' % (key, value))
        file.write('%s\n'%(len(D_NumeroLineasBusEn.items())))
        for key, value in D_NumeroLineasBusEn.items():
            file.write('%s:%s\n' % (key, value))

#-------------------------------------------------------------------------

Data={'Nodos': ['B2','B17'],
'NodosCoord': ['B1','D17'],
      'Arcos': ['I1','K25'],
       'Demanda':['D8','T24']}

EscenarioRed=[i for i in range(10)]
EscenarioMat=[i for i in range(10)]

HojasRed=['RED1', 'RED2', 'RED3', 'RED4','RED5', 'RED6','RED7','RED8','RED9',
'RED10']
HojasMatriz=['OD1', 'OD2', 'OD3', 'OD4','OD5', 'OD6','OD7', 'OD8','OD9',
'OD10']

METODO='M3'

for esc in EscenarioRed:
    sheetRed = HojasRed[esc]
    for mat in EscenarioMat:
```

```
        sheetMat = HojasMatriz[mat]
        Metodo3 = network('Redes.xlsx', 'MATOD.xlsx', sheetRed, sheetMat,
Data)
        P1=pool(Metodo3, 1)

        # Se escribe el pool SIN CODIFICAR
        FileLinePool = ''
        P1.write_lines_NoCode('NOCODE_' + FileLinePool+ 'Pool_' + METODO +
'_' + HojasRed[esc] + '_' + HojasMatriz[mat] + '.txt')

        # SE CODIFICAN LOS RESULTADOS PARA LECTURA EN MODELO
        # Diccionarios con líneas de bus con nodos CODIFICADOS
        D_NodosLineasBusEn = {}
        D_LongLineasBusEn = {}
        D_ArcosLineasBusEn = {}
        D_NumeroLineasBusEn = {}

         # Para codificar los nodos del pool de lineas de bus
        Num_Lineas_generadas=len(P1.dObjLinesReduced)
        SumaCode2 = len(Metodo3.lNodes) + 1
        SumaCode1 = Num_Lineas_generadas + 1

        ToNode={}
        ToCode={}
        LineToNumber={}
        NumberToLine={}
        k=0
        Bus=1
        for (r,s,o) in P1.dObjLinesReduced:
            k=k+1
            LineToNumber[(r,s),o]=k
            NumberToLine[k]=((r,s),o)
            for i in P1.dObjLinesReduced[(r,s,o)].lNodes:
                if (i,k) not in ToCode:
                    ToCode.update({(k,i):code(Bus,k,i)})
        for (k,i) in ToCode:
            ToNode.update({ToCode[(k,i)]:i})
        #Recodificando Lineas
        for (r,s,o) in P1.dObjLinesReduced:
            aux=P1.dObjLines[(r,s,o)].lNodes
            aux2=[]
            for i in aux:
                k=LineToNumber[(r,s),o]
                aux2.append(code(Bus,k,i))
            D_NodosLineasBusEn[(r,s),o]=aux2

        for (r,s),o in D_NodosLineasBusEn:
            D_NumeroLineasBusEn[(r,s)]=sum(1 for (m,n),o in
D_NodosLineasBusEn if m==r and n==s)

        for (r, s),o in D_NodosLineasBusEn:
            D_ArcosLineasBusEn[(r,s),o]=
Devuelve_arcos(D_NodosLineasBusEn[(r, s), o])

        for ((r, s),o) in D_NodosLineasBusEn:
            D_LongLineasBusEn[(r,s),o]=P1.dObjLinesReduced[(r,s,o)].Longitud

        # Se escribe el pool CODIFICADO
        FileLinePool = ''
        P1.write_lines_Code('CODE_' + FileLinePool+ 'Pool_' + METODO + '_' +
HojasRed[esc] + '_' + HojasMatriz[mat] + '.txt')
```