Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

MÁSTER EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

Aplicación de búsqueda tabú a un problema de sincronización semafórica

Realizado por José Javier Guerrero Herruzo

Tutorado por D. Jesús Muñuzuri Sanz

Sevilla, Julio 2016

Índice

Lista de figuras

1. INTRODUCCIÓN

1.1. Motivación del trabajo

Hoy en día, debido al constante crecimiento del área metropolitana de la ciudad de Sevilla y al continuo incremento del volumen de vehículos en la ciudad, se torna una necesidad fundamental el disponer de una gran cantidad de semáforos para controlar el tráfico vehicular de la mejor forma posible.

Sevilla se sitúa tercera en el ranking de ciudades más congestionadas de España, además dicha congestión ha crecido en el último año debido a la reducción del precio del petróleo que ha provocado un aumento del transporte mediante vehículo privado.

Las ciudades más congestionadas de España se pueden ver en la figura 1.1, donde además se puede observar en la última columna una tendencia con respecto al año anterior, para comprobar si las medidas que se estén implantando en dicha ciudad influyen o no en una mejora en la congestión.

Puesto	Ciudad	Horas perdidas en 2015	Variación (en horas)
1	Barcelona	28	+2.71
2	Madrid	22	-0.30
3	Sevilla	18	+0.35
4	Bilbao	16	-0.11
5	Valencia	12	-0.36
6	Zaragoza	11	-0.10

Figura 3.1: Ciudades más congestionadas de España en 2015. Variación con respecto al año anterior. (Fuente: INRIX Traffic Scorecard)

Una correcta programación conjuntamente sincronizada de los semáforos, lo cual no es siempre sencillo, debe ser realizada por los ingenieros civiles en los centros de control de tráfico. Es imprescindible realizar esta programación de forma eficiente, para evitar todos los problemas relacionados con la congestión del tráfico rodado, como pueden ser la seguridad, el ahorro energético, ahorro en el tiempo de trabajo, etc.

Toda esta serie de problemas asociados a el gran flujo de vehículos que circulan por Sevilla actualmente se pueden reducir mediante una correcta coordinación de los semáforos, de modo que se minimice lo máximo posible la congestión en nuestra ciudad. Aunque ya se usa una sincronización de los semáforos para el viario principal, aún queda bastante por mejorar para hacer el conjunto de todos los semáforos de la ciudad más eficientes.

1.2. Objetivos

El objetivo principal de este trabajo será determinar la sincronización más eficiente de los semáforos de la ciudad, concretamente de los que forman parte del viario que soporta un mayor flujo de vehículos, que será el más propenso a congestionarse.

De este modo, se pretende que los semáforos estén mejor interconectados, gracias a que se trabajará con una red de toda la ciudad de Sevilla y se analizará el viario en su conjunto.

Sin duda, dicha optimización podría tener un importante impacto en el ciudadano que emplea su vehículo propio para desplazarse, así como podría reducir los niveles de contaminación asociados al gran volumen de vehículos que se mueven por nuestra ciudad.

Las soluciones obtenidas desde el algoritmo se deberán comprobar con las soluciones de la figura 1.2., que es la última versión disponible por parte del ayuntamiento de Sevilla del plano de coordinación semafórica, aunque actualmente el sentido de algunas calles es diferente al de la foto.

Figura 1.2: Plano de Coordinación semafórica de la ciudad de Sevilla (Fuente: Ayuntamiento de Sevilla)

Aunque deben salir soluciones similares, será interesante el proponer mejoras para dicha sincronización, que vendrán dadas por los resultados que se obtengan del algoritmo en la red de la ciudad de Sevilla.

2. LA REGULARIZACIÓN SEMAFÓRICA

Los semáforos son una de las herramientas más importantes que se tienen para gestionar y regular el tráfico, sobre todo en las grandes ciudades cuando dos o más vías, de uno o dos sentidos de circulación, se cruzan a un mismo nivel y con una intensidad de tráfico demasiado alta para admitir una regulación de preferencia de paso.

La correcta instalación de estos elementos permite, tanto mejorar el flujo de vehículos como incrementar la seguridad de todos los usuarios en nuestras carreteras, por lo que juegan un papel fundamental en la regulación del tráfico.

Debido a que las mejoras de las infraestructuras físicas son costosas y no siempre abordables, los investigadores están de acuerdo en que manteniendo una planificación adecuada y eficiente de los semáforos se puede llegar a una mejora de la circulación de vehículos en las ciudades.

2.1. Definición

La regularización semafórica viene a resolver las detenciones y demoras relacionadas con los vehículos que acceden a una determinada intersección. Para ello, los semáforos permiten el paso alternativo de los vehículos desde los distintos accesos que confluyen a la intersección durante un determinado periodo de tiempo. Este paso se concede ordenadamente y es repetido de forma cíclica. Cuando dichos movimientos son "paralelos" o "divergentes", pueden realizarse simultáneamente aunque sean de diferentes accesos. Sin embargo, no pueden realizarse al mismo tiempo cuando son "convergentes" o se "cruzan".

Aunque en una intersección se suele regular exclusivamente la circulación de vehículos, de forma general y principalmente en zonas urbanas, se suelen regular conjuntamente la circulación de vehículos y peatones.

Principalmente, la regularización semafórica va a tener como campo de aplicación a las zonas urbanas, aunque también se recurre a ella en carreteras, especialmente en intersecciones cercanas a las poblaciones, en travesías y intersecciones peligrosas.

2.2. Sincronización entre cruces

Se puede entender la técnica de regulación como un fenómeno aislado, es decir, como una instalación autónoma, sin relación con otras instalaciones semafóricas o sin importar lo que ocurre aguas arriba o aguas debajo de la intersección. Sin embargo, pueden darse ciertas interferencias no deseadas entre ellos que es preciso solucionar.

Cuando varias intersecciones semafóricas están próximas entre sí, se ha demostrado que es conveniente sincronizar los reguladores con el objetivo de coordinar el encendido y el apagado de los semáforos. Esto tiene como objetivo que un vehículo que circule en un sentido concreto encuentre el menor número de semáforos en rojo y que el tiempo de parada sea mínimo.

Se pueden encontrar dos acciones en la coordinación: la primera, que se denomina

"sincronización", consiste en el hecho físico de unir dos o más reguladores a través de un cable que transporta una señal eléctrica denominada "sincronismo" y que hacen que los reguladores funcionen al unísono de acuerdo a un determinado ciclo; la segunda acción es la propia coordinación que pertenece al plan de ingeniería del tráfico y que determinan como deben funcionar los semáforos para conseguir un efecto práctico.

2.2.1. Sincronización por medio de los mismos reguladores

Los equipos estarán interconectados mediante un cable por el que se transmite la señal que sincroniza la red. Uno de los reguladores será el emisor y enviará la señal periódicamente al resto de reguladores que serán todos los receptores, denominada señal de sincronismo.

Esta señal de sincronismo puede ser diferente de un regulador a otro, especialmente teniendo en cuenta la variedad de modelos y tecnologías que se encuentran actualmente. En la actualidad aún se pueden encontrar señales de sincronismo a 220 V (corriente alterna) para mandos electromecánicos y electrónicos; señales de 12 o 24 V (corriente continua) y, posiblemente, señales mucho más complejas y sofisticadas para reguladores con microprocesador.

Actualmente, con el desarrollo de la electrónica se ha extendido la sincronización de reguladores sin cables. De hecho se pueden considerar dos métodos: por un lado, el sincronismo vía radio, cuyas experiencias no dieron el resultado esperado, y el sincronismo por medio de reloj. Este último se puede subdividir en dos tipos: el de reloj interno, basado en la frecuencia de la red o en un cristal de cuarzo, y el de reloj de tiempo real, que si se sincroniza con una señal astronómica exterior puede ser muy eficaz.

2.2.2. Sincronización a través de un equipo diferente a los reguladores

Este equipo puede ser un generador de sincronismo que es el equipo más elemental y cuya función es exclusivamente la generación de dicha señal, o una central de tráfico, si bien ésta cumple otras funciones más complejas. En este caso, los reguladores son todos receptores, ya que la señal de sincronismo será enviada por este último equipo.

La coordinación de los semáforos debe estar prevista en el Plan de Tráfico. Por tanto es una parte de la ingeniería de trafico con que se ha estudiado la red, y que se implanta en los reguladores tal y como se ha indicado anteriormente.

2.3. Capacidad de las intersecciones reguladas por semáforos

La capacidad de las intersecciones se define para un grupo de carriles como la intensidad de circulación máxima a través de la intersección, en las condiciones de tráfico, calzada y semaforización.

- Condiciones de circulación: abarcan los volúmenes de cada acceso, la distribución de los vehículos en cada movimiento, la distribución por tipo de vehículo en cada movimiento, la situación y utilización de las paradas de autobús en la zona de la intersección, flujos peatonales de cruce y movimientos de estacionamiento.
- Condiciones de calzada: comprende la geometría básica de la intersección incluyendo el número y anchura de carriles, pendientes de la rasante y las asignaciones de uso de los carriles.
- Condiciones de semaforización: incluyen una total definición de las fases semafóricas, el reglaje, el tipo de control y una total evaluación de la progresión semafórica en cada acceso.

En el "Manual de Capacidad de Carreteras" aparecen los procedimientos para evaluar y determinar la capacidad de un grupo de carriles. Esto se hace para aislar los carriles que presten servicio a un movimiento o serie de movimientos en particular. En este manual se establece un procedimiento con las directrices para determinar cuándo y cómo se deben determinar los grupos de carriles de un acceso.

La capacidad en las intersecciones reguladas por semáforos, se basa en los conceptos de saturación e intensidad de saturación, que se define como la máxima intensidad de circulación que se puede circular por un grupo de carriles dados, en las actuales condiciones de circulación y calzada.

Se define el índice de saturación de un grupo de carriles como la relación entre la intensidad de circulación real o provisional del grupo de carriles y la intensidad de circulación.

2.3.1. Niveles de servicio para intersecciones reguladas

El nivel de servicio se va a definir en términos de demora, para medir así la molestia, frustración consumo de combustible y tiempo perdido por el conductor. Concretamente, en la siguiente tabla de la figura 3.1 se puede ver el nivel de servicio establecido mediante la demora media de parada para un período de análisis de 15 min.

NIVEL DE SERVICIO	DEMORA POR PARADA POR VEHÏCULO (seg.)
A	≤ 5,0
B	> 5,0 y ≤ 15,0
C	> 15,0 y ≤ 25,0
D	> 25,0 y ≤ 40,0
E	> 40,0 y ≤ 60,0
F	> 60,0

Figura 4.1: Criterios de nivel de servicio para intersecciones reguladas por semáforos (Fuente: (Dirección general de tráfico. Oposiciones. Temario completo comprimido. Parte 3. Gestión Técnica del Tráfico., 2014))

- Nivel de servicio A: hay un avance extremadamente favorable y la mayoría de los vehículos llegan durante la fase verde, y no se detienen para nada. Los ciclos de corta duración también puede contribuir a una escasa demora.
- Nivel de servicio B: puede ocurrir por una buena progresión, por ciclos cortos o por una combinación de ellas. Hay una mayor detención que en el nivel A, y por tanto, mayores niveles de demora media.
- Nivel de servicio C: pueden deberse a una progresión de mediana calidad, ciclos más prolongados, o ambas circunstancias. Es posible que se empiece a producir alguna falta de capacidad en algún ciclo. Hay un número significativo de vehículos que se detienen aunque muchos atraviesan la intersección sin pararse.
- Nivel de servicio D: empieza a ser notable la influencia de la congestión. Pueden aparecer demoras más prolongadas debido a alguna combinación de progresión desfavorable, duraciones de ciclo prolongadas o elevadas relaciones de

Intensidad/Capacidad. La proporción de vehículos que no se detienen disminuye, así como aparecen faltas de capacidad en ciclos individuales.

- Nivel de servicio E: muchos organismos lo consideran el límite de demora aceptable. Los altos valores de demora indican un avance lento, duraciones de ciclo largas y altas relaciones Intensidad/Capacidad. Se presenta con frecuencia una insuficiencia de capacidad en algunos ciclos individuales.
- Nivel de servicio F: se considera inaceptable por la mayoría de conductores y suele ocurrir cuando se superan las capacidades de las intersecciones. Igualmente, las causas fundamentales pueden ser una progresión deficiente y duraciones de ciclo prolongadas.

2.4. Métodos de coordinación de intersecciones reguladas

En este proyecto se va a perseguir una programación del encendido de las luces de los semáforos de tal forma que los vehículos puedan atravesar una o varias vías, de un extremo a otro, a una velocidad constante y sin detenerse. Para conseguir esto, es necesario determinar el desfase entre el instante de encendido de las luces verdes de los diferentes cruces, que vendrá dado en función de la velocidad deseada y de la distancia entre dichos cruces.

Por tanto, la regularización va a estar muy relacionada con la determinación de ondas verdes, que va a consistir en coordinar una serie de semáforos para permitir el flujo continuo del tráfico sobre varias intersecciones en una misma dirección.

Cualquier usuario que se mueva sobre una onda verde a una velocidad establecida, no tendrá que detenerse en las intersecciones. Esto permite mayores volúmenes de tráfico y se reduce el ruido y el consumo de energía debido al menor uso de los frenos y el acelerador.

Volviendo al desfase, se entenderá como la diferencia entre el momento de referencia y el encendido de una determinada luz verde de un cruce. Para mantener este desfase constante a lo largo del tiempo, es necesario que en todas las intersecciones la duración del ciclo sea la misma.

Para determinar el desfase entre intersecciones, es necesario un estudio de las condiciones existentes. Dicho estudio podría llevarse a cabo mediante técnicas que se pueden agrupar en dos tendencias con bastantes diferencias entre ellas:

1. Métodos basados en criterios geométricos o de banda pasante (ondas verdes). Son aptos para coordinaciones con índice de saturación bajos y muy eficaces en vías con un solo sentido de circulación.

   Otro aspecto positivo es que el usuario aprecia directamente las ventajas de esta coordinación.

2. Métodos basados en optimizar las variables de tráfico (demoras, paradas, colas, etc.), para cuyo cálculo se emplean complejos algoritmos matemáticos. Están indicados en sistemas cercanos a la saturación o cuando se tienen varios ejes lineales que se cruzan formando una malla.

   En este segundo método, no es tan apreciable para el usuario las ventajas de la coordinación, ya que se centra en optimizar el funcionamiento del conjunto de todas las intersecciones para alcanzar la mayor capacidad posible, lo cual suele conllevar una serie de pequeñas molestias puntuales.

Para la mayor parte de los casos (como pueden ser travesías o arterias principales) es preferible usar métodos geométricos. Estos métodos se van a explicar brevemente en este capítulo, ya que permitirán al usuario percibir las ventajas de la coordinación y están muy relacionados con las ondas verdes, cuya obtención es uno de los objetivos principales de este trabajo.

Sin embargo el método que se va a seguir no es un método geométrico, sino una obtención de un árbol a partir de una red donde cada nodo es una intersección. Dicho árbol dirigido será la solución de nuestro problema, ya que los arcos serán las ondas verdes que se tendrán en la ciudad.

2.5. Problema de sincronización semafórica

En este apartado se va a describir más ampliamente el problema que se va a resolver en este trabajo, definiendo no solo el problema, sino especificando las características que van a tener las soluciones y qué es lo que se va a obtener de ellas.

En primer lugar, el problema a resolver es obtener un árbol de una red que interconecte todos los nodos de una ciudad. La red tiene que incluir el número total de arcos y nodos que sean necesarios para definir el viario principal de la ciudad, el cual pueda tener una mayor congestión durante la operatividad diaria.

Un árbol de una red tiene las siguientes características:

- **Todos los nodos tienen que estar conectados**
- **No se pueden producir ciclos**

Por tanto, para conseguir el cumplimiento de estas dos características, si el número total de nodos de la red es N, el **número de arcos en la solución debe ser N-1**.

Estas características definen el por qué la solución a este problema tendrá una forma de árbol, ya que si se tiene una red en la que todos los nodos están conectados y además no hay ciclos, si se le da una dirección al árbol, se puede realizar una sincronización semafórica en el sentido del flujo que se le haya dado a los arcos del árbol.

La coordinación de los semáforos se haría según el sentido especificado en las flechas y estos se irían encendiendo uno tras otro sin ningún problema, ya que se ha especificado que no haya ciclos en la solución, y todos los nodos van a estar interconectados. Quedaría definido así el encendido de todos los semáforos de la red a considerar, ya que todas las intersecciones están siendo consideradas.

Dada la complejidad de obtener un árbol dirigido a partir de una red, se ha optado por obtener un árbol no dirigido donde los sentidos se van a definir una vez resuelto el problema. Por lo tanto, en la red no se tendrá en cuenta los sentidos de los arcos en un principio, ya que el algoritmo solo tendrá en cuenta los dos nodos que forman el arco.

En el caso de que la red tenga vías con un solo sentido de circulación o vías que en uno de los dos sentidos el flujo es mayor que en el otro, sí que es importante la dirección de los arcos, por lo que el procedimiento seguido será almacenar dicha información durante todo el desarrollo del algoritmo, hasta que se obtenga una solución sin arcos dirigidos.

Una vez se tiene esa solución, a los arcos que solo tienen un sentido se les da dicho sentido, lo mismo que al resto se les da el sentido que tenga un mayor flujo siempre que los sentidos proporcionados sean coherentes. Así se obtendrá el árbol final solución al problema, aunque nuestro objetivo será en un primer momento obtener el árbol sin dirigir.

Para las vías de un único sentido y para todas las vías, se ha escogido como flujo del arco el mayor flujo de los dos sentidos de circulación que se tienen en dicho arco. De ahí la importancia de almacenar el sentido de circulación, ya que en el caso de que en uno de los dos sentidos el flujo sea muy diferente a el sentido contrario, se debe conocer cuál de los dos sentidos soportaba el mayor flujo.

Por último, el procedimiento seguido en este trabajo para la resolución de dicho problema ha sido el siguiente:

- Se resuelve en primer lugar una red de prueba, que será una red bastante más simple que la que será necesario resolver para una ciudad, y en la que se puede tener una primera aproximación del algoritmo.
- Una vez se tienen soluciones en forma de árbol que maximizan el flujo de dicha red, será necesario aplicar dicho algoritmo a la red del problema a considerar, que será bastante más compleja. La formación de dicha red se expone en el siguiente capítulo de este trabajo.

3. RED DE LA CIUDAD DE SEVILLA

En este capítulo se va a exponer el procedimiento que se ha realizado para la obtención de la red del viario principal de la ciudad de Sevilla, donde aparecen las intersecciones más importantes y en las que se requiere de una buena regulación para agilizar el paso de vehículos y calcular cuáles serán las ondas verdes necesarias, para que las vías que tengan un mayor flujo de vehículos tengan una menor congestión.

La red que sale de aquí debe contener como se ha dicho las vías más importantes de la ciudad, además de tener una fiabilidad suficiente con respecto a los flujos, de modo que una vez resuelta la red se pueda saber cómo regular los semáforos para alcanzar un nivel de servicio lo más favorable posible.

Se va a exponer como se ha obtenido la red explicando todas las operaciones que se han realizado y en el orden cronológico de estas, para que se pueda ver claramente el procedimiento seguido.

3.1. Datos de partida

Los datos que se han usado han sido proporcionados por el tutor de este proyecto, dada la incapacidad de obtener datos de flujo de las vías de Sevilla por otros medios, por ejemplo a través de la web de tráfico del ayuntamiento de la ciudad, ya que no se ha recibido respuesta a pesar de requerir dicha información en varias ocasiones.

La única información que se ha podido obtener de la web del ayuntamiento es las intensidades medias de tráfico por rutas para días laborales en 2015, que aparece en la siguiente web: http://trafico.sevilla.org/pdf/ResumenIMD2015.pdf. Sin embargo al no aparecer las intersecciones y no estar desglosado más ampliamente los flujos de vehículos, esta información no es de mucha ayuda para nuestro trabajo, aunque se debe tener en cuenta para comparar que nuestra solución cubre las vías que tienen más flujo.

Por tanto los datos se han obtenido de dos archivos Excel, cuyo contenido se va a explicar más ampliamente dado que es en lo que se basa el montaje de nuestra red:

- Lista de nodos: Se proporciona un listado con todas las intersecciones de la ciudad de Sevilla con tres columnas. La primera de ellas contendrá un número identificador de dicha intersección, mientras que otras las dos hacen referencia a las coordenadas X e Y de cada nodo.

  En los datos iniciales, aparecen una serie de nodos ficticios los cuales no son importantes para nuestro objetivo de sacar la red, ya que se necesitan nodos reales, así que son eliminados directamente.

- Lista de arcos: Se tienen todos los arcos del viario de Sevilla con un identificador para cada uno de ellos. Además viene facilitado tanto el número de carriles de los que consta dicha vía, como el flujo que pasa por ella, se supone que en vehículos por hora, que se será un dato fundamental para poder sacar el viario principal.

  Además de ello, para cada arco aparece el nodo origen y el nodo de destino, haciendo referencia a la lista de nodos. Dado que los arcos son direccionales, para las calles que tengan dos direcciones se tendrán dos arcos, uno para cada sentido.

3.2. Eliminación de elementos ficticios

En primer lugar, dado que hay una serie de nodos en los datos que no son reales, estos no deben aparecer en nuestra red, por lo que todos ellos son eliminados. De todas formas dichos nodos tienen un flujo de vehículos nulo, por lo que en caso de no ser eliminados ahora, serían eliminados en los pasos siguientes.

No hay que olvidar que no solo hay que eliminar dichos nodos, sino que habrá también que suprimir los arcos que salgan o entren a dichos nodos. Por tanto se va a ir reduciendo en paralelo tanto la lista de nodos como la lista de arcos.

3.3. Cálculo del viario principal

Se va a explicar, paso a paso, como se han gestionado los datos de partida para llegar a una red con el viario principal de la ciudad de Sevilla.

Para el cálculos de las vías más importantes, en primer lugar se ha hecho una preselección con los nodos que forman parte de los arcos con un mayor flujo, es decir, con los nodos por los que circulan un mayor número de vehículos.

Después se eliminarán los nodos que no aporten nada a la red y se llevará una segunda selección para obtener los nodos que formarán parte de nuestra red, sobre la que se realizarán los posteriores cálculos para obtener las ondas verdes.

3.3.1. Preselección de nodos con un mayor flujo

Se ha considerado viario principal de la ciudad de Sevilla a aquellas vías cuyo flujo se mayor que 1000 vehículos por hora. Dado que se tiene un listado con todos los arcos y el flujo de cada uno de ellos, se filtra con el Excel para que solo queden los arcos con más flujo.

Una vez que se tienen esos arcos, que serán los que formarán nuestra red, hay que seleccionar los nodos que conectan a dichos arcos. El procedimiento seguido es el siguiente:

1. Poner en una lista todos los nodos que conectan a los arcos que se han seleccionado, para obtener así una primera preselección de los nodos.
2. Eliminar duplicados en dicha lista de nodos, para que no haya nodos repetidos y aparezcan todos los que se necesitan.
3. Como se tiene una tabla con todos los nodos y sus coordenadas por un lado, y la lista de los nodos que se ha preseleccionado se emplea la función BUSCARV para sacar los datos de las coordenadas para los nodos que han sido preseleccionados, lo cual nos permite tener los datos de las coordenadas instantáneamente.
4. Se representan dichos nodos para poder observar que el resultado de la preselección sea coherente y que contiene a todas las vías importantes de la ciudad.

En primer lugar, se va a exponer la representación de dicha preselección de nodos (Figura 3.1) donde aparecerán los nodos distribuidos de acuerdo a un sistemas de coordenadas dado por los datos de la lista de nodos inicial.

Seguidamente se expone en la figura 3.2 un plano con las vías principales de la ciudad de Sevilla para poder comparar ambos gráficos y ver las calles con más intersecciones son las que forman parte del viario principal, aunque cueste un poco de identificar al principio.

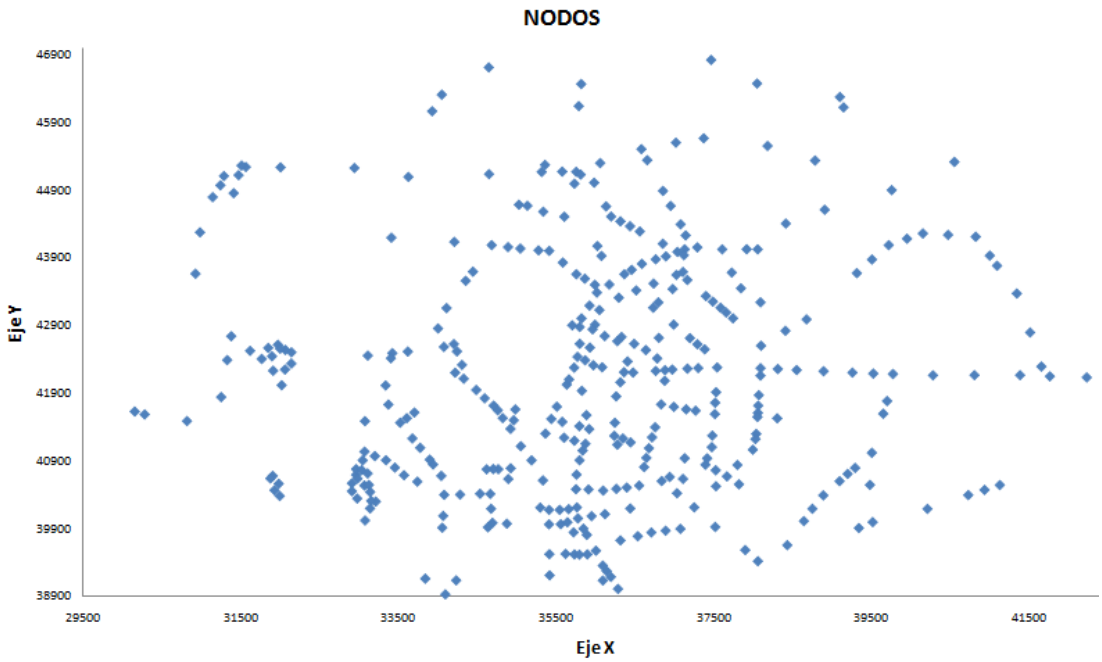

Figura 3.1: Preselección de nodos para la red de la ciudad de Sevilla (Fuente: elaboración propia)

Figura 3.2: Viario principal de Sevilla (Fuente: http://www.viamichelin.es/web/Mapas-Planos)

3.3.2. Eliminación de nodos exteriores

En el plano anterior con los nodos iniciales, hay muchos nodos que no pertenecen a la ciudad de Sevilla, sino que son parte del área metropolitana y que no van a ser de nuestro interés ya que se deberían de calcular aparte, por lo que se va a proceder a su eliminación.

Por otro lado están los nodos que forman partes de carreteras que entran en la red de Sevilla, y que siguen todos una línea y no forman parte del conjunto de la red, por lo tanto estos nodos no formarán parte de nuestra red ya que no tiene sentido añadirlos.

Se llega así a la representación de nodos que aparece en la figura 3.3, que es igual a la anterior en la que se han eliminado los nodos antes descritos.

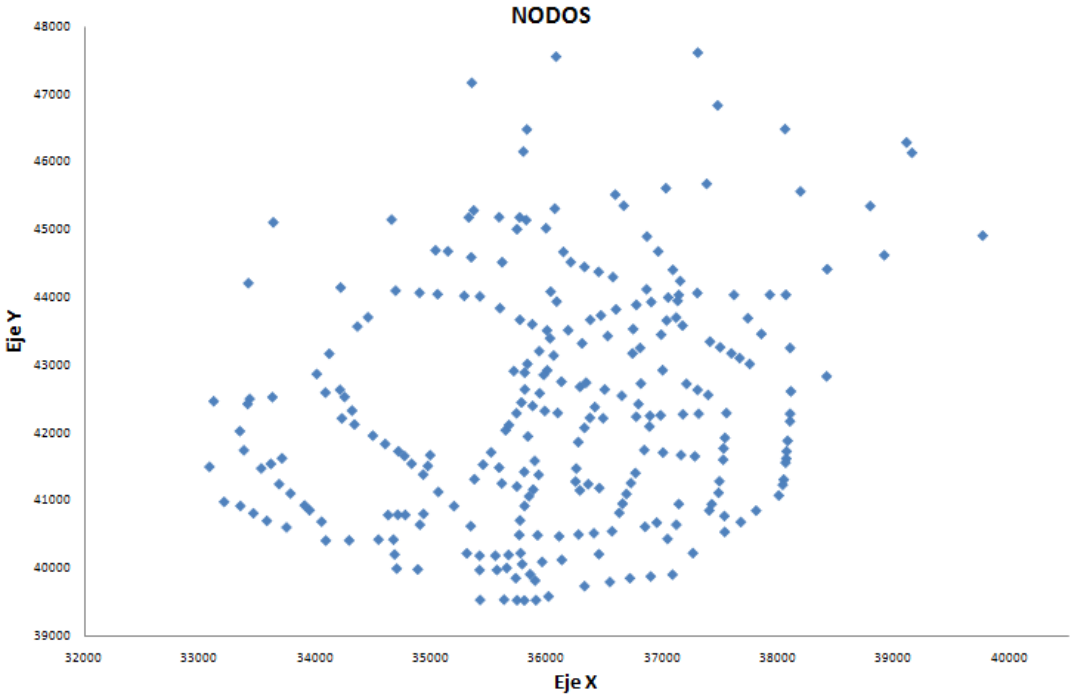

Figura 3.3: Preselección de nodos para la red de la ciudad de Sevilla sin nodos exteriores (Fuente: elaboración propia)

3.3.3. Selección de nodos finales

En este apartado se va a exponer el procedimiento para, una vez se ha llegado a la lista de nodos que aparecen representados en la figura 5, quedarse con los nodos más importantes de dicha figura, y que conectan las vías principales de la ciudad.

Para ello, será necesario la eliminación tanto de los nodos que hacen referencia a semáforos situados en paso de peatones, como los nodos que separan a una vía pequeña de una vía principal. Todos estos nodos no son importantes y habrá que extraerlos de nuestro plano.

En cambio, los nodos importantes y que constituirán la red, serán aquellos que interconecten las principales vías. En consecuencia, será necesario conocer cuáles serán las vías principales y ello se hará comparando cada uno de nuestros nodos con un mapa real de las calles de la ciudad. Así se puede comprobar qué nodos son los que se tendrán que añadir sin falta dado que conectan vías relevantes.

Puesto que por un lado se tiene la representación de los nodos con un mayor flujo y por otro el plano de la ciudad, será necesario un análisis de cada uno de los 266 nodos que se habían

preseleccionado, para conocer cuáles de estos nodos son los que hacen referencia a los cruces más importantes y conectan a las vías principales.

El procedimiento seguido es el siguiente:

1. Se identifican todos los 266 nodos que se tienen preseleccionados (cada uno tiene un número asociado).
2. Una vez identificados, para cada una de las calles, se observa qué nodo conecta con el resto, ya que pueden haber nodos cercanos pero solo uno es el principal que está conectado con otros principales.

   Para ello será necesario examinar cuidadosamente la lista de arcos, ya que los nodos principales tienen que ser los que formen parte de los arcos de mayor flujo y que a su vez sean intersecciones que conecten vías importantes
3. Se apuntan los nodos que son candidatos para estar en la red y se desechan nodos intermedios que no aportan a la red. Al mismo tiempo, se van apuntando los arcos que formarán nuestra red, una vez que se van conociendo los nodos candidatos.

   Para saber el flujo de los arcos finales, no solo será necesario examinar el arco que sale o entra al nodo elegido, sino que se toma como flujo de dicho arco el mayor de todos los arcos intermedios que haya entre dos nodos elegidos.
4. Se llega así a una lista de nodos que serán los que constituyan la red de Sevilla, y una serie de arcos que interconectan dichos nodos, que tendrán un flujo de vehículos dado por el mayor entre todos los flujos de vehículos que circulan entre dichos nodos por hora.

En la figura 3.4 se puede observar la representación de los nodos que han sido seleccionados mediante este procedimiento, y que serán los que constituyen nuestra red. En esta última selección se ha quedado la red en 65 nodos.

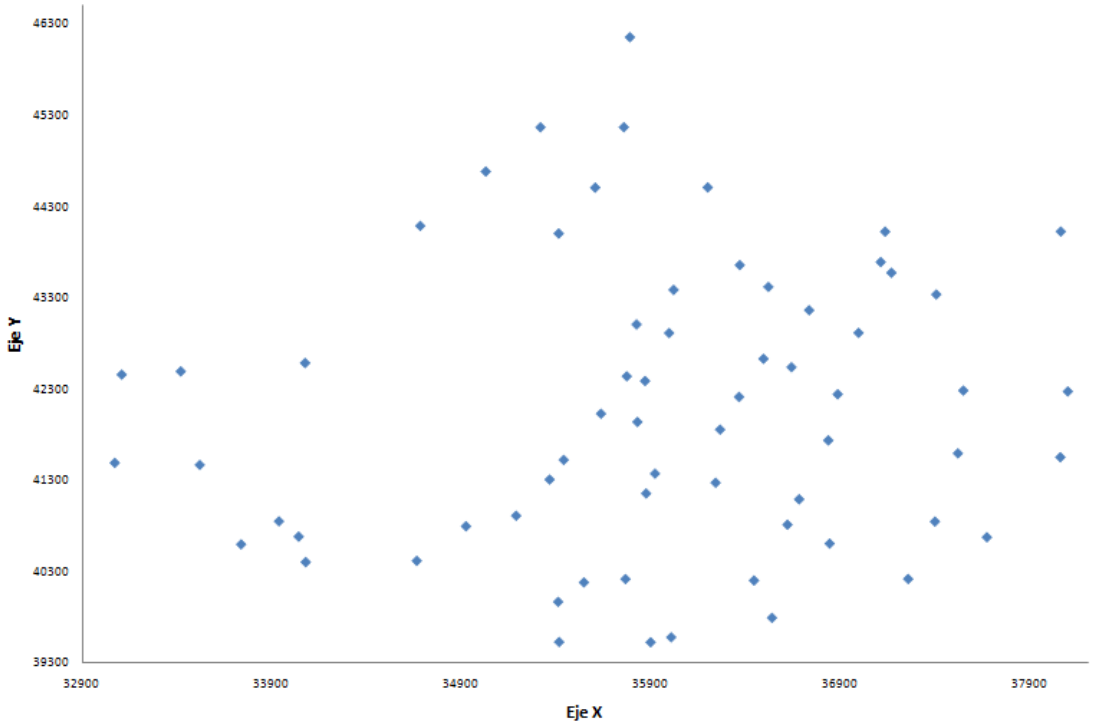

Figura 3.4: Selección de nodos finales (Fuente: elaboración propia)

Al igual que antes, para poder comprobar que dicha representación hace referencia al plano de Sevilla, se va a exponer en la figura 3.5 el mismo plano de la figura 4 pero más ampliado para poder observar a qué intersección hacen referencia los nodos.

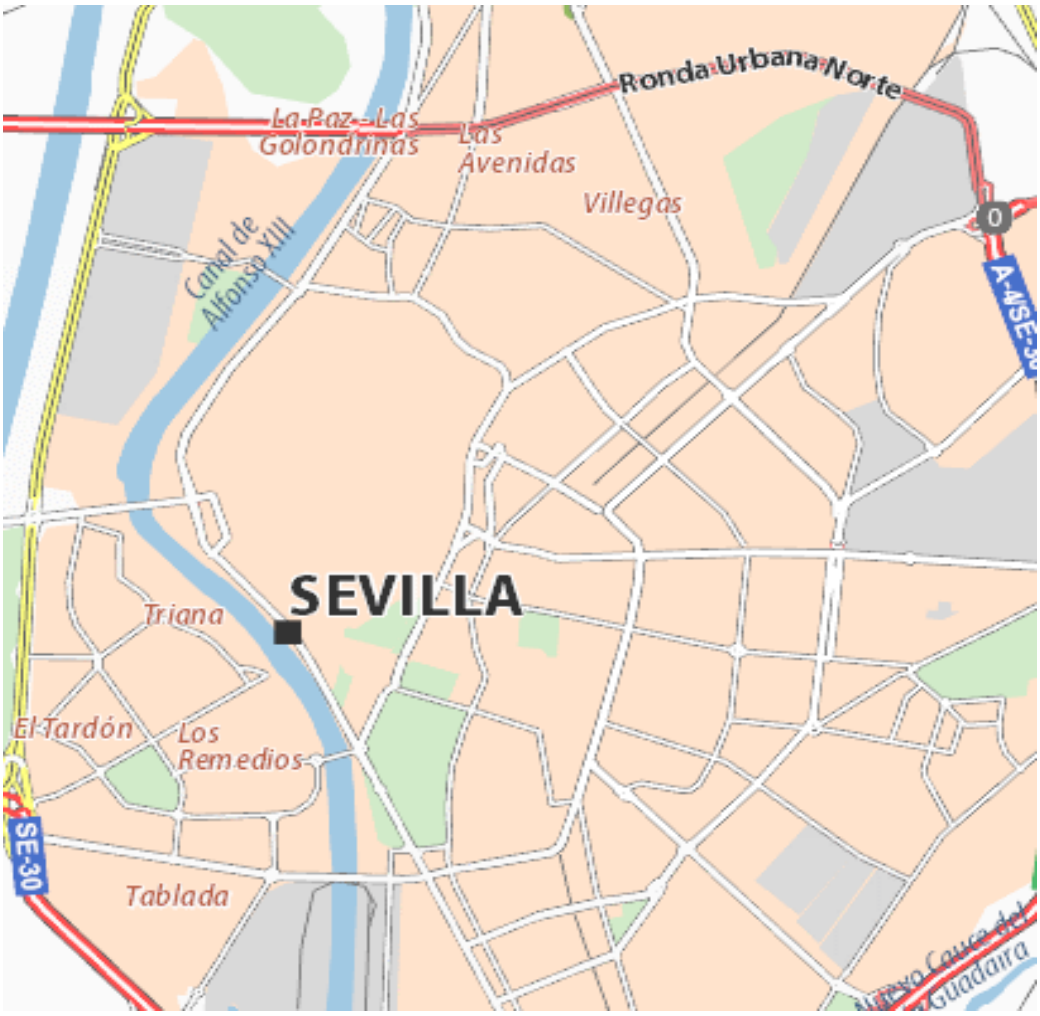

Figura 3.5: Viario principal de la ciudad de Sevilla eliminando las calles exteriores (Fuente: http://www.viamichelin.es/web/Mapas-Planos)

Para trabajar sobre la red, lo que se ha hecho es, una vez se iban sacando los nodos finales, se iban apuntando los arcos de los cuales se iban a constituir nuestra red. Por supuesto, aquí no podrían aparecer los nodos que han sido eliminados. Estos arcos se iban dibujando sobre un plano A3 con los nodos preseleccionados, para así saber que nodos ya están colocados y cuáles faltan por añadir.

Una vez completada la red en papel, se procedió a eliminar todos los nodos "sobrantes" para llegar así a la representación de la figura 6, con únicamente los nodos que son utilizados en la red.

Por tanto, si lo que se realizó a mano se pasa a ordenador, se llegaría a la red de la figura 8, donde aparecen todos los nodos seleccionados y también los arcos que la forman. Es a esta red, a la cual se le tendrá que aplicar algoritmos de resolución para obtener un árbol que incluya a las vías con un mayor flujo.

Dicha red aparece en la figura 3.6, donde se han colocado todos los arcos que podrían formar parte de la solución, pero que solo algunos de ellos los serán. Aquí si se puede comparar mucho mejor con el plano de la ciudad, que ya que cada arco va a hacer referencia a una de las calles y

dichos arcos serán más grandes o más pequeños dependiendo de si las vías que cruzan a esa calle o avenida son importantes.

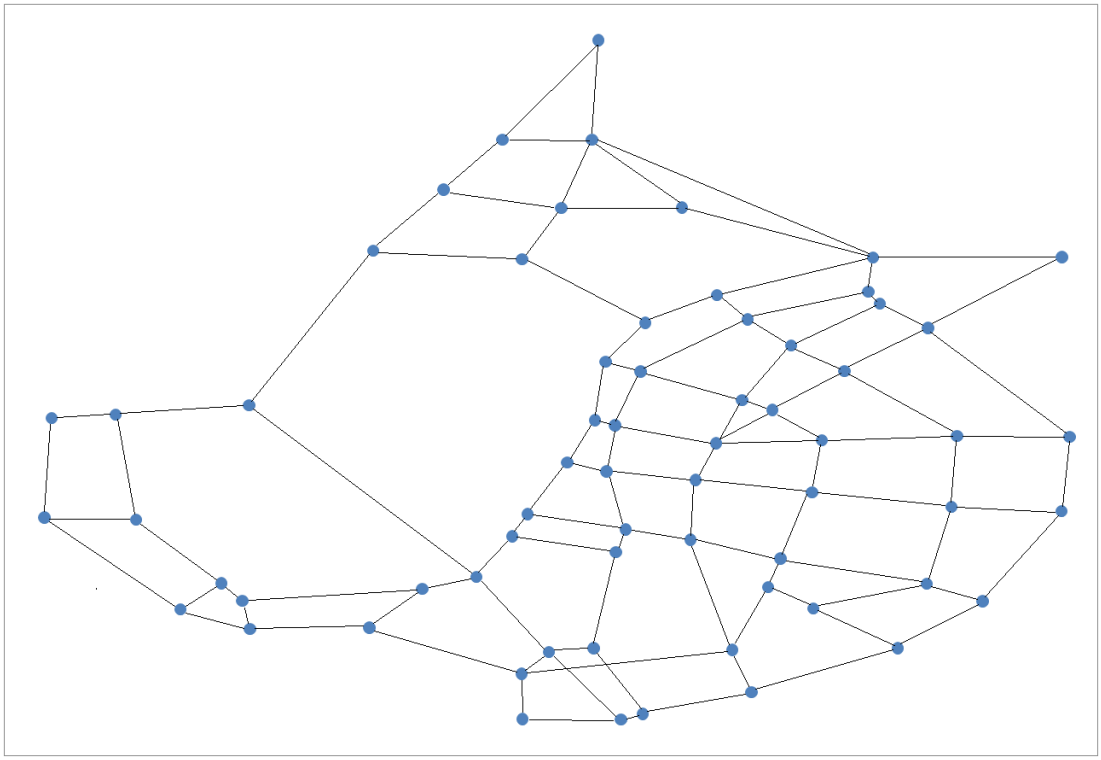

Figura 3.6: Red final del viario principal de la ciudad de Sevilla (Fuente: Elaboración propia)

Como ya se expuso anteriormente, la decisión para obtener estos arcos ha sido el nivel de flujo, siendo estos los que poseen un nivel más alto y que por tanto, una mejora sustancial en el encendido de los semáforos de dichas calles, puede repercutir en una mejora del tráfico en toda la ciudad de Sevilla.

Hay que tener en cuenta para que sentido de cada arco se tiene un mayor flujo, para que cuando se obtenga una solución a este problema, se direccione el arco en el sentido mayor del flujo. Además también hay que tener en cuenta las vías de un solo sentido ya que en el sentido opuesto el flujo sería nulo.

4. MÉTODOS DE RESOLUCIÓN

Los métodos de resolución para resolver este problema son muy diversos, dado que se necesita obtener un árbol de una red, lo cual lleva a un problema de optimización importante y para el que se han ido desarrollando diversos métodos de resolución a lo largo de las últimas décadas.

En la actualidad, los algoritmos para resolver estos problemas son muy variados en distintos aspectos, ya sea en el enfoque de optimización utilizado: local o global, o a qué clase de algoritmos pertenece, por ejemplo si están basados en programación lineal, son heurísticas clásicas o metaheurísticas.

En los subcapítulos posteriores, se procede a presentar los métodos más importantes de resolución de problemas de optimización. La mejora de estos métodos es continua, y existen gran cantidad de posibilidades cuando se trata de afrontar una familia de problemas. Los algoritmos tendrán distintas necesidades de recursos, por lo que su resolución provoca unas ventajas e inconvenientes respecto a los tiempos de computación que son necesarios.

Los algoritmos siguientes se van a dividir en tres familias, que serán los métodos exactos de resolución, los métodos Heurísticos y los métodos Metaheurísticos. Todos estos métodos podrían ser aplicados a la red para así obtener una solución, aunque dependiendo del tamaño será más conveniente optar por un método u otro.

4.1. Métodos exactos

Son aquellos que parten de una formulación como modelo de programación lineal (enteros) o similares, y que alcanzan una solución factible (entera) gracias a algoritmos que acotan el conjunto de soluciones factibles.

Debido a la complejidad que supone la resolución de los modelos matemáticos, solo los problemas con una red mediana o pequeña podrán ser resueltos mediante estos métodos, teniendo en cuenta que la resolución suele aplicarse a problemas relajados.

Generalmente, los métodos exactos no suelen ser aplicados en la realidad, ya que resultan inadecuados debido a que se tienen con estructuras de nodos muy amplias, cuya resolución conllevaría un coste computacional muy grande. Por tanto, solo suelen ser aplicados en ciertos casos con una pequeña cantidad de nodos.

Se van a destacar dos tipos de métodos exactos de resolución, que será el método de ramificación y poda, y el de ramificación y corte.

4.1.1. Branch and Bound (Ramificación y poda)

Consiste en dividir el conjunto de soluciones para facilitar la búsqueda de la solución óptima. Se trata de ramificar todas las soluciones enteras en subconjuntos separados, cada vez con un tamaño menor. Más tarde, se examina cada subconjunto y se halla su mejor solución.

El proceso que realiza es, una vez que se ha establecido una cota superior e inferior, el algoritmo va eliminando (de ahí la poda) la "rama" del conjunto de soluciones en la que no se encuentra la solución óptima, reduciendo así el espacio de búsqueda.

4.1.2. Branch and Cut (Ramificación y Corte)

Más que un método, se le conoce como una mezcla de ellos, ya que está basado en el *Branch and Bound*, pero además se le añade un método de planos de corte en los nodos. Se comienza eligiendo el primer nodo que es evaluado y se continúa decidiendo si se van a generar planos de corte o no. Una vez realizado esto, se aplican los métodos de ramificación y poda.

4.2. Métodos heurísticos

Un método heurístico es un algoritmo que permite obtener soluciones de buena calidad para un problema dado. Esto permite tener menores tiempos de ejecución, sin embargo, no asegura alcanzar el óptimo del problema, aunque se puede aproximar bastante.
Dependiendo de cómo acometa su labor, las heurísticas pueden clasificarse en:

- Constructivas: van elaborando soluciones factibles a medida que progresa el algoritmo.
- De mejora: trabajan sobre una solución ya factible.
- Técnicas de relajación: están asociados a la programación lineal entera. La más conocida es la técnica de Relajación Lagrangiana, que consiste en descomponer el modelo lineal entero, en un conjunto de restricciones difíciles y en otras más fáciles, y en cuanto a las primera las pasamos a la función objetivo multiplicándolo por una penalización, al igual que se hacía con el método de los multiplicadores de Lagrange. Sirve para obtener cotas al problema original y, en consecuencia, se acelera el proceso de resolución.

A continuación se procederá a explicar más detenidamente una de estas heurísticas que será las técnicas de búsqueda local.

4.2.1. Técnicas de búsqueda local

El objetivo será mejorar progresivamente la función objetivo partiendo de una solución inicial dada. El algoritmo debe encontrar una solución mejor a la anterior en cada iteración. El proceso global no finaliza hasta que no se puede encontrar un movimiento que mejore la solución actual.

Como su nombre indica se trata de un procedimiento de técnicas de búsqueda local, por lo que es de esperar que la solución obtenida sea un óptimo local. Para poder salir de dicho óptimo se debe permitir al problema poder moverse a soluciones de menor valor, para así alcanzar un óptimo distinto al anterior, teniendo así un algoritmo más complejo.

Lin y Kernighan proponen un algoritmo que se basa en llevar a cabo movimientos compuestos, en los que cada uno de los movimientos simples podrá mejorar o empeorar la función objetivo, sin embargo, el movimiento global debe ser de mejora, así no se pierde el control sobre el proceso de búsqueda.

Este último método mezcla diferentes movimientos que alteran la estructura de la solución, haciendo que el óptimo local se alcance de un modo lento y gradual, e introduciendo una componente de diversificación.

4.3. Métodos metaheurísticos

Los procedimientos metaheurísticos son métodos aproximados que se diseñan para resolver problemas complejos de optimización combinatoria, para los que los métodos heurísticos no son efectivos ni eficientes. Estos métodos combinan conceptos que derivan de la inteligencia artificial y de mecanismos estadísticos. Una definición de metaheurística dada por (Osman & Kelly, 1996) es:

"Es un proceso iterativo que guía a una heurística subordinada combinando inteligentemente diferentes conceptos con el objetivo de explorar y explotar los espacios de búsqueda, y usando estrategias de aprendizaje para estructurar la información y hallar soluciones cercanas al óptimo de un modo eficiente."

Estos métodos se pueden clasificar dependiendo del tipo de heurística al que haga referencia:

1. Métodos Constructivos: Se introducen elementos partiendo de una solución inicial vacía. El algoritmo más importante de este tipo es el GRASP.

2. Métodos Evolutivos: Se elaboran grupos de soluciones completas y se realiza una selección basándose en el valor de ciertos atributos. Posteriormente se combinan algunas soluciones que han sido seleccionadas y se reemplaza finalmente el grupo de soluciones. Un ejemplo de estos método es el Algoritmo Genético.

3. Métodos de Búsqueda: Estos métodos dan por hecho que debe existir una solución óptima, y ejecutan un procedimiento que si bien no alcanza esta solución, consigue una cercana. El riesgo del método es quedar atrapado en un óptimo local, por lo que habrá que crear procedimientos para salir de dicho óptimo. Estos métodos se dividen según la forma de salir del óptimo local, por lo que habrá tres maneras de proceder:

    a. Volver a una solución inicial distinta y comenzar de nuevo con el algoritmo. Por ejemplo, la Metaheurística multi-start.

    b. Variar la estructura del entorno de la solución cuando se alcanza el óptimo. Por ejemplo, Metaheurística de búsqueda de entornos variables.

    c. Permitir movimientos que empeoren la solución actual para así abandonar el óptimo local. Aquí se encuentra el Recocido Simulado y la Búsqueda Tabú.

4.3.1. GRASP (greedy randomized adaptative search procedure)

Este método se puede traducir como: "Procedimiento de búsqueda voraz aleatoria y adaptativa" y es una técnica encargada de resolver problemas de optimización combinatoria, en la que primero se intenta construir soluciones de alta calidad, para que posteriormente sean procesadas y obtener así otras soluciones de mayor calidad.

Se trata de un algoritmo de tipo iterativo, en los que en cada iteración se tiene una fase de construcción de la solución y otra en la que se optimiza la solución generada en la fase anterior, mediante algún algoritmo de búsqueda local. La estructura básica del algoritmo es la siguiente:

1. Construir una solución aleatoria mediante un procedimiento *Greedy*.

2. Aplicar una búsqueda local a la solución anterior para mejorarla.

3. Actualizar la mejor solución encontrada.

Se deben repetir los pasos anteriores hasta que se satisfaga un determinado criterio de parada.

Se establece una similitud con el problema de Programación Lineal, donde se construye una solución factible y después se aplica el algoritmo Simplex. La principal diferencia, es que en GRASP se le da gran importancia a la calidad de la solución generada.

4.3.2. Algoritmo Genético

Este método (Holland, 1975) el se trabaja sobre grupos de soluciones. En cada iteración, mediante una serie de procedimientos se transformas las soluciones en otras que serán una mezcla de las anteriores.

Este algoritmo es una analogía de la teoría de la evolución de las especies de Darwin, ya que las soluciones generadas conservan algunas características de las anteriores, dependen del grado de mutación que se tenga en cada iteración.

Un algoritmo genético puede presentar diversas variaciones, dependiendo de cómo se aplican los operadores genéticos y de cómo se realiza la selección y el reemplazo de los individuos para formar una nueva población. La estructura general será la siguiente:

1. Inicialización: Se genera una población inicial de forma aleatoria, que está constituida por un conjunto de cromosomas que representa las distintas soluciones del problema. Para los casos en los que no se generen aleatoriamente, se debe garantizar cierta diversidad en las soluciones, para así tener una representación de la mayor parte de la población y evitar una convergencia prematura.

2. Evaluación: Se le aplica la función objetivo a cada uno de los cromosomas para así conocer cómo de " buena" es la solución codificada.

3. Desarrollo del algoritmo: Mientras que no se alcance un número de iteraciones o no haya cambios en la población, se debe realizar lo siguiente:

    a. Selección: Se deben elegir los cromosomas a ser cruzados en la siguiente generación. Se elegirá a los cromosomas con mejor evaluación.

    b. Cruzamiento: Se opera sobre dos cromosomas a la vez para generar dos nuevos "descendientes" en los que se combinan características de los cromosomas "padres".

    c. Mutación: Modifica al azar una parte del cromosoma de los individuos, para así alcanzar zonas del espacio que no se podían alcanzar con la población actual.

    d. Reemplazo: una vez aplicado lo anterior, se selecciona los mejores individuos para conformar la siguiente población.

4.3.3. Simulated Annealing

Este método, que parte del algoritmo Hill-Climbing, es conocido como Recocido simulado, tiene su analogía con un proceso que se le realiza a los metales para mejorar su estructura cristalina. Este proceso consiste en calentar el material hasta altas temperaturas, e ir reduciendo lentamente la temperatura para permitir el movimiento de los átomos, hasta que dejen de moverse. Esto debe

realizarse adecuadamente, ya que un calentamiento excesivo destruye el orden alcanzado y uno insuficiente no permite el movimiento de los átomos.

A continuación se aplican estas ideas a los algoritmos de optimización mediante la siguiente analogía:

Una configuración cristalina de los átomos metálicos será considerada como una solución, la energía asociada a esa configuración es la longitud de la solución y su temperatura es una distancia.

En cada iteración del problema, el modelo evalúa algunos vecinos del estado actual y probabilísticamente decide entre una transición a un nuevo estado o permanecer en ese estado. Un estado se puede definir en función de la posición de los átomos del material en el momento actual; un desplazamiento de un átomo conllevaría a un estado vecino del primero. Se repite la comparación entre estados vecinos hasta que se encuentre un estado óptimo que minimice la energía del sistema o hasta que se alcance otras condiciones de parada.

La probabilidad de que una configuración sea elegida depende de su energía (calidad de la solución) y de la temperatura de la iteración:

- Si T→∞, todas las configuraciones tienen la misma posibilidad de ser elegidas, independientemente de su calidad.
- Si T→0, sólo las configuraciones de coste mínimo tienen una probabilidad no nula.

El sucesor aleatorio elegido se convierte en el siguiente si supone una pérdida de energía, ya que se está minimizando, aunque también podrá ser elegido un sucesor que provoque un incremento dependiendo de la temperatura.

El esquema de la temperatura, T, es una función que regula su descenso, de modo que se garantice una buena solución independientemente de la configuración inicial. Normalmente, se obtiene como una función del tiempo en la que t será un contador de tiempo y k una constante a determinar de forma experimental:

$$T = \frac{k}{\ln(1+t)}$$

Por tanto, los pequeños cambios en el proceso continúan hasta que no haya posibilidad de mejora. Esto es, se reduce paulatinamente la temperatura y van disminuyendo a la vez las probabilidades de permitir movimientos que empeoren la función objetivo, hasta que ya no se permite ninguno.

4.3.4. Búsqueda Tabú

La *Tabú Search* (Glover, Summer 1989) es una de las metaheurísticas más importantes desarrollada para solucionar problemas de optimización combinatoria. u característica fundamental es que emplea estructuras de memoria como base de su estrategia, almacenando toda la información posible para mejorar la solución.

El algoritmo búsqueda tabú, se basa en qué habrá una serie de movimientos prohibidos (movimientos tabú) dentro de la solución que se quiere mejorar. Estos movimientos tabú ayudan a que el algoritmo no permanezca en los óptimos locales y, así, buscar en una zona más amplia. Esto es lo que se conoce como "memoria" y en este caso, se trataría de una memoria a corto plazo.

Debido a todo esto, es considerada una búsqueda inteligente ya que va almacenando información

a medida que itera el algoritmo y luego va a actuando de acuerdo a esos datos almacenados. El método del algoritmo para salir del óptimo local puede ser mediante técnicas de diversificación, es decir, de acuerdo a otro tipo de memoria a largo plazo, se guía a la búsqueda a las zonas inexploradas.

La búsqueda tabú ha sido considerada una de las metaheurísticas que mejores resultados proporciona para una gran variedad de problemas, y por ello es la que emplearemos en este trabajo para resolver el problema de la regularización semafórica. Por tanto, este método será explicado más ampliamente en el siguiente capítulo, para ir conociendo todos sus detalles y procedimientos.

5. BÚSQUEDA TABÚ

La *tabú search* (TS) o búsqueda tabú (Glover, Summer 1989) es un procedimiento metaheurístico que se utiliza para guiar un algoritmo de búsqueda local a explorar el espacio de soluciones más allá de la simple optimalidad local.

El término tabú, se define coloquialmente como "una prohibición impuesta por costumbres sociales como una medida de protección" y también como "marcada como que constituye un riesgo". En nuestro caso, el riesgo que se trata de evitar es el de no poder salir del óptimo local.

El éxito en aplicaciones prácticas de optimización ha provocado un crecimiento acelerado de la búsqueda tabú, empleando únicamente la TS o mezclando la TS con otras heurísticas o procedimientos algorítmicos. Se muestra a continuación algunas áreas en las que se ha empleado búsqueda tabú, y problemas concretos de cada área:

- Secuenciación: flow-time cell manufacturing, scheduling, planificación de la fuerza laboral, etc.
- Diseño: CAD, diseño de redes de transporte, distribución en planta, etc.
- Tecnología: inversión sísmica, diseño estructural de ingeniería, distribución de energía eléctrica, construcción de estaciones espaciales, etc.
- Lógica e inteligencia artificial: lógica probabilística, satisfactibilidad máxima, integridad de datos, diseño de redes neuronales,etc.
- Telecomunicaciones: rutificación de llamadas, asignación de caminos, diseño de redes para servicios, etc.
- Rutificación: rutifiación de vehículos, time windows routing, rutifiación con flota fija, rutificación multi-modal, TSP, etc.
- Producción, inventarios e inversión: fabricación flexible, JIT, planificación de inventarios, MRP con restricciones de capacidad, etc.
- Optimización combinatoria general: programación 0-1, redes todo-nada, optimización de carga fija, etc.

En primer lugar, tenemos las siguientes características referidas a la memoria adaptativa de la búsqueda:

- Selectividad (añadiendo el olvido estratégico)
- Abstracción y descomposición (mediante memoria explícita y por atributos).
- Tiempo: recencia de eventos(tiempo transcurrido entre el último evento y la actualidad), frecuencia de eventos y diferenciación entre corto y largo plazo.
- Calidad e impacto: atracción relativa de elecciones alternativa y magnitud de cambios en relaciones de estructura o restricciones.

- Contexto con interdependencia regional, estructural o secuencial.

Por otro lado, las características que vienen dadas por una exploración sensible del espacio de búsqueda son:

- Imposición estratégica de limitaciones e inducciones: viene dada por las condiciones tabú y los niveles de aspiración.
- Enfoque concentrado en regiones prometedoras y buenas características de las soluciones: referido a lo que conoceremos como procesos de intensificación.
- Caracterización y exploración de nuevas regiones atrayentes: procesos de diversificación.
- Patrones de búsqueda no monótonos: se consigue mediante la oscilación estratégica.
- Integración y extensión de las soluciones: a través del reencadenamiento de trayectorias.

5.1. Definiciones básicas

En primer lugar, se van a exponer conceptos generales de los métodos de búsqueda. Se trata de un problema de optimización combinatoria en el que se tienen las siguientes características:

- X será el conjunto de soluciones del problema.
- $x \in X$ es una posible solución del problema.
- $f(x)$ será la función objetivo a optimizar.
- $N(x)$ es el entorno o vecindario de x, y significa el conjunto de posibles soluciones asociadas a una solución.
- Cada solución x' se puede obtener a partir de x mediante una operación llamada movimiento.

Se define el concepto de movimiento por ser esencial para comprender el método de búsqueda tabú. Se tiene un problema P de optimización combinatoria, con una función objetivo:

$$\text{Minimizar } f(x), x \in X$$

Para resolver P se puede establecer convenientemente una secuencia de movimientos que llevan de una solución tentativa a otra.

Se define un movimiento m como el mapeo definido en un subconjunto $X(m)$ de X, m: $X(m) \rightarrow X$.

Si el algoritmo está en el nodo i (la solución factible i), los movimientos posibles son aquellos arcos que unen el nodo i con alguno de sus nodos adyacentes (otras posibles soluciones que se derivan de la solución i).

A $x \in X$ está asociado el conjunto $N(x)$ (vecindario o entorno) el cual contiene los movimientos $m \in M$ que pueden ser aplicados a x, es decir:

$$N(x) = \{m \in M: x \in X(m)\}$$

Por otro lado, se tiene una solución inicial x_0 a partir de la cual se inicia la búsqueda. Esta solución puede ser obtenida de dos formas, mediante un procedimiento heurístico que me proporcione una buena solución inicial (sobre todo si se tiene conocimiento de donde puede haber una buena solución) o aleatoriamente, e ir mejorando la solución con el proceso de búsqueda.

El procedimiento de búsqueda va a depender de la vecindad escogida, ya que se parte de la solución inicial x_0 y una vez obtenida la vecindad $N(x_0)$, aplicamos un criterio de selección para obtener como siguiente solución alguna de la vecindad. Dicho criterio suele ser normalmente elegir la solución que tenga un mejor valor de la función objetivo. Este criterio es conocido como *greedy* y consiste en mejorar la solución lo máximo posible.

A continuación es necesario añadir nuevas características a nuestro problema para evitar que el algoritmo cicle respecto a un óptimo local que puede distar bastante del óptimo global. Es más, en problemas de alta dificultad como el que se trata en este trabajo, lo más lógico es que la solución obtenida con una búsqueda local diste mucho de la solución óptima.

5.2. Características fundamentales de la Búsqueda Tabú

La característica fundamental de la *tabú search*, que la hace diferente frente a otras metaheurísticas es el uso que hace de la memoria, la cual está basada en una lista tabú y un conjunto de datos que llevan a seleccionar el siguiente movimiento.

La determinada lista tabú, permite al algoritmo escoger soluciones que aunque empeoren la solución anterior, nos permita escapar de un punto en el que no se mejora, ya que no se puede alcanzar una mejor solución con movimientos simples. Por tanto, se realizan así una serie de movimientos que modifican la solución, hasta poder llegar a una solución que no tenga relación que reduzca el coste total.

5.2.1. Lista tabú (memoria basada en hechos recientes)

La lista tabú se define como una memoria a corto plazo, que contiene soluciones o atributos de soluciones que por algún motivo, no deben de ser escogidos. El algoritmo puede tener una o varias listas tabú, dependiendo de las necesidades de nuestro problema.

Hay una gran diversidad en cuanto a la forma de construir la búsqueda tabú, ya que dependiendo del problema puede interesar unos atributos u otros. Sin embargo, después de cierto tiempo, las soluciones que se almacenan en la liste pierden su estatus tabú, suponiendo que la búsqueda está lo suficientemente lejos como para que no se alcance de nuevo la misma solución.

Respecto al contenido de la búsqueda tabú, podemos encontrar:

- Soluciones que han sido visitadas recientemente.
- Movimientos realizados recientemente.
- Características o atributos de las soluciones visitadas.

La longitud de la lista tabú (conocida como *tabú tenure*) controla la memoria del proceso de búsqueda. Ésta definirá el número de iteraciones en las que una solución, un movimiento o un atributo permanece en la lista tabú. Con longitudes pequeñas, la búsqueda se concentra en pequeñas zonas del espacio de búsqueda. En cambio, con longitudes más grandes, se obliga a visitar una zona mucho más grande del espacio de búsqueda. Dependiendo del método que use la

búsqueda para añadir las soluciones a la lista, se definen dos tipos:

- Búsqueda tabú activa: La lista tabú es implementada como una memoria FIFO de tamaño l (*tabú tenure)*, por lo que al añadir la última solución visitada se extrae la solución que se incluyó hace l iteraciones (lo mismo ocurre en el caso de que la lista contenga movimientos o atributos).

- Búsqueda tabú reactiva: La longitud de la lista tabú varía de forma dinámica durante el proceso de búsqueda. Esto aporta robustez al algoritmo, por lo que se implementa en las versiones más avanzadas del *tabú search*.

  Esto se puede ver claramente, cuando, por ejemplo, se tiene dos tipos de atributos, en los que los de tipo 1 deben permanecer en la lista tabú el doble de iteraciones que los de tipo 2. Entonces, si un atributo de tipo 1, cuya *"tabú tenure"* es 10, es almacenado en la iteración 50, y un atributo tipo 2 se almacena en la iteración 52, se tendría que el atributo tipo 2 abandonaría la lista tabú antes (iteración 57) que el de tipo 1 (iteración 60), aunque haya sido añadido más tarde.

  Por ejemplo, si se está construyendo un árbol en un grafo, se parte de un nodo y se van agregando nodos (con su consiguiente arista), tras algunas iteraciones puede necesitarse retirar una arista (con su correspondiente nodo). Se tienen pues dos tipos de movimientos: agregar una arista (tipo 1) y retirar una arista (tipo 2). Cuando se agrega un par de nodos (x,y) al árbol, se hace una entrada a la lista tabú que penaliza su salida durante k_1 iteraciones. Análogamente cuando sale (u,v) se hace una entrada en la lista tabú que penaliza su entrada durante k_2 iteraciones. Como hay más aristas fuera del árbol que en él, se ve razonable implementar una estructura tabú que mantenga a una arista recientemente retirada con la condición tabú más tiempo (para dar oportunidad a otras aristas) que a una arista recientemente agregada.

5.2.2. Niveles de aspiración

El criterio de aspiración introduce un elemento importante de flexibilidad dentro de la búsqueda tabú. El estatus tabú de una solución (o un movimiento), puede ser ignorado si la solución cumple ciertas condiciones, entrando así en un "olvido estratégico" de la solución, para que dicha solución o atributo pueda salir de la lista tabú antes de que se cumpla su plazo.

Un ejemplo claro, se tiene cuando el algoritmo haya una solución mejor que todas las obtenidas anteriormente, pero es rechazada por la lista tabú. Mediante una correcta implementación del criterio de aspiración, se puede considerar admisible dicha solución aunque se considere un movimiento prohibido.

Las aspiraciones pueden ser de dos tipos:

- Aspiraciones de movimiento: cuando el criterio se satisface, se revoca la restricción que impone la lista tabú sobre dicho movimiento.

- Aspiraciones de atributo: cuando se cumple el criterio, el estatus tabú del atributo es revocado. En este último caso, dependiendo de sí la restricción tabú puede activarse por más de un atributo, el movimiento podrá o no cambiar su estatus tabú.

Los criterios de aspiración surgen por tres motivos:

1. Sí todos los posibles movimientos se consideran tabú, entonces se deberá escoger el

movimiento "menos tabú" de todos ellos. Esto es que sí se tienen varios movimientos restringidos, se escogerá el movimiento que le resten menos iteraciones para volver a ser permitido.

2. En el caso de que el valor de la función objetivo de una solución, sea mejor que el mejor valor obtenido hasta el momento. Por ejemplo, un movimiento m es aceptado sí, en un problema de minimización, el valor de la solución x resultante es menor que el menor coste obtenido hasta entonces.

3. Para los casos en los que un atributo de aspiración para alcanzar una solución x se satisface si la dirección para en x otorga una mejora y el actual movimiento también es de mejora. En ese caso, se toma a x como posible candidato.

5.2.3. Estrategias para la lista de candidatos

El carácter agresivo de TS se ve reforzado buscando el mejor movimiento disponible que pueda ser determinado con una cantidad apropiada de esfuerzo. Debe tenerse en cuenta que el significado "mejor" no está limitado solamente a la evaluación de la función objetivo (como se viene diciendo, las evaluaciones tabú pueden estar afectadas por penalizaciones e incentivos, que se determinan por la historia de la búsqueda). Las estrategias para la lista de candidatos se usan para restringir el número de soluciones examinadas en una iteración dada, para los casos en los que N(x) es grande o la evaluación de sus elementos es costosa.

Además, las evaluaciones tabú incluyen también periódicamente incentivos para estimular la elección de otros tipos de soluciones, como resultado de niveles de aspiración e influencias a largo plazo.

En el siguiente epígrafe, se describe como la búsqueda tabú saca provecho de la memoria para llevar a cabo estas funciones.

5.3. Uso de la memoria en *Tabú Search*

La búsqueda tabú se caracteriza porque utiliza una estrategia basada en el uso de estructuras de memoria para abandonar óptimos locales, en los que se puede caer al "moverse" de una solución a otra por el espacio de soluciones.

Las estructuras de memoria empleadas pueden ser de dos tipos:

- Explícita: Cuando las soluciones son almacenadas de manera completa, registrándose una serie de soluciones élite visitadas durante la búsqueda. Por ejemplo, las soluciones (x_1, x_5, x_7) son las x_i soluciones obtenidas en iteraciones anteriores. Una extensión de esta memoria, registra vecindarios altamente atractivos pero inexplorados de soluciones élite.

- Basada en atributos: Se guarda información acerca de ciertos atributos de las soluciones pasadas, para propósitos de orientación de la búsqueda. Este tipo de memoria registra información acerca de los atributos o características que cambian al moverse de una solución a otra. Por ejemplo en un grafo los atributos pueden consistir en nodos o arcos que son aumentados, eliminados o reposicionados por el mecanismo de movimientos.

La memoria, por lo tanto puede ser explícita o de atributos o ambas.

Resumiendo, la estructura de memoria explícita almacena soluciones que nos pueden llevar a un

óptimo local y la memoria de atributos tiene como propósito guiar la búsqueda.

Es importante considerar que los métodos basados en búsqueda local requieren de la exploración de un gran número de soluciones en poco tiempo, por ello es crítico el reducir al mínimo el esfuerzo computacional de las operaciones que se realizan a menudo, lo que se puede conseguir registrando los atributos de las soluciones en vez de éstas para orientar la búsqueda más rápidamente.

La estructura de la memoria en la metaheurística de búsqueda tabú opera en relación a cuatro dimensiones principales:

- Calidad
- Influencia
- Corto plazo (hechos recientes)
- Largo plazo (frecuencia de eventos)

5.3.1. Calidad de las soluciones

La calidad se refiere a la habilidad para diferenciar el mérito de las soluciones, identifica qué las hace tan buenas e incentiva la búsqueda para reforzar las acciones que conducen a una buena solución y desalienta aquellas que conducen a soluciones pobres.

La flexibilidad de la estructura de memoria permite que la búsqueda sea guiada en un contexto multiobjetivo, donde la bondad de una dirección de búsqueda particular puede estar determinada por más de una función.

El concepto de calidad en la búsqueda tabú es más amplio que el usado implícitamente en los métodos de optimización, en los cuales se considera que un movimiento es de mejor calidad que otro porque produce una mejor "mejora"(tal es el caso del descenso más rápido). Bajo el enfoque de búsqueda tabú un movimiento puede ser de mejor calidad si, por ejemplo, su frecuencia de ocurrencia en el pasado es baja o no ha ocurrido antes y nos permite explorar nuevas regiones. La definición de calidad de una solución es flexible y puede ser adaptada a la naturaleza del problema.

5.3.2. Influencia

La dimensión influencia, considera el impacto de las elecciones hechas durante la búsqueda. Mide el grado de cambio inducido en la estructura de la solución o factibilidad, no sólo en calidad sino también en estructura. En cierto modo, la calidad puede entenderse como una forma de influencia.

Registrar información acerca de las elecciones de un elemento de una solución particular incorpora un nivel adicional de aprendizaje, registra qué elementos o atributos generan ese impacto. Esta noción puede ser ilustrada para el problema de distribuir objetos desigualmente pesados entre cajas, donde el objetivo es dar a cada caja, tan aproximadamente como sea posible, el mismo peso. Un movimiento que transfiere un objeto muy pesado es un movimiento de alta influencia, cambia significativamente la estructura de la solución actual, sin embargo, otro que intercambia objetos de pesos similares entre dos cajas no introduce mayor influencia. Tal movimiento puede no mejorar la solución actual, ya que es relativamente buena.

Se privilegian los movimientos etiquetados como influyentes, pero en alguna iteración se puede optar por alguno poco influyente, pues éstos pueden ser tolerados si proporcionan mejores

valores. En tal punto, y en ausencia de movimientos de mejora, los criterios de aspiración cambian para dar a los movimientos influyentes un rango mayor y éstos puedan salir de la lista tabú antes del plazo establecido en su *tabú tenure*.

En el problema de distribuir objetos desigualmente pesados entre cajas, donde el objetivo es dar a cada caja, tan aproximadamente como sea posible, el mismo peso, una solución de calidad puede ser aquella que hace que la diferencia de pesos de las cajas sea pequeña, es decir las cajas están aproximadamente balanceadas.

Estas dos dimensiones de la memoria: calidad e influencia, pueden estar consideradas de manera explícita en el algoritmo tabú, es decir, incluir una estructura de datos que registre la etiqueta de un movimiento o solución como de calidad y/o influyente paralela al registro de su condición tabú o no tabú; o puede estar dada de manera implícita, como considerar que es de calidad si es un óptimo local y/o considerar que es influyente si permite diversificar la búsqueda.

Sin embargo las otras dos dimensiones: memoria de corto plazo y la de largo plazo, siempre van expresadas de manera explícita por medio de la lista tabú y de la tabla de frecuencias.

Una distinción importante en la búsqueda tabú es la de diferenciar la memoria de corto plazo y la de largo plazo. Cada tipo de memoria tiene sus propias estrategias, sin embargo, el efecto de la utilización de ambos tipos de memoria es el mismo: modificar el vecindario N(x) de la solución actual x (convertirlo en un nuevo vecindario N'(x))

En la memoria de corto plazo, el vecindario es generalmente:

$$N'(x) = \{N(x) - \text{Lista tabú}\} \ , \ N'(x) \subset N(x)$$

Respecto a las estrategias que usan la memoria de largo plazo N'(x) puede ser expandido para incluir otras soluciones que no están en N(x). En ambos casos el vecindario de x, N(x) no es un conjunto estático sino un conjunto que varía dinámicamente de acuerdo a la historia de la solución x.

En esencia la memoria de corto plazo utiliza la estructura tabú para penalizar la búsqueda y la memoria de largo plazo utiliza las frecuencias para determinar si un movimiento no tabú puede ser elegido o no.

5.3.3. Memoria a corto plazo

Es una "memoria" donde se almacenan los movimientos o atributos del pasado reciente, y que puede ser utilizada para "recordar" aquellos movimientos que hacen caer de nuevo en soluciones ya exploradas. Su objetivo es penalizar la búsqueda para evitar el ciclado.

Una manera de definir el entorno o vecindario reducido de una solución, consiste en etiquetar como tabú las soluciones previamente visitadas en un pasado cercano (*recency*), ciertos movimientos se consideran prohibidos (tabú), de forma que no serán aceptados durante un cierto tiempo o un cierto número de iteraciones, se considera que tras un cierto número de iteraciones la búsqueda está en una región distinta y puede liberarse del estatus tabú.

Un ejemplo de cómo opera la memoria a corto plazo se puede ver en la Figura 5.1., cuyos bloques son definidos en los siguientes puntos:

- Examen de la lista de candidatos: generar un (nuevo) movimiento desde la lista de candidatos, para crear una solución de prueba x' a partir de la solución actual.

- Prueba tabú: Identificar los atributos de x que cambiaron para crear x'. (Por ejemplo, pueden ser elementos añadidos o quitados, valores modificados de variables o funciones). Este bloque decide si los atributos pertenecen a un conjunto crítico de

atributos tabú, definidos previamente.

- Evaluación tabú sin penalización: Se crea una evaluación sin penalizar ya que no tiene estatus tabú.
- Prueba de aspiración: Se decide si la solución x' satisface un umbral de aspiración.
- Evaluación tabú con penalización: Se crea una evaluación penalizando la solución, ya que se trata de una solución con atributos tabú.
- Actualización de la elección: si la evaluación de x' es la mejor de todos los candidatos examinados, se debe registrar de una forma adecuada.
- ¿Chequeo suficiente?: se observa sí se han examinado suficientes movimientos (Según nuestro criterio).
- Ejecutar el movimiento elegido: realizar el movimiento desde x a x'.

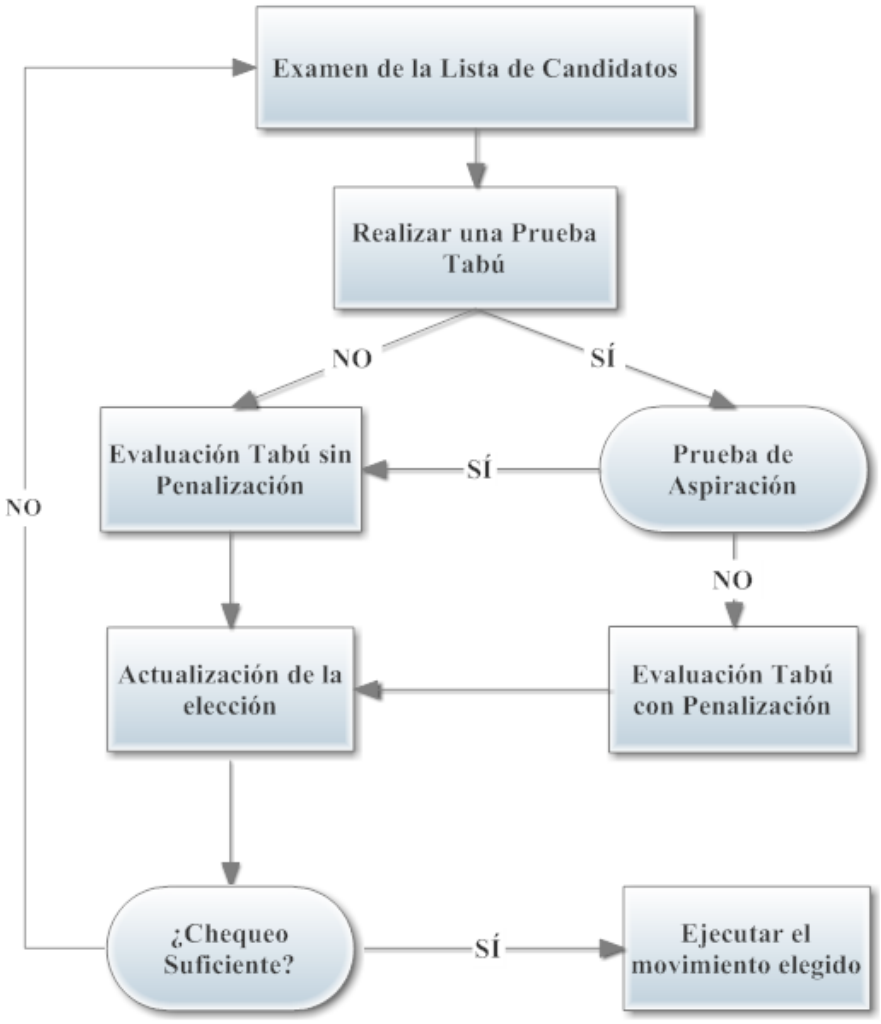

Figura 5.1: Esquema del proceso tabú de evaluación o memoria a corto plazo (Díaz, Glover, & Ghaziri, 1996)

5.3.4. Memoria a largo plazo

La memoria basada en frecuencia proporciona un tipo de información que complementa la información proporcionada por la memoria basada en lo reciente, ampliando la base para seleccionar movimientos preferidos. Como antes, la frecuencia a menudo toma en cuenta las dimensiones de calidad de la solución e influencia del movimiento. En esta estructura de memoria se registra la frecuencia de ocurrencias de los movimientos, las soluciones o sus atributos y puede ser:

- Frecuencia de transiciones: Cantidad de veces que una solución es la mejor o que un atributo pertenece a una solución generada.

- Frecuencia de residencia: Cantidad de iteraciones durante la cual un atributo pertenece a la solución generada.

  Una alta frecuencia de residencia, por ejemplo, puede indicar que un atributo es altamente atractivo si S es una secuencia de soluciones de alta calidad, o puede indicar lo contrario si S es una secuencia de soluciones de baja calidad.

  Por otro lado, una frecuencia de residencia que es alta, baja cuando S contiene tanto soluciones de alta como de baja calidad y puede apuntar a un atributo fortalecido (o excluido) que restringe al espacio de búsqueda, y que necesita ser desechado (o incorporado) para permitir diversidad.

La memoria a largo plazo tiene dos estrategias asociadas, intensificar y diversificar la búsqueda.

5.3.4.1. Intensificación

La intensificación consiste en regresar a regiones ya exploradas para estudiarlas más a fondo. Para ello se favorece la aparición de aquellos atributos asociados a buenas soluciones encontradas, iniciándose el regreso a esas soluciones atractivas para buscar más extensamente. En la Figura 5.2 se muestra un ejemplo de una estrategia de intensificación.

Se tienen dos variantes que han demostrado tener bastante éxito:

- Se introduce una medida de la diversificación para asegurar que las soluciones registradas difieran una de otra en un grado deseado, y borra toda memoria de corto plazo antes de reanudar el proceso desde la mejor de las soluciones registradas. (Voss, 1993)

- Se mantiene una lista secuencial de longitud limitada que añade al final una nueva solución sólo si es mejor que cualquier otra previamente vista. (Nowicki & Smutnicki, 1993)

  El actual último miembro de la lista es siempre el escogido (y suprimido) como base para reanudar la búsqueda. Sin embargo, la memoria a corto plazo de la búsqueda tabú que acompañó a esta solución también es guardada, y el primer movimiento inhibe además el movimiento previamente tomado de esta solución, con lo que un nuevo camino de solución será iniciado. Algunas veces este enfoque de recuperar la soluciones élite seleccionadas es llamado *backtracking* (desandar).

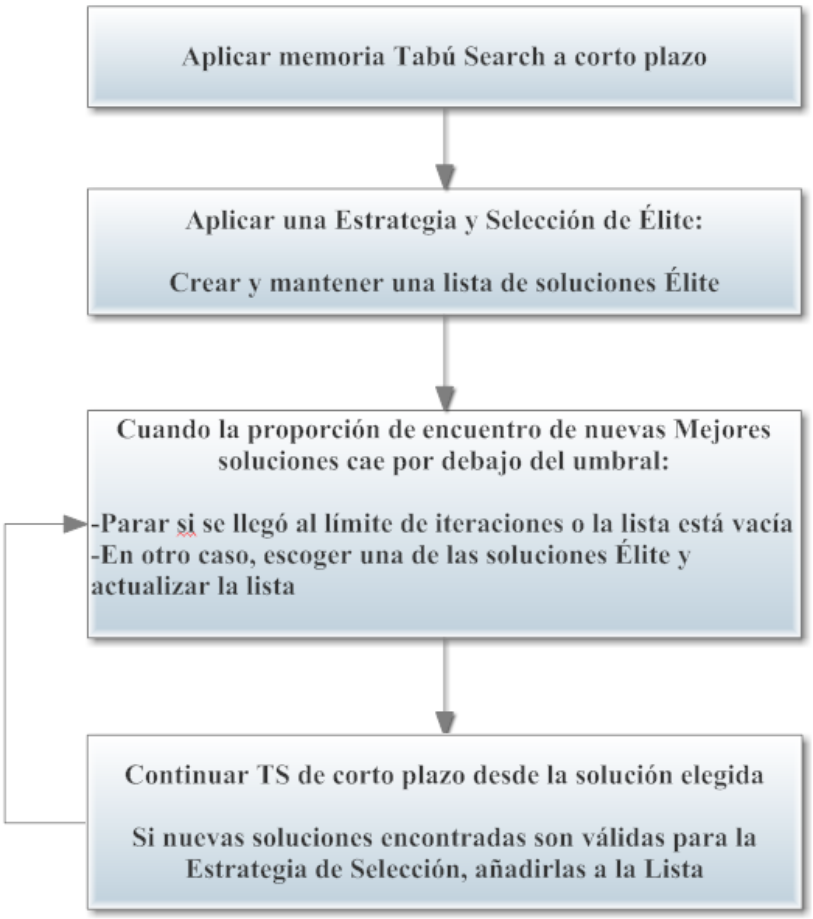

Figura 5.2: Enfoque simple de la intensificación en la búsqueda tabú (Díaz, Glover, & Ghaziri, 1996)

5.3.4.2. Diversificación

La Diversificación consiste en visitar nuevas áreas no exploradas del espacio de soluciones. Para ello se modifican las reglas de elección para incorporar a las soluciones atributos que no han sido usados frecuentemente.

Alternativamente, pueden introducir dichos atributos al reiniciar parcial o completamente el proceso de solución. Los mismos tipos de memorias previamente descritos son útiles como fundamento para tales procedimientos, aunque estas memorias sean mantenidas a través de subconjuntos (generalmente más largos) de soluciones, que aquellos mantenidos por estrategias de diversificación.

Un ejemplo simple de diversificación, que mantiene una memoria basada en frecuencia sobre todas las soluciones generadas es mostrado en la Figura 5.3.

Una forma clásica de diversificación consiste en reiniciar periódicamente la búsqueda desde puntos elegidos aleatoriamente, si se tiene alguna información acerca de la región factible se puede hacer un "muestreo" para cubrir la región en lo posible, si no, cada vez se escoge aleatoriamente un punto de partida (método multi-arranque).

Otro método propone registrar los atributos de los movimientos más utilizados en los anteriores movimientos, penalizándolos a través de una lista tabú a largo plazo.

Una correcta integración de las estrategias de intensificación y diversificación permite al

algoritmo alcanzar buenos resultados al problema en cuestión.

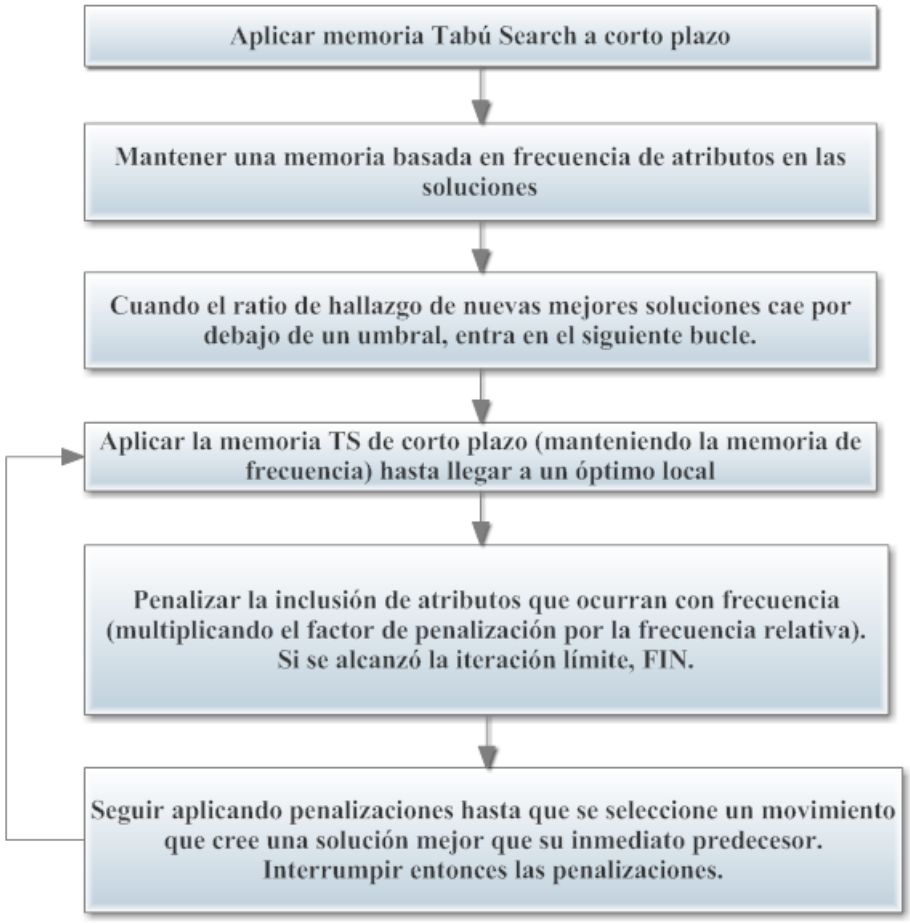

Figura 5.3: Enfoque simple de diversificación en búsqueda tabú (Díaz, Glover, & Ghaziri, 1996)

6. BÚSQUEDA TABÚ APLICADA A LA REGULARIZACIÓN SEMAFÓRICA

El objetivo principal del proyecto es el desarrollo de un algoritmo que permita resolver el problema de la regularización semafórica y la posterior obtención de ondas verdes. Aquí se adapta la búsqueda tabú a la resolución de dicho problema. Se va a tratar de obtener soluciones que maximicen lo máximo el número de vehículos que circulan sin detenerse por las intersecciones.

El algoritmo que se han realizado se exponen a continuación. La implementación de dicho algoritmo se ha realizado resolviendo el problema de la obtención de un árbol para una red de prueba, más pequeña que lo que será la red de Sevilla. Al final, a partir de la red la mejor solución será la que aporte un árbol con un mayor flujo, de todos los árboles posibles que se podrían formar.

6.1. Red de prueba

En primer lugar, para evitar el desarrollo del algoritmo en una red compleja donde la comprobación de los resultados y evaluación de la efectividad tiene muchas complicaciones, se expone una red sobre la que se harán las pruebas iniciales del algoritmo, para ver si las soluciones que aporta el algoritmo coinciden o no con la solución óptima de la red.

La red de prueba se ha querido que sea pequeña, pero que aún así contenga ciclos y una gran variedad de soluciones admisibles, para que el algoritmo vaya saltando de una solución a otra hasta encontrar la mejor.

Las características de la red son las siguientes, cuya estructura completa aparece indicada en la figura 6.1:

- 16 nodos
- 24 arcos

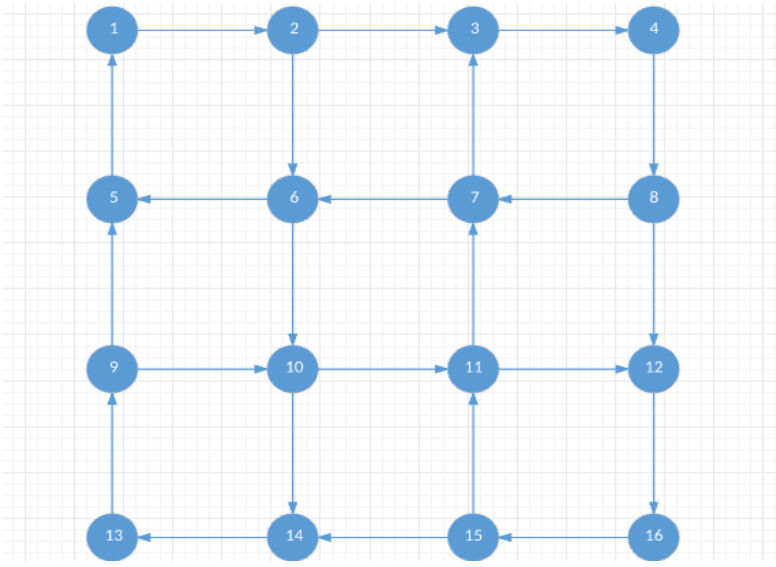

Figura 6.1: Red de prueba para realización de algoritmos (Fuente: **¿???**)

A los arcos de dicha red se les ha asignado un flujo arbitrario, y se ha resuelto la red para arcos no dirigidos, es decir, que los flujos para ambos sentidos del arco serán los mismos.

6.1.1. Solución de la red de prueba

Esta red de prueba se ha resuelto con el modelo matemático mediante la herramienta de programación lineal de Excel "solver". Así se ha obtenido la solución óptima, cuya función objetivo es de 1210 y cuyo árbol solución es el de la figura 6.2.

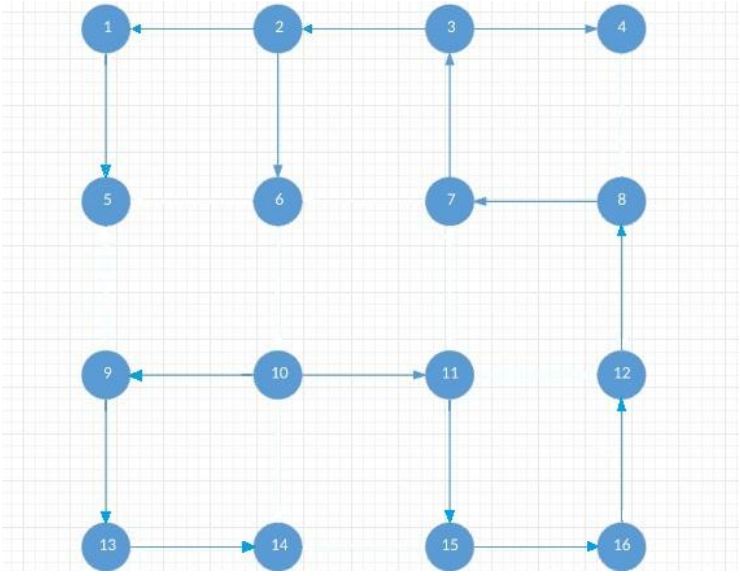

Figura 6.2: Solución a la red de prueba para realización de algoritmos (Fuente: elaboración propia)

Esta red ha sido de utilidad a la hora de desarrollar el algoritmo cuando aún no se tenía preparada la red de Sevilla, ya que permite aplicar el algoritmo y ver si la solución obtenida tiene soluciones admisibles, esto es, que no aparezcan nodos no conectados y que no se formen ciclos en la solución.

Una vez se encuentra un algoritmo que cumple todas las condiciones del problema y proporciona un árbol solución igual al de la solución óptima, ya habría que proceder a aplicar dicho algoritmo a la red de la ciudad, y comprobar si para un mayor número de nodos las soluciones son adecuadas.

6.2. Codificación del algoritmo

A continuación se va a exponer por partes como se ha codificado el algoritmo, sin entrar en el método de resolución pero explicando cuales serán los datos que se introducen, como se codifica la solución y como se evalúan las distintas soluciones.

6.2.1. Datos del problema

Se tendrán que proporcionar los siguientes datos al algoritmo, entre paréntesis aparecerá el nombre usado en la codificación. Con todos estos datos, se crean las matrices de incidencia y resto de elementos para que el algoritmo funcione:

- Número de nodos (nodos).

- Número de arcos (n).
- Vector origen de los arcos, que hace referencia a uno de los dos nodos que forman el arco (origen).
- Vector destino de los arcos, que junto al vector origen forman los arcos (destino). Para cada posición de dichos vectores se tienen los dos nodos que forman cada arco, por lo que tendrán que estar bien ordenados.
- Vector flujos de vehículos (flujos). Para cada posición i, este vector proporciona el flujo de vehículos que circula por el arco con nodos: origen(i) y destino(i).

6.2.2. Solución

La solución del problema se ha codificado de una forma bastante clara. Cada solución estará compuesta por un vector binario de tamaño igual al número de arcos que haya en el problema a considerar.

A continuación se expone como sería la solución para la red de prueba. Dado que se tienen 24 arcos en dicho problema, el número de elementos de dicha solución será 24.

1	1	1	0	0	1	1	1	0	1	0	1	1	0	1	1	0	0	1	0	1	0	1	1

Por otro lado, dado que se quiere obtener un árbol de dicha red, el número de unos en cada solución debe ser fijo. Por ejemplo, un árbol para la red de prueba con 16 nodos debe tener un total de 15 arcos, o lo que es lo mismo, 15 unos en la solución.

Los arcos a los que se hace referencia para la solución expuesta arriba para la red de prueba son los siguientes:

x12	x23	x34	x56	x67	x78	x910	x1011	x1112	x1314	x1415	x1516
						...					
x15	x59	x913	x26	x610	x1014	x37	x711	x1115	x48	x812	x1216

El objetivo será colocar dichos unos para que no se formen ciclos y todos los nodos queden conectados, pero eso se explicará más adelante y se hará mediante la función de evaluación.

6.2.3. Evaluación de las soluciones

La evaluación de las soluciones es, probablemente, lo más complejo de nuestro algoritmo. La necesidad de penalizar que no se queden nodos sin conectar y que no se produzcan ciclos, complica bastante la codificación. Todo lo que viene aquí explicado hace referencia a la función que aparecerá en la lista de anexos denominada evaluasol.m y que será la que evalúa las soluciones del problema. En dicha explicación se hará referencia a los valores de dicha función.

La función de evaluación final, empleada para la resolución del problema de la red de Sevilla, va a estar dividida en tres partes, de las cuales una será la referida a los flujos y las otras dos serán penalizaciones:

1. **Nivel de flujo**:

   Consiste en la evaluación del flujo de vehículos que se abarca en la solución. En este caso, cada arco que entre en la solución aportará su flujo para la función objetivo total.

   La codificación es bastante sencilla dado los datos que se tienen y como se ha definido la solución. Simplemente multiplicando el vector solución por el vector flujos ya se tendría el nivel de flujo de la solución.

   Dado que el resto de partes de la función de evaluación son penalizaciones, es este nivel de flujo el que formará parte de la función objetivo cuando se eliminen las penalizaciones.

2. **Penalización por nodos sin conectar**:

   Se penaliza con un valor alto el que alguno de los nodos queden sin conectar, ya que un árbol no puede tener ningún nodo aislado.

   En primer lugar se genera una matriz de incidencia con los arcos de la solución. Así para cada arco que esté dentro de la solución, se le pone un uno en una matriz de tamaño nodos (número de nodos), en el valor (i,j) de la matriz, siendo i el nodo origen y j el destino.

   Una vez creada la matriz de incidencia para la solución, se realiza una comprobación para ver si los nodos están conectados. Se recorren todos los nodos y para cada nodo i, se suman tanto la fila i, como la columna i de la matriz.

   Si la suma descrita anteriormente es igual a 0, esto quiere decir que dicho nodo no está conectado con ningún otro en la solución, y habrá que penalizar dicha solución con un valor muy alto.

   Además, si hay nodos sin conectar, no se lleva a cabo el punto 3 de la evaluación, ya que es el que más tiempo computacional ocupa y por tanto así se logra reducir el tiempo de computación total del algoritmo.

3. **Penalización por ciclos en la solución**:

   En el caso de que todos los nodos estén conectados, se procede a comprobar si hay ciclos en la solución. Dicha comprobación es algo compleja, por lo que se va a explicar detenidamente el proceso seguido.

   El hacer solo esta evaluación cuando todos los nodos estén conectados permite ahorrar un importante tiempo computacional, ya que una vez que falte algún nodo sin conectar la solución no será válida por lo que no es necesario evaluar una segunda penalización.

   En primer lugar se crean dos vectores binarios, ambos de tamaño nodos. El primero se denomina explorar y contendrá todos los nodos que quedan por explorar conectados a los que ya están explorados, en los que aparecerá un valor uno. Por otro lado se tiene el vector terminados que indica los nodos que ya han sido explorados.

Se empezará a explorar por el nodo 1, por lo que para que el algoritmo empiece a funcionar se asigna explorar(1)=1. Mediante un bucle while se le obliga al algoritmo a iterar mientras que no haya un uno en todos los elementos del vector terminados o que la suma de todos los elementos del vector terminados sea igual a la suma de todos los elementos del vector explorar. Este último quiere decir que se habrá llegado a un ciclo, ya que no hay nodos por explorar diferentes a los ya explorados.

En cada iteración se crea un vector de ceros de tamaño nodos llamado pendientes, en el que se añaden los nodos que se deben explorar y no han sido explorados. Se comprueban todos los nodos y si explorar(i)=1 y terminados(i)=0 entonces pendientes(i)=1, por lo que se debe explorar.

Una vez se crea dicho vector de elementos pendientes de comprobar, se escoge el primero de los nodos que aparecen con un 1 en dicho vector y se analiza con quien está conectado dicho vector gracias a la matriz de incidencia.

Dicho nodo se almacena en una variable llamada analizar y comprueba gracias a la matriz incidencia, para todos los nodos i si los valores incidencia(analizar,i) o incidencia(i,analizar) son iguales a 1. Si lo son y si además dichos nodos no han sido ya explorados (terminados(i)=0), se añade el nodo pendiente de explorar mediante explorar(i)=1.

Se actualiza el vector terminados y se sigue con el bucle while hasta salir de el por una de las dos condiciones explicadas. Una vez fuera, si se ha salido de dicho bucle y la suma de los elementos del vector terminados es inferior al número de nodos, esto quiere decir que se ha encontrado un ciclo, y por tanto se penaliza dicha solución con un valor alto.

La función objetivo total será la suma del nivel de flujos de la solución más una de las dos penalizaciones, bien por nodos sin conectar o por ciclos en la solución.

6.3. Resolución mediante Búsqueda tabú

En este apartado se van a definir tanto parámetros propios de la lista tabú como elementos necesarios para el desarrollo del algoritmo, como pueden ser la solución inicial de la que se parte o la vecindad de las soluciones.

6.3.1. Solución inicial

Todo lo que aparece explicado aquí hace referencia a la función generasol.m que aparecerá en la lista de anexos al final de este trabajo, y es junto a la función evaluasol.m la función con más carga de codificación.. En la explicación de esta función aquí descrita aparecen variables usadas en dicha codificación.

Durante el desarrollo inicial del algoritmo, se partió de una solución inicial completamente aleatoria, con nodos-1 valores igual a uno y el resto cero, pero sin tener en cuenta que se eviten ciclos, o que todos los nodos queden conectados, por lo que todos estos problemas podían aparecer en nuestra solución inicial y condicionar así el resto de soluciones.

Por tanto, y dado que el algoritmo mostraba problemas para escapar de soluciones con ciclos, se

decidió partir de una solución inicial que fuera un árbol y que, por tanto, dicha solución tiene todos los nodos conectados y no se forman ciclos. A partir de ahí el algoritmo se centra en no salir de soluciones que sean arboles, gracias a la penalización por ciclos y por nodos sin conectar.

Se va a exponer brevemente como genera un árbol inicial el algoritmo desarrollado.

- **Creación de matriz de incidencia:**

En primer lugar se crea una matriz de flujos que será similar a una matriz de incidencia de todos los arcos del problema, pero en lugar de aparecer un uno entre dos nodos, aparecerá el flujo que hay en ese arco.

Una vez se genera dicha matriz, se parte de un nodo aleatorio y se inicia el proceso de creación de la matriz de incidencia de la solución. Se crea dicha matriz en lugar de la solución tal y como se ha codificado porque es más sencillo, aunque posteriormente se procede a pasar de dicha matriz al vector solución.

Se tiene por un lado el vector nodousado, que almacenará los nodos que han sido usados para nuestra solución. Por otro lado se tiene un vector llamado candidatos, que almacenará los candidatos a entrar en la solución, que no deben haber sido usados y que deben ser accesibles desde los nodos que aparecen en nodousado.

El procedimiento será durante n-1 veces (que será el número de arcos que tenga nuestra solución y, por lo tanto, el número de unos que debe aparecer en la matriz de incidencia) se genera una lista de candidatos para los nodos usados, se recorre esa lista y se observa para cada nodo de esa lista con qué otros nodos están conectados y se rellena el vector candidatos sin comprobar si se han usado anteriormente.

Posteriormente se procede a recorrer los vectores candidatos y nodousado, esto es, para cada nodo usado se recorren todos los candidatos del vector. Si coincide que uno de los candidatos no estaba usado y además existe flujo en la matriz de flujo que conecta a nodousado(j) con candidatos(k), se rellenan los índices descritos anteriormente de la matriz de incidencia de la solución.

Una vez realizado todo este proceso, ya se tiene una matriz de incidencia que contiene un árbol solución aleatorio ya que conecta todos los nodos (los nodos solo podían ser usados una sola vez) y el vector es de valor nodos-1, por lo que se obliga con estas características a que la solución obtenida tenga forma de árbol.

- **Obtener solución inicial codificada**

El siguiente paso es pasar dicha matriz de incidencia al formato de nuestra solución. Para ello se recorrerán todos los nodos, con una variable p para las filas y otra variable q para las columnas. Cuando incidencia(p,q) sea igual a 1, esto querrá decir que ese arco tiene que entrar en la solución.

Tal y como hemos introducido los datos, será necesario crear una variable que recorra los vectores origen y destino, para encontrar que arco hace referencia a ese valor 1 que se ha encontrado en la matriz de incidencia. Una vez que se encuentre dicho valor, siendo r la variable que recorre los vectores desde 1 hasta n, se coloca un uno en sol(r).

Se tendrá así una solución que formará un árbol de la red, unido esto a las penalizaciones que se aplican por formar un ciclo o por dejar sin conectar nodos, hacen que las soluciones obtenidas durante el desarrollo del algoritmo sean ciclos, ya que no conviene salir de dichas soluciones por las altas penalizaciones.

6.3.2. Vecindad

Para la vecindad del algoritmo se ha optado por un intercambio entre dos elementos de una solución, para así dar a una solución distinta que mejore a la anterior. Más concretamente, se prueba a intercambiar todos los elementos que aparecen en la solución por lo que el número de vecinos de una solución será bastante amplio, para problemas grandes como el que se tiene en este trabajo.

La función del algoritmo que realiza este intercambio se denomina actualizasol.m y será explicada a continuación:

Si, por poner un ejemplo, se tiene un vector con 15 elementos, esto es, un problema con 15 arcos, y se tienen 9 nodos, por lo que el valor de unos en la solución debe ser obligatoriamente 8, una posible solución sería:

[0 1 1 1 0 1 1 0 0 1 1 0 0 0 0]

Por tanto, los vecinos de esta solución serán todos aquellos que se obtengan mediante intercambios de dos elementos de la solución, siempre y cuando la solución obtenida sea diferente a la anterior (para evitar así intercambios de unos por unos o ceros por ceros, que dan lugar a la misma solución).

Por ejemplo, para nuestra función, el vector anterior se recorrerá mediante dos variables i y j, donde i < j. Si i=1 y j=2, la solución vecina que se obtendrá es la siguiente:

[1 0 1 1 0 1 1 0 0 1 1 0 0 0 0]

Para i=1, si j=3 y j=4 se siguen dando vecinos diferentes a la solución de partida:

[1 1 0 1 0 1 1 0 0 1 1 0 0 0 0]

[1 1 1 0 0 1 1 0 0 1 1 0 0 0 0]

Sin embargo para i=1 y j=5 el valor de la solución es el mismo igual a 0, por lo que si se intercambian dichos elementos la solución obtenida es la misma, por lo tanto no se genera ningún vecino. En nuestro caso el algoritmo no intercambia dichas soluciones ya que la solución obtenida sería la misma que a la que se está obteniendo la vecindad:

[0 1 1 1 0 1 1 0 0 1 1 0 0 0 0]

Además también se comprueba que el intercambio que se está realizando sea admitido según una lista tabú, que será explicado más adelante y que va a restringir ciertos intercambios por haber sido realizados previamente.

Cada uno de los vecinos obtenidos son evaluados en nuestra función y se va almacenando el mejor vecino obtenido para todos los intercambios de una solución, y la función objetivo de dicho vecino, para seguir comparando y que se quede con la mejor solución de todas.

Además también se irá almacenando variables relativas a la lista tabú, que hacen referencia a los índices de la solución que se han intercambiados, para luego poder actualizar la lista tabú, que será explicada a continuación.

6.3.3. Lista tabú y número de iteraciones

Estos dos elementos contienen los dos únicos parámetros de nuestro algoritmo, el tamaño de la lista tabú y el propio número de iteraciones.

6.3.3.1. Lista tabú

Para la formulación del algoritmo, será necesario definir un tamaño de la lista tabú para permitir a la búsqueda explorar el espacio de soluciones, además de un número máximo de iteraciones.

En primer lugar, se va a explicar la lista tabú elegida, que va a tener una memoria basada en atributos, ya que va a almacenar movimientos en la solución. Se va a exponer brevemente una explicación del tamaño de la lista tabú, para comprender mejor su funcionamiento.

- Tamaño de la lista tabú (tabusize1) : También conocido como *tabú tenure*, se trata del valor que define el número de iteraciones que estará restringida una solución o el número de iteraciones que se restringe un movimiento concreto entre tareas.

  Dependiendo de cómo sea, se podrá salir más fácilmente de óptimos locales, ya que me restringe un gran número de soluciones o muchos movimientos, haciendo empeorar la solución y llegando a otros puntos del espacio de búsqueda.

  Este parámetro será diferente para cada problema, ya que cada uno tiene unas características que hacen que para salir de una región (con óptimo local), se necesite un tamaño tabú acorde a como se haya definido la lista.

En nuestro algoritmo, la lista tabú será una matriz de tamaño n, inicialmente rellena de ceros y que conforme se va iterando se va rellenando con unos según los intercambios que se vayan realizando de una solución a otra.

Si para obtener una solución determinada se ha intercambiado el elemento i con el elemento j de una determinada solución, dicha lista tabú hace que no se pueda volver a realizar dicho intercambio de i con j durante un número de iteraciones, así se consigue que no se repitan soluciones y no se entre en bucle.

Para este problema, se ha probado con diferentes tipos de listas tabú para observar cuál de ellas daba mejor resultado sobre el problema. Aunque al final se ha elegido la que restringe el intercambio de dos elementos de solución, también se pueden tener los siguientes casos:

- ❖ Restringir no solo el intercambio entre dos elementos de la solución, sino el intercambio de esos dos elementos con cualquier otro. La diferencia es que no solo añadimos un 1 en el intercambio sino en toda la fila/columna de esos elementos.

  De esta forma se tendría una lista tabú más restrictiva y permitiría poder salir de soluciones similares que se encuentren en un óptimo local, pero que para este problema no ha dado buenos resultados.

- ❖ Aunque se ha optado por usar una memoria en atributos, también sería una posible opción el usar una memoria explícita que almacene las últimas soluciones obtenidas por el algoritmo, para que no se vuelvan a repetir.

  En este caso, decir que también se ha probado a usar este tipo de memoria pero que igualmente los resultados no han sido del todo satisfactorios.

6.3.3.2. Número de iteraciones

Por otro lado se tiene el número de iteraciones, un parámetro fundamental en nuestro algoritmo y que hay que definir de acuerdo a las soluciones encontradas, analizando en qué momento se encuentran las mejores soluciones.

- Número de iteraciones (niter): Será el número de veces que el algoritmo realiza el proceso de búsqueda y elección de la mejor solución posible. Es necesario para evitar que el algoritmo entre en un proceso sin fin, por lo que se establece una cota superior. El valor debe estar equilibrado, ya que:
  - Un valor muy bajo puede que no le otorgue suficiente tiempo al programa para encontrar una buena solución.
  - Si se tiene un valor muy alto el tiempo de computación sería demasiado grande, y se tendría un algoritmo muy ineficiente.

6.3.4. Funcionamiento del algoritmo

En la figura 6.3 se puede observar el funcionamiento del algoritmo expuesto paso a paso, donde se observa también como se va a actualizar la lista tabú en cada iteración del algoritmo.

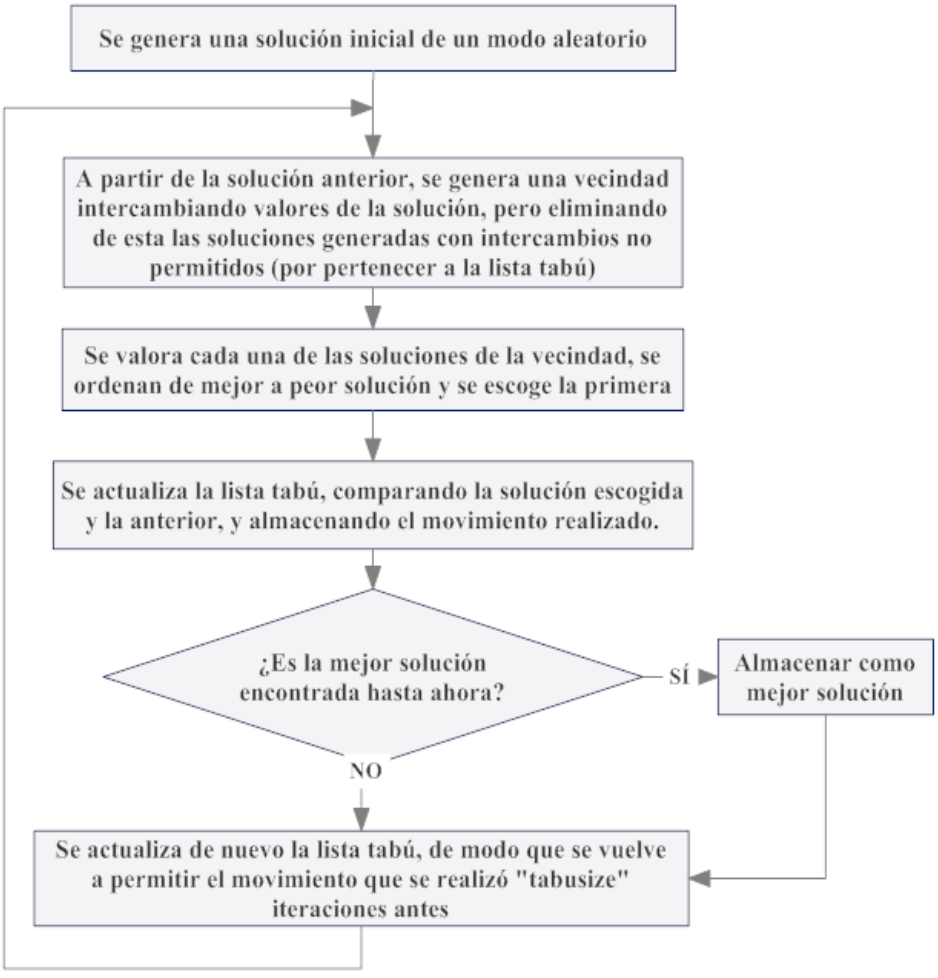

Figura 6.3: Búsqueda tabú con memoria basada en atributos (Fuente: elaboración propia)

Si se recorre el código de la función principal de este algoritmo, tal y como se ha hecho con las otras funciones, para explicar claramente que procesos realiza el algoritmo, dicho método seguido es el siguiente:

En primer lugar, se inicializa la variable temporal mediante la función tic, para así conocer el tiempo computacional que emplea nuestra función.

Tras ello, se añaden los datos que van a ser usados por el algoritmo, como son el número de arcos del problema (n), el número de nodos (nodos), el valor de los flujos de cada arco que será un vector de tamaño n (flujos), dos vectores de tamaño el número de arcos que contendrán al origen y al destino de cada arco (origen y destino).

Hay que tener en cuenta que aunque haya un origen y destino en dichos arcos, algunos de ellos son bidireccionales (la mayoría de ellos) y otros no (los que hacen referencias a calles con un solo sentido de circulación.

Seguidamente se definen unos parámetros para el algoritmo, en concreto, se añade el número de iteraciones que se quiere que tenga el algoritmo, se añade el tamaño de lista tabú que se quiere en nuestro algoritmo y se inicializan dos variables, la variable historico que será para representar las soluciones y la lista tabú, llamada listatabu1, que será en principio una matriz de ceros de tamaño n y que posteriormente se cambiará conforme se desarrolle el algoritmo.

A continuación se genera una solución inicial, obtenida mediante la función generasol.m expuesta en el apartado 6.3.1. y que como se ha explicado antes, se ha conseguido que la solución que salga de dicha función sea un árbol, y gracias a las penalizaciones se permite que se siga manteniendo un árbol durante todo el algortimo.

Dicha solución inicial es evaluada mediante la función evaluasol.m explicada en el apartado 6.2.3 y tiene en cuenta todas las características ahí descritas. Una vez evaluada, la solución inicial será la mejor solución por lo que las variables mejor_sol y mejor_fo que serán las que almacenen los mejores valores durante todo el algoritmo, se inicializan con la solución inicial.

Entonces se genera un bucle durante el número de iteraciones que se haya determinado al principio. En cada iteración se realiza lo siguiente:

- Llamar a la función actualizasol.m para generar un nuevo vecino. Como se explicó en el apartado 6.3.2. se seleccionará a la mejor solución de todas las posibles soluciones obtenidas a partir de un intercambio de dos elementos de la solución.
- Se actualiza la lista tabú, restando un 1 a todos los elementos de la matriz de lista tabú. Para que los valores que estaban como 0 sigan así, se busca todos los elementos de la matriz que sean menor a 0 y se les pone igual a 0.
- Se evalúa la solución obtenida mediante la función de vecindad y se añade dicha función en el vector historico, que será para ver como evalúan todas las soluciones durante el número de iteraciones.
- Se realiza la comprobación para saber si la solución obtenida en esta iteración es la mejor solución obtenida hasta ahora, para almacenar la mejor solución a lo largo del algoritmo.

Una vez se ha llegado al número de iteraciones, se presenta por pantalla cual es la mejor solución encontrada durante todo el funcionamiento del algoritmo, la función objetivo obtenida de esa solución, el tiempo de computación y la iteración en la que ha encontrado la mejor solución al problema.

6.3.4. Parámetros del algoritmo

Una vez se ha definido el funcionamiento del algoritmo, será necesario comprobar cuales serán los parámetros de nuestro algoritmo, que en este caso solo tenemos dos:

- **niter**: se establece su valor mediante pruebas en el algoritmo, probando a poner un número alto de iteraciones y comprobando en qué iteración encuentra la mejor solución obtenida.

  Para ello se inicializa una variable denominada contadoriter al inicio de la función principal, y se actualiza siempre que se actualicen las variables mejor_sol y mejor_fo que almacenaran los mejores valores encontrados hasta el momento.

  En nuestro caso se ha determinado para diferentes pruebas del algoritmo, en qué iteración encuentra el algoritmo la mejor solución y para ello se le ha dado un valor alto al número de iteraciones (niter=200) y se ha almacenado la variable contadoriter. Las pruebas dieron las siguientes soluciones:

Prueba	contadoriter
1	24
2	27
3	31
4	25
5	26
Media:	26,6

  Luego a la vista de los valores de contadoriter, las mejores soluciones del algoritmo las encuentra al principio, y por tanto el funcionamiento de la lista tabú no es del todo necesario, ya que la mejor solución la encuentra al principio.

  Sin embargo, dado que es posible que se repitan algunas soluciones y el algoritmo entre en bucle, se mantiene dicha lista, además de que para problemas más complejos para los que se pueda reutilizar este algoritmo es posible que dicha lista tabú si tenga una mayor importancia y permita el movimiento por el espacio de soluciones.

  En definitiva, se toma **niter=40** para que se permita alcanzar la mejor solución, ya que en todas las pruebas el valor era como mínimo un 25% inferior y se tiene un margen suficiente para llegar a la mejor solución.

- **tabusize1**: será el tamaño de la lista tabú, y tendrá que ser suficientemente grande para que permita salir de los óptimos locales y suficientemente pequeño para que no se queden soluciones sin alcanzar que puedan mejor la función objetivo por culpa de una restricción en la búsqueda tabú.

  Para el cálculo del número de iteraciones se empleó una lista tabú de tamaño 30, dado que para el testeo para obtener niter, el número de iteraciones era 200. Ahora que niter es fijo e igual a 40, se debe poner un tamaño de lista tabú más pequeño.

Se determina que un tamaño de lista tabú **tabusize1=10** podría permitir escapar de ciertos óptimos locales y permitir acceder a todas las soluciones sin que pueda ocurrir que se queden soluciones mejores sin explorar.

Luego ya se tienen definidos los dos parámetros que tiene nuestro algoritmo, y por tanto ya se puede analizar la solución encontrada, ver si es un árbol de la red del problema y comparar dicho problema con otros métodos de resolución, para comprobar si de esta forma se mejoran las soluciones y se llega un nivel de flujo mayor en el árbol de la solución.

7. RESULTADOS

A continuación se muestra los resultados proporcionados por el algoritmo escogido para la red de la ciudad de Sevilla.

Es importante especificar con detalle el ordenador con el que se prueba el algoritmo, ya que va a influir sobre el tiempo de computación. En nuestro caso, las características son las siguientes:

- Ordenador: HP ProBook 4520s
- Procesador: Intel Celeron CPU P4500 a 1,87 GHz
- Memoria RAM: 2,86 GB
- Sistema Operativo: Windows 7 Home Premium

Dado que la red obtenida no es del todo compleja y su resolución, aunque mucho más lenta que para la red de prueba, no ocupa mucho tiempo computacional, se ha podido testear el algoritmo en muchas ocasiones, para observar si los resultados difieren de una prueba a otra.

Al principio, conforme se iba desarrollando el algoritmo los resultados eran más dispares, algunas veces daba un resultado bueno y otras caía en penalizaciones. Sin embargo, gracias a las características del algoritmo final, se ha conseguido que la mejor solución de todas las obtenidas mientras se desarrollaba, sea la que proporciona el algoritmo cada vez que se ejecuta.

El árbol solución que proporciona el algoritmo es el que se muestra en la figura 7.1, sin tener en cuenta los sentidos de circulación de las calles.

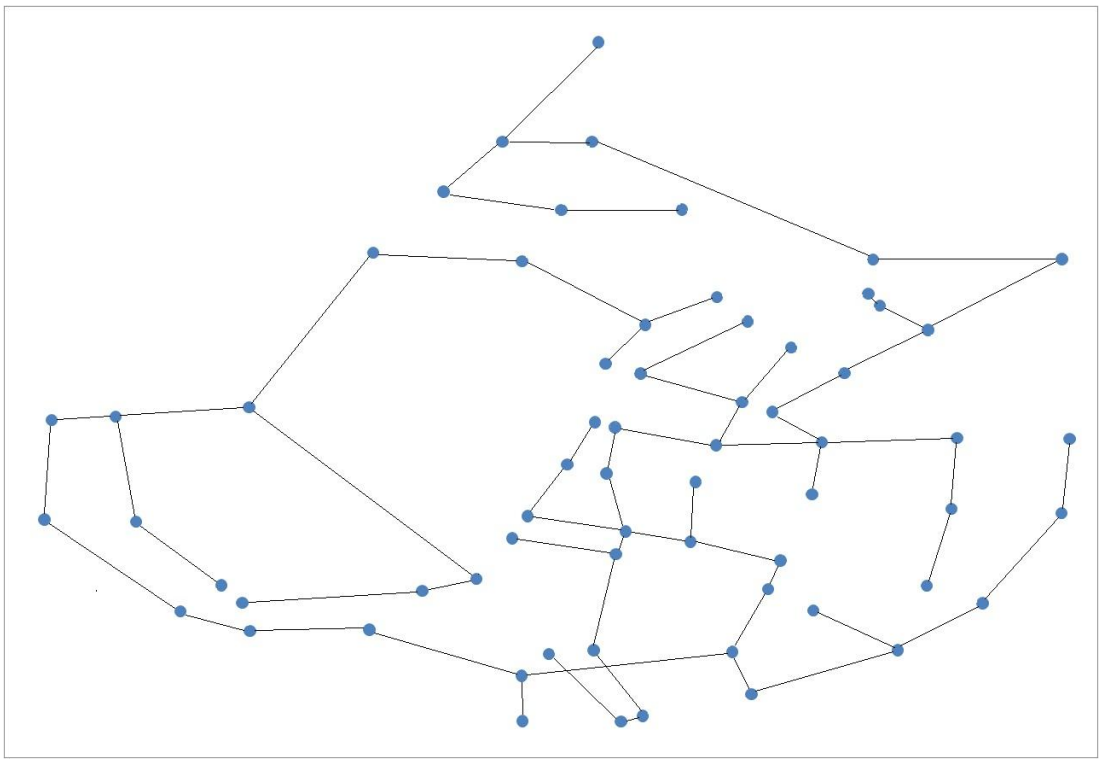

Figura 7.1: Solución para la red de Sevilla mediante el algoritmo

Dicha solución, tiene un valor de la **función objetivo de 117566**, el cual, dado que la solución no tiene nodos sin conectar ni tiene ciclos, hace referencia al flujo de vehículos de los arcos de dicha solución, además de que al ser un árbol se confirma que la función de evaluación funciona correctamente.

Para la resolución, no se han tenido en cuenta las calles de un solo sentido, ya que el problema se ha resuelto con arcos no dirigidos. Sin embargo, sí que se ha ido almacenando el sentido de cada una de las calles, para que una vez que se tenga la solución se le asigne un sentido a los arcos de esta.

Para las calles de un solo sentido está claro cuál será la dirección del arco, mientras que para las que tienen dos sentidos, el sentido del arco será el que tuvo un mayor flujo a la hora de seleccionar los flujos de cada arco.

Por tanto, si se traslada la solución anterior de la figura 7.1 al árbol con arcos dirigidos, escogiendo el mayor flujo de los dos sentidos, se tiene el árbol de la figura 7.2.

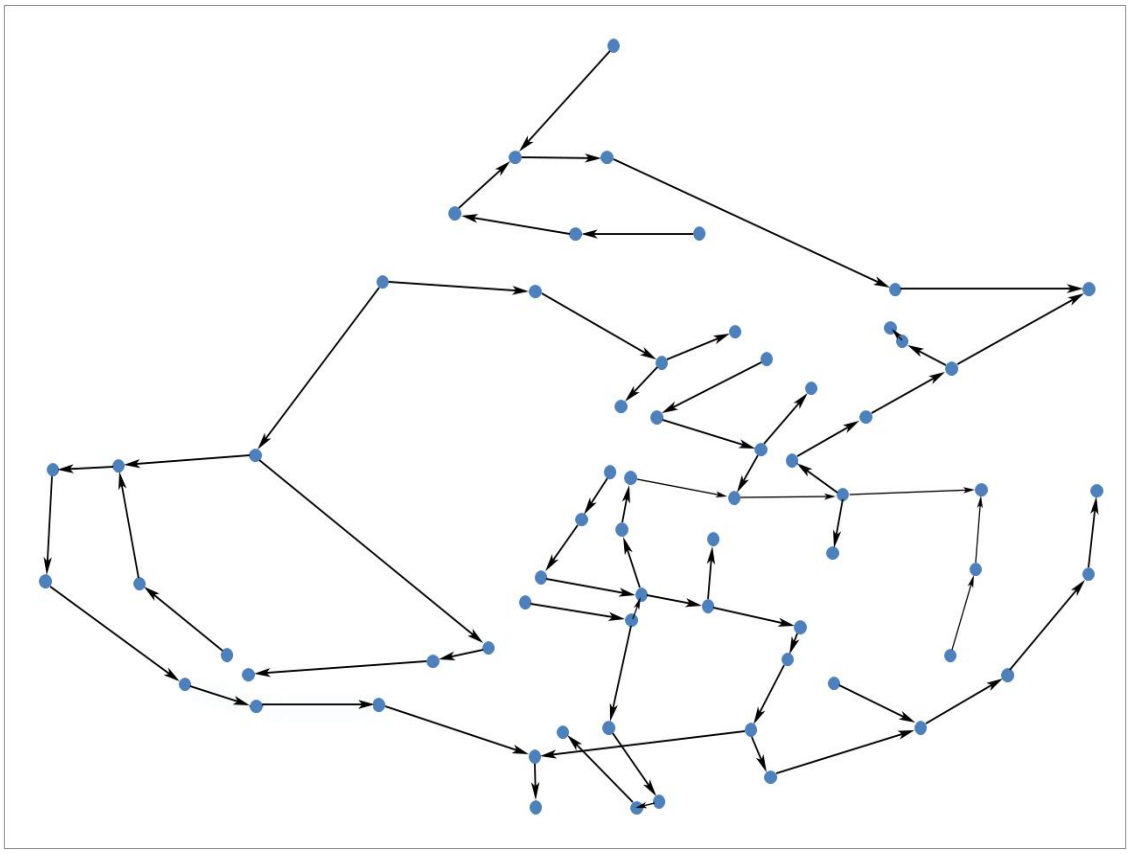

Figura 7.2: Árbol solución de la red de Sevilla con arcos dirigidos

Hay que tener en cuenta, que en varias calles el flujo de vehículos era prácticamente el mismo en ambos sentidos de circulación, por lo que en dichas vías se ha tomado el sentido que siguen el resto de vías que llegan y salen a dicho arco, para que el árbol solución sea coherente, aún así, el sentido de algunos arcos se ha tomado aleatorio ya que el flujo era similar.

Hay que tener en cuenta, que aunque el sentido se haya tomado aleatorio en casos de flujos similares en ambos sentidos, para la resolución del problema se sigue teniendo como flujo de ese arco el mayor flujo de los dos sentidos, ya que el problema se ha resuelto con el flujo máximo del arco, sea cual sea su sentido.

El problema de ser una red desarrollada para este trabajo es el poder comparar si los resultados obtenidos son buenos, si se acercan al óptimo o lo alcanzan, o si hay algún error que no se está teniendo en cuenta.

Por suerte, dado que la red de Sevilla se ha realizado en colaboración con otro compañero para su trabajo fin de grado (Jiménez, 2016) se tienen los datos de valor de la función objetivo y de tiempo de computación de su algoritmo, por lo que es interesante poner ambos resultados en una tabla para así comparar los valores.

	Algoritmo Tabú	Algoritmo (Jiménez, 2016)
Función objetivo:	117566	111894
Tiempo computacional:	22 minutos aprox.	900 minutos aprox.

Comparando dicho algoritmo con este, se puede comprobar una mejora sustancial tanto en calidad de la solución como en tiempo computacional, ya que el algoritmo desarrollado en (Jiménez, 2016) crea árboles aleatorios y se queda con el mejor de ellos, mientras que el nuestro hace una búsqueda mejorando en cada iteración la solución, mediante ciertos intercambios.

8. CONCLUSIONES

En primer lugar, decir que la elaboración de dicho algoritmo no hay sido algo fácil, ya que el modelar la red y sacar de ahí un árbol ha sido complicado. Sin embargo, el hecho de poder superar todos los inconvenientes de codificación del algoritmo para así llegar a soluciones sin errores, ha sido muy gratificante y he quedado bastante satisfecho con los resultados proporcionados por el algoritmo.

Se van a especificar en varios puntos las conclusiones a las que se pueden llegar a expensas de los resultados obtenidos y el trabajo realizado:

- El algoritmo de búsqueda tabú no es del todo adecuado para esta red, ya que la mejor solución la encuentra durante las primeras iteraciones, por lo que no se da lugar a que la lista tabú haga su propósito de salir de óptimos locales, pese a que se ha probado el algoritmo con varios tipos de listas tabú.

  El principal motivo por el que ocurre esto puede ser que se realice una búsqueda exhaustiva del espacio de soluciones, dado que la complejidad de la red no es muy alta, y por tanto se llega a la mejor solución gracias a dicha búsqueda.

  Se concluye así que, para esta red, un algoritmo de búsqueda exhaustiva puede ser igual de efectivo que uno de búsqueda tabú. En cualquier caso, es interesante dejar el algoritmo con búsqueda tabú, dado que para problemas más complejos que este, puede ser importante el empleo de la lista tabú para salir de óptimos locales.

- Si que se ha podido comparar las soluciones obtenidas en este trabajo, con las soluciones obtenidas en (Jiménez, 2016) con una sustancial mejora en los resultados, tanto en valor de la función objetivo como tiempo computacional.

  Queda constatado que el uso de soluciones aleatorias en estos problemas es mucho más ineficaz que un algoritmo que mejore la solución mediante el uso de una vecindad y variaciones en las soluciones.

  Para este problema, comparando los resultados con respecto a los resultados proporcionados en (Jiménez, 2016), en este algoritmo **la mejora ha sido de algo más de un 5%** en la función objetivo, mientras que **el tiempo computacional se ha reducido algo más de 40 veces**.

- Dado que la red a la que se aplica el algoritmo desarrollado, también ha sido desarrollada para este trabajo, no se tiene una red de otra ciudad, real o ficticia, de una complejidad mayor o igual a la que se ha desarrollado aquí, en la que poder comprobar la eficacia de nuestro algoritmo.

Por otro lado, si que se puede comprobar el algoritmo con las ondas verdes que se observan en el último plano de coordinación semafórica que se tiene de la ciudad de Sevilla, tal y como se describió en la introducción de este trabajo.

En la figura 8.1 se puede observar dicha comparación, aunque es complicado de ver, pero se puede observar que las vías más importantes aparecen en ambas imágenes. Por otro lado los sentidos de dichos arcos pueden variar dependiendo de los flujos que se tienen para los arcos de la red desarrollada

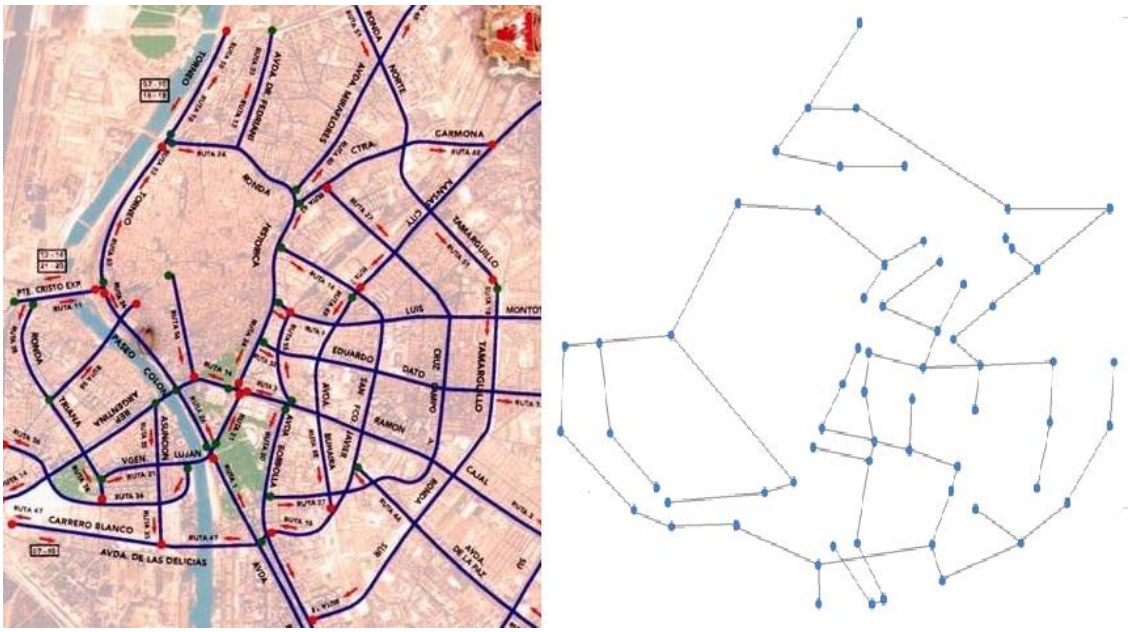

Figura 8.1: Comparación ondas verdes de Sevilla, con el árbol solución de nuestro algoritmo

En definitiva, se puede concluir que la aplicación de algoritmos de este tipo para el control de los semáforos en grandes ciudades, las cuales requieren una regulación que no se puede obtener a simple vista sino que requiere una gestión de datos de flujos, pueden aportar ciertas mejoras encaminadas a reducir los problemas derivados de la congestión por cruces semafóricos.

Dichas mejoras en la congestión, repercuten en menores tiempos de parada con un mejor flujo de vehículos en los cruces, y ayuda a disminuir los riesgos de accidentes en las vías, lo cual hace que este tipo de algoritmos puedan ser atractivos para quienes tienen el control de la sincronización de semáforos en las distintas ciudades.

BIBLIOGRAFÍA

Al-Mudhaffar, A. (2006). Impacts of Traffic Signal Control Strategies. *Doctoral Thesis. Royal Institute of Technology* . Stockholm.

Bonneson, J., Sunkari, S., Pratt, M., & Songchitruksa, P. (2009). *Traffic signal operations handbook.* Austin, Texas: Texas A&M University. Texas Transportation Institute (TTI).

Capaldo, F. S., & Biggiero, L. (2012). Some surveys in order to static traffic light coordination. *First International Conference on Traffic and Transport Engineering*, (págs. 313-320). Belgrado.

Díaz, A., Glover, F., & Ghaziri, H. e. (1996). *Optimización Heurística y Redes Neuronales.* Madrid: Paraninfo.

Dirección general de tráfico. Oposiciones. Temario completo comprimido. Parte 3. Gestión Técnica del Tráfico (2014). Obtenido de http://www.dgt.es/es/la-dgt/empleo-publico/oposiciones/2014/20141216-temario-promocion-interna-2014-parte-3-gestion-tecnica-del-trafico.shtml

Garcia-Nieto, J., Alba, E., & Olivera, A. C. (2012). Swarm intelligence for traffic light scheduling: Application to real urban areas. *Engineering Applications of Artificial Intelligence. V25 (2)* , (págs. 274-283).

García-Nieto, J., Alba, E., & Olivera, A. (2012). Planificación de Ciclos en Semáforos con PSO: Casos de EStudio sobre Málaga y Sevilla. *VIII Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'12)*, (págs. 117-124). Albacete.

Glover, F. (Summer 1989). Tabu Search - Part I. *ORSA Journal on computing, V1 (3)* (págs. 190-206).

Holland, J. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* Ann Arbor: University of Michigan Press.

Ivorra, M. C., Ivorra, J. F., Prieto, J. I., Carrión, M. T., Cardona, M. G., Jover, R. T., y otros. (2002). Métodos geométricos de coordinación de intersecciones reguladas por semáforos. *XIV Congreso Internacional de Ingeniería Gráfica.* Santander.

Jiménez, J. M. (2016). *Propuesta de algoritmo para sincronización semafórica en una ciudad.* Sevilla: Trabajo Fin de Grado. Universidad de Sevilla.

Nowicki, E., & Smutnicki, C. (1993). New results in the worst-case analysis for flow-shop scheduling. *Discrete Appl. Math. 46* , (págs. 21-41).

Osman, I., & Kelly, J. (1996). Meta-Heuristics: An overview. En *I.H. Osman, J.P. Kelly (Eds.), Meta-Heuristics—Theory and Applications, Kluwer Academic Publishers* (págs. 1-21). Boston.

Voss, S. (1993). Tabu Search: Applications and Prospects. *Z. Du, P. Pardalos (Eds.), Network Optimization Problems: Algorithms, Applications, and Complexity, Utopia Press, Singapur* (págs. 333-353).

Warberg, A., Larsen, J., & Jorgensen, R. (2008). Green Wave Traffic Optimization - A Survey. *Informatics and Mathematical Modelling. (D T U Compute. Technical Report; No. 2008-01).*

LISTA DE ANEXOS

En esta parte del proyecto se van a exponer el algoritmo implementado en lenguaje M, para resolver la red de Sevilla que se ha obtenido para este trabajo, mediante un algoritmo basado en la metaheurística de búsqueda tabú, y que va a consistir de cuatro funciones.

Para ello, en primer lugar se expone la siguiente función que será la principal, y la que llama al resto de funciones de las que consta, denominada **búsqueda_tabu_sevilla.m**.

```
%Búsqueda tabú aplicada a red de la ciudad de Sevilla
%José Javier Guerrero Herruzo - Trabajo Fin de Master
tic;

%DATOS DE LA RED
n=108; %numero de arcos del problema
nodos=65; %n° de nodos del problem

%añadimos el valor de los flujos
flujos=[1802    2690    363 2322    1307    1494    1120    1802    1011
452 2292    1443    2391    2105    1167    1934    2526    1307    1246
1288    1522    1448    1127    1385    1413    2326    1069    1386    1575
1173    1215    2161    1501    1397    1496    1496    1638    1525    1532
2317    1646    1768    1390    1049    2868    1711    1438    1760    1058
2194    2136    1243    1001    2127    1352    1438    1110    1624    1635
1290    1391    2341    1514    1058    1719    1805    1351    2082    1249
1556    1657    1178    1178    1190    1093    1100    1520    1093    552
1717    2036    1223    1710    1388    518 1179    1811    2758    1319
2406    1533    1344    1767    1631    1017    1035    1347    1268    1131
1203    1451    1131    1769    1826    1359    2500    2500    2500];

%vector origen y destino de cada arco
origen=[3   3   2   6   2   7   5   7   6   9   7   11  8   11  18  12
13  16  17  65  17  14  15  16  22  15  20  13  19  18  17  19  27  24  25
23  26  62  9   24  63  61  28  63  28  59  29  60  31  10  27  29  58  31
57  60  37  22  57  23  55  29  59  48  30  48  31  64  36  55  36  35  47
33  34  46  32  45  41  38  39  41  46  49  54  40  54  40  53  40  43  43
45  40  49  47  53  51  39  43  42  33  42  50  44  51  52];
destino=[5  1   4   1   4   4   6   8   8   5   2   11  18  12  12  9   13
16  18  16  13  15  65  14  14  21  25  22  24  21  19  20  20  26  23  24
22  25  10  10  28  62  63  27  60  29  62  23  59  26  58  30  30  48  30
59  57  21  36  55  35  60  34  58  47  33  64  32  55  39  56  37  36  46
34  35  45  33  44  44  37  38  34  49  64  49  39  53  35  52  41  52  32
40  44  50  38  50  50  46  42  51  42  41  45  51  55  56];

niter=40; %numero de iteraciones
historico=[];
tabusize1=10; %tamaño de lista tabú
listatabu1=zeros(n); %lista tabú

%generar una solucion inicial
sol=generasol(n,flujos,origen,destino,nodos);
mejor_sol=sol;
```

```
%Procedemos a evaluar la función objetivo
fo=evaluasol(sol,flujos,nodos,n,origen,destino);
mejor_fo=fo;
contadoriter=0;
for i=1:niter

[sol,listatabu1]=actualizasol(sol,flujos,n,nodos,origen,destino,listatabu1,ta
busize1);

 listatabu1=listatabu1-1;%reducir en uno todos los valores de la lista tabú
 listatabu1(find(listatabu1<0))=0;

fo=evaluasol(sol,flujos,nodos,n,origen,destino);
historico=[historico fo];
%almacenar la mejor solución
if fo>mejor_fo
    mejor_sol=sol;
    mejor_fo=fo;
    contadoriter=i;
end

end
t=toc;

t
mejor_sol
mejor_fo
contadoriter
```

A la primera función a la que se llama es a **generasol.m**, que genera un árbol aleatorio a partir de los datos introducidos en la función. Dicho árbol se codifica como en el resto del algoritmo dentro de la misma función. Proporciona por tanto una solución inicial al problema.

```
%José Javier Guerrero Herruzo - Trabajo Fin de Master
function [sol]=generasol(n,flujos,origen,destino,nodos)

    matriz_flujos=zeros(nodos);
    incidencia=zeros(nodos);
    nodousado=[];

    for i=1:n
        matriz_flujos(origen(i),destino(i))=flujos(i);
    end

    nodousado=randi([1,65]);
    for j=1:nodos-1

        %elaborar lista de candidatos a partir de los nodos ya usados
        candidatos=[];
        num=length(nodousado);
        for l=1:num
            for m=1:nodos
                if matriz_flujos(nodousado(l),m)>0
                    candidatos=[candidatos,m];
                end
                if matriz_flujos(m,nodousado(l))>0
                    candidatos=[candidatos,m];
                end
            end
        end

        %crear matriz de incidencia con la solución de un árbol aleatorio
        tam1=length(candidatos);
        tam2=length(nodousado);
        aux=0;

        for j=1:tam2
          for k=1:tam1
            if aux==0
                if (candidatos(k)~=nodousado)
                    if matriz_flujos(nodousado(j),candidatos(k))>0 && aux==0
                        incidencia(nodousado(j),candidatos(k))=1;
                        nodousado=[nodousado candidatos(k)];
                        aux=1;
                        j=tam2;
                    end
                    if matriz_flujos(candidatos(k),nodousado(j))>0 && aux==0
                        incidencia(candidatos(k),nodousado(j))=1;
                        nodousado=[nodousado candidatos(k)];
                        aux=1;
                        j=tam2;
                    end
                end
```

```
            end
          end
        end
    end

  %creación del vector solución a partir de la matriz de incidencia
  sol=zeros(1,n);
  for p=1:nodos
    for q=1:nodos
        if incidencia(p,q)==1
            for r=1:n
                if origen(r)==p
                    if destino(r)==q
                        sol(r)=1;
                    end
                end
                if destino(r)==p
                    if origen(r)==q
                        sol(r)=1;
                    end
                end

            end
        end
    end
  end
end
```

La siguiente función será la que evalúa las distintas soluciones, denominada **evaluasol.m,** en la que se pueden observar las tres partes de las que consta, tal y como se explico en el apartado 6.2.3.

```
%José Javier Guerrero Herruzo - Trabajo Fin de Master
function [fo]=evaluasol(sol,flujos,nodos,n,origen,destino)

incidencia=zeros(nodos);
ct=sum(flujos.*sol); %costes totales de los flujos
nodosdesconectados=0;

%crear matriz de incidencia para penalizar nodos sin conectar
for i=1:n
    if sol(i)==1
        x=min(origen(i),destino(i));
        y=max(origen(i),destino(i));
        incidencia(x,y)=1; %rellena matriz incidencia
    end
end
%penalizar nodos sin conectar
cp=0;
for j=1:nodos
    conecta_nodo=sum(incidencia(j,:))+sum(incidencia(:,j));
    if conecta_nodo==0
        cp=cp-100000;
        nodosdesconectados=1;
    end
end

%penalizar los nodos sin conectar con el resto del arbol
ca=0;
if nodosdesconectados==0 %solo se comprueba si todos los nodos están
conectados

   explorar=zeros(1,nodos);
   terminados=zeros(1,nodos);
   explorar(1)=1;

   while (sum(terminados)~=nodos) && (sum(terminados)<sum(explorar))
    pendientes=zeros(1,nodos);%elementos pendientes de investigar

    for i=1:nodos
       if explorar(i)==1 && terminados(i)==0
            pendientes(i)=1;
       end
    end

    analizar=find(pendientes,1,'first');
    for i=1:nodos
       if incidencia(analizar,i)==1
           if terminados(i)==0
               explorar(i)=1;
           end
       end
       if incidencia(i,analizar)==1
           if terminados(i)==0
               explorar(i)=1;
```

```
            end
        end
     end

     terminados(analizar)=1;
    end
    %comprobar si se han investigado todos los nodos, si no, hay ciclo
    if sum(terminados)<nodos
        ca=-100000;
    end
end

fo=ct+cp+ca; %función objetivo total= flujos + penalizaciones
end
```

Por último se tiene la función **actualizasol.m** que proporciona la vecindad, y que selecciona al mejor de los vecinos de una solución, a partir de los intercambios de elementos de dicha solución, tal y como se ha descrito en el apartado 6.3.2.

```
%José Javier Guerrero Herruzo - Trabajo Fin de Master
function
[mejorvecino,listatabu1]=actualizasol(sol,flujos,n,nodos,origen,destino,lista
tabu1,tabusize1)
%se empieza con un valor muy bajo del fitness
mejorfitness=-1000000;

for i=1:n-1
    for j=i+1:n
        if sol(i)~=sol(j) && listatabu1(i,j)==0
            vecino=sol;
            vecino(i)=sol(j); %intercambiamos soluciones
            vecino(j)=sol(i); %intercambiamos soluciones
            fitness=evaluasol(vecino,flujos,nodos,n,origen,destino);
            if fitness>mejorfitness
                mejorvecino=vecino;
                fil=i; %esto es para actualizar la lista tabú
                col=j; %esto es para actualizar la lista tabú
                mejorfitness=fitness;
            end
        end
    end
end

%actualizar la lista tabú
listatabu1(fil,col)=tabusize1;
end
```