Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Localización de estaciones en redes de transporte ferroviario

Autor: Marcos Acuña Pinilla
Tutor: Juan Antonio Mesa López-Colmenar

Dpto. Matemática Aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Localización de estaciones en redes de transporte ferroviario

Autor:
Marcos Acuña Pinilla

Tutor:
Juan Antonio Mesa López-Colmenar
Catedrático de Universidad

Dpto. Matemática Aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Trabajo Fin de Grado:

Localización de estaciones en redes de transporte ferroviario

Autor: Marcos Acuña Pinilla

Tutor: Juan Antonio Mesa López-Colmenar

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

A mi familia.

A Isa.

Todos tenemos sueños. Pero para convertir los sueños en realidad, se necesita una gran cantidad de determinación, dedicación, autodisciplina y esfuerzo.

"Jesse Owens"

AGRADECIMIENTOS

Llegados a este punto me gustaría agradecer a todas las personas que de una forma u otra, a lo largo de estos cinco años me han ayudado:

Gracias Isa por ser la persona que más horas me ha aguantado en la biblioteca, siempre regalándome su mejor cara y un ¡Ánimo tu puedes! Por saber llevarme en esos momentos de bajón, sacarme una sonrisa, confiar en mí y sobre todo por ser mi compañera de vida.

Gracias a mi familia por ofrecerme siempre su apoyo incondicional y creer siempre en mí. Gracias papá por ser un referente para mí en la vida. Gracias mamá por ser la persona que mejor sabe llevarme y tener un amor inmenso. Gracias hermana por compartir sonrisas que solo tú y yo entendemos.

A los cuatro, gracias por compartir conmigo vuestra felicidad y vuestro día a día.

Me gustaría agradecer también a los que empezaron siendo completos desconocidos hace cinco años, pasaron a ser compañeros y a día de hoy son grandes amigos. Gracias por tantos ratos de fatigas, bromas y risas compartidos.

Por último agradecer a mi tutor, Juan Antonio Mesa López-Colmenar, por dedicarme su tiempo con interés y ayudarme en cualquier momento.

A todos de corazón, Gracias.

Marcos Acuña Pinilla

Sevilla, 2015

RESUMEN

En este trabajo se considera el problema de localizar estaciones en una red representada mediante un grafo empotrado en el plano euclídeo. Asumiremos que viajar por la red es más rápido que viajar a través del plano de acuerdo a la distancia euclídea. En el plano se conoce la ubicación de un conjunto de puntos, que pueden representar pueblos, ciudades o zonas de transporte, entre los cuales existe una demanda de viaje codificada mediante una matriz origen-destino. Se supone que los viajeros disponen de un medio alternativo y que seleccionan el medio de transporte de acuerdo con el tiempo de viaje. Los viajeros comparan el tiempo de viaje en el modo alternativo con el del modo combinado que utiliza la red de alta velocidad y eligen el de menor tiempo. En este trabajo, de forma genérica, hemos fijado unos tiempos máximos para que el viajero escoja la red de alta velocidad

El objetivo del trabajo es localizar dos estaciones sobre la red de forma que se maximize la captura o cobetura total de pares de demanda por parte del modo combinado.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

En las últimas décadas la población urbana y metropolitana está creciendo a un ritmo elevando, provocando que cada vez existan más zonas urbanizadas en las que vivir. Por otra parte, la mayor disponibilidad de medios de transporte y el aumento, en su caso, de la velocidad hacen que la movilidad crezca.

Como consecuencia de este crecimiento, se eleva el uso de elementos de transportes de alta velocidad como trenes o metros. Este último es la manera más eficiente de desplazarse en zonas metropolitanas, ya que al hacer uso de él, evitamos que muchas personas conduzcan sus vehículos propios disminuyendo así la congestión de tráfico y como resultado la polución que se genera. Por otra parte, los trenes de alta velocidad han reducido el uso del avión en distancias medias y han generado nuevas demandas

El trabajo presentará la planificación de estos sistemas de transporte terrestre de alta velocidad. Los objetivos que perseguiremos serán: mejorar la movilidad de la población para ámbitos de ocio y laborales, reducir el tráfico, la contaminación y el consumo de energía y por último acortar los tiempos de viaje ya que se propondrá el uso de elementos de alta velocidad.

La planificación del problema depende de factores: políticos, de construcción, ingenieriles, urbanísticos, de las compañías de transportes o las necesidades de los pasajeros. Como cabe imaginar, es muy difícil poner a todos estos elementos de acuerdo. Como consecuencia de la necesidad de ese arduo pacto, es de vital importancia realizar un amplio estudio previo que englobe todos los factores. Tendremos que analizar la zona donde se implantará el sistema de transporte de alta velocidad y los aspectos fundamentales del viaje.

A partir de diferentes herramientas analíticas trataremos de averiguar la opción que permita al sistema dar servicio al mayor número de pasajeros posibles. Para ello habrá que ver dónde residen las personas y dónde se colocarán las estaciones. Un criterio podría ser la distancia entre la casa del pasajero y la estación por donde entra a la red de transporte. Si estamos trabajando en entornos con poblaciones densas buscaremos que la distancia entre ambos puntos anteriormente mencionados sea menor de 400 metros o estén a menos de 5 minutos andando. Sin embargo si estamos en poblaciones donde la densidad de residentes es menor, seguirá siendo válida si ampliamos a 1 kilómetro la distancia máxima que hay entre el hogar del pasajero y la estación en la que se monta para entrar. Otro criterio, algo mejor que el anterior, sería analizar el viaje como un par origen-destino. De esta forma no solo tendríamos en cuanta la distancia entre el hogar y la estación de entrada, sino también el trayecto o el tiempo que emplearemos en ir a la estación de salida y el necesario desde aquí hasta el punto final de destino deseado por el pasajero. De esta forma realizaríamos un análisis más amplio y efectivo del viaje total del pasajero de forma que se cubrirían mejor sus necesidades.

Una vez tengamos el estudio realizado habrá que razonar si lo mejor es construir una nueva red de transporte de alta velocidad, extender la longitud de algunas ramas de una red ya existente o

añadir nuevas ramas independientes a un sistema ya efectivo. Esta disyuntiva nos abrirá nuevamente más variantes a analizar.

Las configuraciones de los sistemas de alta velocidad que vamos a tratar en este trabajo fin de grado serán de dos tipos. La primera de ellas será un único segmento recto en el que trataremos de localizar las estaciones. El segundo escenario que nos plantearemos será una red formada por varios segmentos rectos unidos entre sí en sus extremos formando un árbol. Una vez ya tengamos construida la red, lo siguiente será localizar dentro de ese sistema un par de estaciones por las que el pasajero entrará y saldrá del sistema de forma que el viaje total (lugar de partida - estación de entrada - estación de salida – lugar de destino) sea óptimo.

2 ANTECEDENTES

Abordar el análisis, la planificación y la construcción o extensión de un sistema de alta velocidad como podría ser el caso de una red de metro, es una de las tareas más realizadas en las últimas décadas en el ámbito de los medios de transportes terrestres. Numerosos artículos, revistas y libros de investigación relacionados con el tema han sido escritos en todo el mundo, favoreciendo así el fuerte avance que se ha conseguido en los últimos 20 años. Cada documento propone un criterio diferente para la determinación de las estaciones en el sistema de alta velocidad.

El primer criterio que se consideró fue minimizar el tiempo total de viaje del pasajero, mencionado por *Vuchic and Newell* (1968). Estos analizaron el problema de determinar un número de estaciones y los espacios entre ellas para minimizar el tiempo total de transporte. El modelo se limita a las personas que se desplazan al centro de la ciudad a través de un corredor y tiene en cuenta varios aspectos realistas tales como la velocidad de acceso, tiempos de permanencia, la cinemática de los trenes, la competencia entre modos de transporte y la población de pasajeros a lo largo de la línea. Estos autores resolvieron el problema a través de un conjunto de ecuaciones en diferencias de segundo orden.

Posteriormente *Hamacher et al.* (2001) trabajó en un razonamiento relacionado con el anterior criterio. Este trataba la maximización de la reducción del tiempo de viaje cuando introducimos nuevas estaciones. Un problema similar con respecto a una línea de alta velocidad fue tratado por *Repolho et al.* (2013). Estos autores presentaron un programa que se aplica a la ubicación de las estaciones en varios pasillos de una línea de alta velocidad prevista en Portugal, la cual compite con otros modos de transporte alternativos.

La minimización del tiempo de viaje adicional que se genera por las diferentes paradas de los trenes en las nuevas estaciones fue estudiado por *Schöbel* (2002) y *Schöbel et al.* (2009), en el contexto del transporte público urbano. El mismo problema lo han estudiado *Carrizosa et al.* (2013), pero añadiéndole el análisis de la cinemática de los trenes entre paradas.

Uno de los criterios más utilizados en situaciones reales es maximizar la cobertura ya que la construcción de las estaciones es una operación compleja y sobre todo costosa. Este problema se ha estudiado en el artículo de *Laporte et al.* (2002), donde se analizan las estaciones de una línea de alta velocidad limitando las separaciones que existen entre estaciones en una arista predefinida. Para resolver este problema, los autores hacen uso de una representación del problema mediante un grafo construido ad hoc en el que aplicaron un algoritmo de camino más largo. La versión continua de este problema, cuando se fija el número de nuevas estaciones, fue estudiado por *Kranakis et al.* (2003). *Schöbel* (2005) considera el problema de maximizar la cobertura y reducir al mínimo el número de nuevas estaciones.

Vuchic (1969) y posteriormente *Laporte et al.* (2005) introdujeron un nuevo criterio más realista. Este consistía en maximizar la cantidad de pasajeros que fuesen a usar la red de alta velocidad. El contexto del problema tratado por *Vuchic* es similar al de *Vuchic y Newell* (1968), mientras que en *Laporte et al.* (2005) una línea de la red de velocidad de longitud limitada y sus estaciones son localizadas simultáneamente. Más tarde, *Körner et al*. (2014) utilizaron el mismo criterio para la ubicación de dos nuevas estaciones en los segmentos que forman las ramas de la red de alta velocidad.

Gross et al. (2009) han abordado el problema de la accesibilidad máxima, donde vamos a tener un número fijo de estaciones y exigimos minimizar la suma de las distancias a los puntos de demanda.

Por último, los problemas de minimizar la inversión económica para la construcción de las nuevas estaciones y reducir al mínimo el número de estaciones que cubran todos los puntos de demanda fue investigado por *Hamacher et al. (2001).*

Para este trabajo de fin de grado el criterio que hemos empleado ha sido el de maximizar la cobertura de viaje.

Para lograr maximizar este objetivo es conveniente hacer referencia a la consulta documental de trabajos previamente realizados que guardan relación con los objetivos propuestos en este estudio. En función de ello se menciona:

- El capítulo *The design of rapid transit network* perteneciente a el libro *Location Science* (2015) de Juan A. Mesa y Gilbert Laporte. En él se hace una introducción general al tema y sus autores destacan, entre otras cosas, tres posibles variantes de planificación: la construcción completa de una nueva red, la adhesión de una nueva línea a un sistema ya construido o la extensión de alguna de sus líneas. En este trabajo vamos a ocuparnos del caso en el que hacemos una red totalmente nueva. También recomiendan el uso imprescindible de un amplio estudio previo a la construcción.

- El artículo *A maximum trip covering location problem with an alternative mode of transportation on tree networks and segments* (2012) de Mark-Christoph Körner, Juan A. Mesa, Federico Perea, Anita Schöbel y Daniel Scholz. Este artículo es la base del proyecto realizado, ya que los autores describen la forma genérica de proceder tanto si queremos analizar una red completa formada por varias ramas o un sistema con una única arista.

- El artículo *Maximizing trip coverage in the location of a single rapid transit alignment* (2005) de Gilbert Laporte, Juan A. Mesa, Francisco A. Ortega e Ignacio Sevillano se centra exclusivamente en el caso en el que tratemos la red de transito rápido como un único segmento recto. Los autores proponen varias formar de abordar

este problema, siéndonos de utilidad algunas ideas propuestas para nuestro estudio en estas determinadas circunstancias.

3 MÉTODO DE SELECCIÓN DE LOS PUNTOS CANDIDATOS

Tras la introducción anterior, en este texto vamos explicar la metodología que hemos utilizado para seleccionar los puntos candidatos en los que construir las estaciones de nuestra red de alta velocidad.

Lo primero que vamos a hacer es aclarar y definir la notación que vamos a emplear.

3.1 Notación

- Sea: $\boldsymbol{A} = \{A_i = (a_i, b_{i)} \in \mathbb{R}^2 : i = 1, \dots, n\}$ el conjunto de los n puntos de demanda en $\mathbb{R}^2$ dotado con la distancia euclídea d_2(A,B)= $||A - B||_2$. Estos representarán las coordenadas de los puntos de demanda en el área urbana o interurbana a tratar.

- Sea: $\boldsymbol{T} = (t_{ij}) \in \mathbb{R}^{nxn}$ la matriz de demandas. De esta manera el elemento t_{ij} nos indicará el número de viajes que se darán del punto de demanda A_i al A_j y del A_j al A_i. Está matriz tendrá por tanto la diagonal y todos los elementos que queden por debajo de ella iguales a cero.

- Sea: $\boldsymbol{N}\ (V, E)$ la red representativa del sistema de alta velocidad. Cada vértice v $\in V$ representa un nodo y cada arista e $\in E$ tiene una longitud asociada l_e.

- Para cada par de puntos x, y $\in \boldsymbol{N}$ definiremos la distancia de alta velocidad como: $d_{\mathrm{N}}(x, y) = \alpha\, d(x, y), con\ \alpha \in (0,1)$. Llamaremos factor de velocidad a α, el cual nos permitirá acortar los tiempos de viaje.

- Sea: $\boldsymbol{D} = (d_{ij}) \in \mathbb{R}^{nxn}$ la matriz de distancias máximas totales permitidas para los usuarios del sistema rápido. De esta manera el elemento d_{ij}nos indicará la máxima distancia aceptada para ir del punto de demanda A_i al A_j o del A_j al A_i, ya que será la misma longitud. Está matriz tendrá por tanto la diagonal y todos los elementos que queden por debajo de ella iguales a cero.

Nuestro objetivo es ubicar dos estaciones bidimensionales $\boldsymbol{X} = \{X_1, X_2\} \subset \boldsymbol{N} \subset \mathbb{R}^2$, a lo largo de la red de alta velocidad de forma que obtengamos el mayor rendimiento posible en el sentido de que el sistema de alta velocidad capte el mayor número de viajes posible.

Para ayudar a situarnos en el problema, se puede observar la imagen posterior. En ella se simboliza un sistema de alta velocidad formado por ocho aristas representadas en negro, nueve puntos de demanda representados como un círculo verde y las dos estaciones que calcularemos tras todo el proceso, representadas como una estrella de color rojo. Se puede ver como las estaciones han de estar contenidas dentro de alguna de las aristas que constituyen la red de alta velocidad.

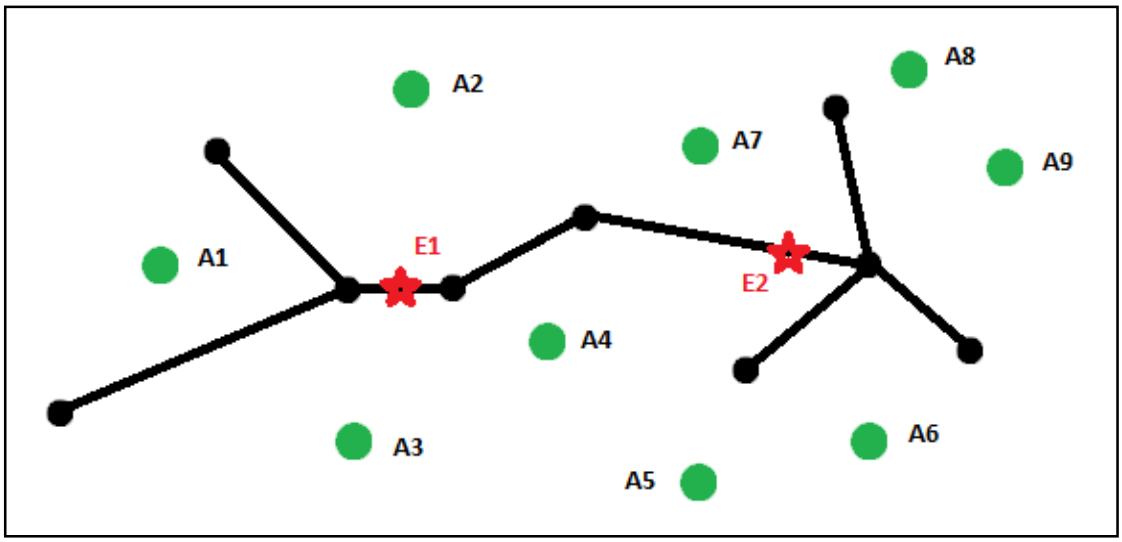

Figura 1

Si consideramos un par de puntos de demanda origen-destino (i,j) y un par de estaciones de entrada-salida (k,l), podemos expresar la longitud total que recorreremos en nuestro trayecto como:

$$h_{ij}^{kl} \coloneqq d_2(A_i, X_k) + d_N(X_k, X_l) + d_2(X_l, A_j)$$

En el caso de que el usuario busque minimizar la distancia total de su viaje, tomando como inicio de este el punto de demanda origen y como fin el punto de demanda destino, la distancia desde A_i a A_j a través del sistema de alta velocidad será:

$$f_{ij}(\boldsymbol{X}) \coloneqq min_{X_k, X_l \in X}\{d_2(A_i, X_k) + d_N(X_k, X_l) + d_2(X_l, A_j)\} = min_{X_k, X_l \in X}\, h_{ij}^{kl}$$

Siendo X el conjunto de todos los puntos de la red

Dado d_{ij} con $0 \leq d_{ij} \leq d_2(A_i, A_j)$, se considerará que un par de puntos de demanda origen-destino (i,j), serán cubiertos por las estaciones $X_k, X_l \in N$ solo si:

$$f_{ij}(\boldsymbol{X}) \leq d_{ij}$$

Buscaremos maximizar el número de pares de puntos de demanda que serán cubiertos.

Denotaremos por $\boldsymbol{F}(\boldsymbol{X}) = \Sigma_{(i,j):f_{ij}(X) \leq d_{ij}} t_{ij}$, al sumatorio de todas las demandas (t_{ij}) para las cuales el modo combinado que usa la red de alta velocidad es preferible frente al modo único o privado.

Resumiendo, el problema considerado en este texto puede escribirse como:

Dada una red de alta velocidad ***N***, unos puntos de demanda ***A***, las demandas de los viajes ***T*** y los niveles de aceptación ***D***, hay que encontrar dos estaciones $\{X_1, X_2\} \subset \boldsymbol{N}$ de forma que el número

de viajes cubiertos ***F***, se maximice. Abreviando:

$$máx_{\boldsymbol{X}\subset \boldsymbol{N}}\ F(\boldsymbol{X}).$$

3.2 Caso de una única arista:

En esta sección vamos a tratar de resolver el problema de localizar dos nuevas estaciones en una arista recta de longitud ***L***. Vamos a establecer por facilidad que la primera estación ha de estar más a la izquierda que la segunda de ellas, estando ambas contenidas dentro de la línea recta. Trataremos la arista recta solo con una coordenada de forma que se sitúe horizontal (todas las alturas serán 0) para simplificar el proceso.

Como consecuencia, las estaciones tendrán estas coordenadas:

$$X_1 = (x_1,0) \text{ y } X_2 = (x_2,0), \text{ con } 0 \leq x_1 \leq x_2 \leq \boldsymbol{L}$$

Vamos a establecer $C_{ij} \coloneqq \{(x_1, x_2) \in [0, L] \times [0, L']: f_{ij}(x_1, x_2) = d_{ij}\}$ como la curva de nivel que limita el conjunto de nivel en el que el par origen-destino que va del punto de demanda *i* al j queda cubierto. Si la intersección de dos curvas de nivel, C_{ij} y $C_{i'j'}$, no es vacía significará que al menos existen un par de puntos en los que localizar las estaciones de forma que se cubren las demandas de ambos pares de origen-destino, (A_i, A_j) y $(A_{i'}, A_{j'})$.

Llamaremos $C = \bigcup_{(i,j)\neq(i',j')} C_{ij} \cap C_{i'j'}$ a la unión de intersecciones de pares de conjuntos distintos.

De esta manera pasará algunas de estas tres opciones:

1. *C* contendrá al menos una solución óptima del problema.
2. Existe un par de origen-destino (i,j) en el que cualquier punto del conjunto C_{ij} es óptimo.
3. Cualquier punto del conjunto [0, L] x [0, L'] es óptimo.

La ecuación que gobierna la intersección de las curvas de nivel C_{ij} y $C_{i'j'}$ tiene como máximo 16 raíces, debido a que el teorema de Bezout demuestra que una ecuación de cuarto grado en dos variables tiene 4^2 raíces, de las cuales se puede demostrar que 4 se van al infinito mediante un análisis posterior. Así trabajaremos con 12 raíces que podrán ser múltiples o incluso complejas. Estas últimas tampoco nos servirán ya que no tendrían sentido físico.

3.2.1 Algorimo genérico 1: Par de máxima cobertura en una arista

Datos de entrada:

- Un conjunto de puntos *A* que representan los puntos de demanda.
- Una arista recta de longitud *L*.
- Una matriz de demanda *T*.
- Un factor de alta velocidad $\alpha \in (0, 1)$.
- Una matriz de distancias asumibles *D*.

Procedimiento:

1. Inicializar al vacío el conjunto C.
2. Para cada par de puntos de demanda origen-destino (i,j) calcular su curva de nivel C_{ij}.

3. Para cada par de pares de puntos de demanda origen-destino (i,j) y (i',j') calcular la intersección de sus curvas de nivel C_{ij} y $C_{i'j'}$. Establecer *C* como la unión de todas las intersecciones entre par de pares de puntos de demanda.

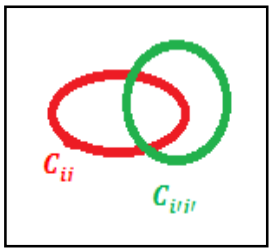

Figura 2

4. Si ambas curvas son distintas del vacío pero no intersecan hay que aplicar este test de inclusión:
    a. Si la curva C_{ij} está contenida en la curva $C_{i'j'}$ hay que añadir un punto de C_{ij} al conjunto *C*.

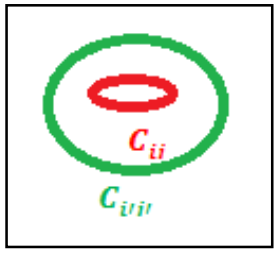

Figura 3

    b. Si la curva $C_{i'j'}$ está contenida en la curva C_{ij} hay que añadir un punto de $C_{i'j'}$ al conjunto *C*.

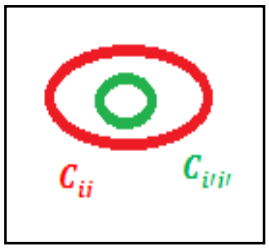

Figura 4

    c. Si no es ningún caso anterior, hay que añadir un punto de cada curva C_{ij} y $C_{i'j'}$ al conjunto *C*.

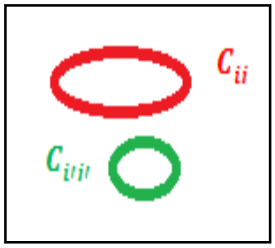

Figura 5

5. Para finalizar:
    a. Si el conjunto *C* es distinto del vacío hay que evaluar la función objetivo en cada punto (x_1, x_2) de los almacenados en el conjunto *C* y elegir el que proporciona el máximo de la función. Establecer X_1 y X_2 como las soluciones finales.
    b. Si no, elegir cualquier par de estaciones pertenecientes a L, X_1 y X_2.

Datos de salida:

- Las dos estaciones X_1 y X_2 pertenecientes a *L* que resuelven el problema.

3.3 Caso de red tipo árbol:

En esta sección vamos a tratar de resolver el problema de localizar 2 nuevas estaciones en una red tipo árbol *T*. Haremos uso del caso anterior en el que colocamos dos estaciones en la misma arista y además estudiaremos el caso en el que las estaciones estén localizadas en distintas aristas rectas e_p y e_q con p. Para este proceso las coordenadas de las estaciones serán dos:

$$X_1 = (x_1^1, x_1^2) \text{ y } X_2 = (x_2^1, x_2^2)$$

Para ir de X_1 y X_2 solo existirá un camino $p(X_1, X_2)$ y será a través de las aristas que forman el árbol.

Z es un camino convexo combinación de X_1 y X_2 a lo largo de $p\ (X_1, X_2)$ con respecto a $\lambda \in [0, 1]$ si:

- $Z \in p\ (X_1, X_2)$ y
- $d_\tau(X_1, X_2) = \lambda\ d_\tau(X_1, X_2)$

En este caso tendremos:

$$d_\tau(Z, X_2) = (1 - \lambda)\ d_\tau(X_1, X_2)$$

Así, el segmento que une un par de puntos X_1 y X_2 en las aristas $e_p\ x\ e_q$ lo denominaremos segmento de camino, ps:

$$ps\ (X_1, X_2) = \cup_{\lambda \in [0,1]} \{Z \in e_p\ x\ e_q : d(X_1, Z) + d(Z, X_2) = d(X_1, X_2) = \lambda d(X_1, X_2)\}$$

Sabremos si una función es un camino convexo en $e_p\ x\ e_q$ si:

$$f(Z) \leq \lambda f(X_1) + (1 - \lambda) f(X_2), \forall\ Z \in ps\ (X_1, X_2), \forall\ \lambda \in [0, 1]$$

De esta manera vamos a ir del punto de demanda inicial, A_i, al lugar de la primera arista, e_p, donde se encuentre la primera estación, X_1. Después iremos a la siguiente estación, X_2, que estará en algún punto de otra arista distinto al primero, e_q, recorriendo las aristas intermedias que existan entre e_p y e_q. Terminaremos recorriendo el tramo que va desde la estación de salida hasta el punto de demanda final, A_j. Diremos que la primera arista se recorre solo desde el punto de la estación de entrada hasta su final y la arista por donde se sale, solo se transita desde su inicio hasta la estación de salida.

Parametrizaremos dichas aristas de esta forma:

$$e_p = \{X = g_p(\lambda) \coloneqq T_p + \lambda_1 \, x \, s_p : \lambda_1 \in [0,1]\},$$
$$e_q = \{X = g_q(\lambda) \coloneqq T_q + \lambda_2 \, x \, s_q : \lambda_2 \in [0,1]\}$$

Con T_p, T_q, s_p, s_q $\in \mathbb{R}^2$ y $g_p(0)$=T_p , $g_q(0)$=T_q . Como es previsible T_p, T_q responden a los puntos iniciales de las aristas y s_p, s_q a las direcciones de los ejes e_p y e_q respectivamente.

Definimos la curva de nivel $C_{ij}^{kl} \coloneqq \{(x_1, x_2) \in [0, L]x[0, L']: f_{ij}(x_1, x_2) = d_{ij}\}$ que delimita el conjunto de nivel en el que el par origen-destino (i,j) está cubierto cuando la estación de acceso al sistema rápido es k y la de la de egreso es l. Si la intersección de dos curvas de nivel, C_{ij}^{kl} y $C_{i'j'}^{k'l'}$, no es vacía, significará que al menos existen un par de estaciones que cubren las demandas de ambos pares de origen-destino, (A_i, A_j) y $(A_{i'}, A_{j'})$. Llamaremos C a la unión de intersecciones de pares de conjuntos. De esta manera pasará algunas de estas tres opciones:

1. *C* contiene al menos una solución óptima del problema.
2. Existe un par de origen-destino (i,j) y (k,l) $\in \{(1,2),(2,1)\}$ en el que cualquier punto del conjunto es óptimo.
3. Cualquier punto del conjunto [0,L]x[0,L'] es óptimo.

La ecuación que gobierna la intersección de las curvas de nivel C_{ij}^{kl} y $C_{i'j'}^{k'l'}$ tiene como máximo 16 raíces, debido a que el teorema de Bezout demuestra que una ecuación de cuarto grado en dos variables tiene 4^2 raíces, de las cuales se puede demostrar que 4 se van al infinito mediante un análisis posterior. Así trabajaremos con 12 raíces que podrán ser múltiples o incluso complejas. Estas últimas tampoco nos servirán ya que no tendrían sentido físico.

El problema en este tipo de grafos vamos a necesitar tres tipos de algoritmos. El algoritmo1, el que llamaremos algoritmo 2 que describirá la forma de proceder para el proceso de dos estaciones situadas en dos aristas diferentes y el algoritmo 3 en el que trataremos una red tipo árbol completa haciendo uso de los dos anteriores.

3.3.1 Algorimo genérico 2: Par de máxima cobertura con puntos situados en diferentes aristas.

Datos de entrada:

- Un conjunto de puntos *A* que representan los puntos de demanda.
- Un par de aristas rectas $e_p \; y \; e_q$.
- Una matriz de demanda *T*.
- Un factor de alta velocidad α Є (0, 1).
- Una matriz de distancias asumibles *D*.

Procedimiento:

1. Inicializar al vacío el conjunto C.
2. Para cada par de puntos de demanda origen-destino (i,j) y cada (k, l) $\in \{(1,2),(2,1)\}$ calcular su curva de nivel C_{ij}^{kl}.
3. Para cada par de pares de puntos de demanda origen-destino $\{(i,j),(i',j')\}$ y $\{(k,l),(k',l')\}$ calcular la intersección de sus curvas de nivel C_{ij}^{kl} y $C_{i'j'}^{k'l'}$. Establecer *C* como la unión de todas las intersecciones entre par de pares de puntos de demanda.

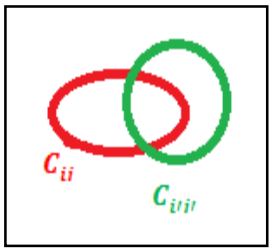

Figura 6

4. Si ambas curvas son distintas del vacío pero la intersección de ambas no se produce hay que aplicar este test de inclusión:
   a. Si la curva C_{ij}^{kl} está contenida en la curva $C_{i'j'}^{k'l'}$ hay que añadir un punto de C_{ij}^{kl} al conjunto *C*.

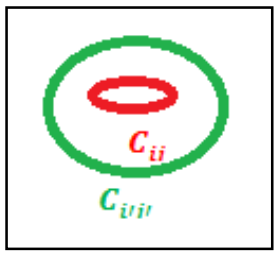

Figura 7

   b. Si la curva $C_{i'j'}^{k'l'}$ está contenida en la curva C_{ij}^{kl} hay que añadir un punto de $C_{i'j'}^{k'l'}$ al conjunto *C*.

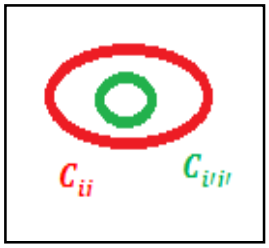

Figura 8

c. Si no es ningún caso anterior, hay que añadir un punto de cada curva C_{ij}^{kl} y $C_{i'j'}^{k'l'}$ a la curva *C*.

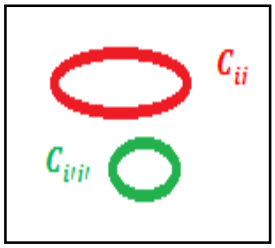

Figura 9

5. Para finalizar:
   a. Si el conjunto *C* es distinto del vacío hay que evaluar la función objetivo, $\boldsymbol{F}(X) = \Sigma_{(i,j):f_{ij}(X)\leq d_{ij}} t_{ij}$, en cada punto (x_1, x_2) de los almacenados en el conjunto *C* y elegir un máximo. Establecer X_1 y X_2 como las soluciones finales.
   b. Si no, elegir cualquier par de estaciones pertenecientes a L, X_1 y X_2.

Datos de salida:

- Las dos estaciones X_1 y $X_2 \in e_p \, x \, e_q$ que resuelven el problema.

3.3.2 Algorimo genérico 3: Par de máxima cobertura en una red tipo árbol

Datos de entrada:

- Un conjunto de puntos *A* que representan los puntos de demanda.
- Un red tipo árbol formada por varias aristas rectas.
- Una matriz de demanda *T*.
- Un factor de alta velocidad α Є (0, 1).
- Una matriz de distancias asumibles *D*.

Procedimiento:

1. Para cada arista e_p aplicar el Algoritmo 1. Establecer (X_1^p, X_2^p) como la solución obtenida.
2. Para cada par de ejes distintos $e_p \, y \, e_q$ con p < q aplicar el Algoritmo 2. Establecer (X_1^{pq}, X_2^{pq}) como la solución obtenida.
3. Obtener la mejor solución X_1 y X_2 de todas las candidatas (X_1^p, X_2^p) y (X_1^{pq}, X_2^{pq}).

Datos de salida:

- Las dos estaciones X_1 y $X_2 \in$ a la red tipo árbol, *T*, que resuelven el problema.

4 EXPLICACIÓN DEL CÓDIGO MATLAB PARA EL CASO DE UNA RED FORMADA POR UNA ÚNICA ARISTA

El código que se va a explicar a continuación se utilizará para resolver el problema de la localización óptima de estaciones en redes de transporte ferroviario, en el caso de que el recorrido sea un segmento.

Para comenzar el código necesitaremos introducir como entrada algunos elementos que pasamos a describir:

- A: matriz formada por tantas filas como puntos de demanda tratemos y por 2 columnas, que serán las coordenadas x e y de cada uno de los puntos de demanda que estén en el plano.
- L: longitud del segmento dentro del cual van a estar las dos nuevas estaciones.
- D: matriz de distancias máximas permitidas entre puntos de demanda. Esta matriz será de dimensiones [número de puntos de demanda x número de puntos de demanda]. La matriz D solo será distinta de 0 en los elementos que queden por encima de la diagonal principal ya que por la naturaleza del problema el par "ij" está cubierto si y sólo si lo está el "ji". De esta forma, el elemento "D (i, j)" nos indicará la distancia que podría existir entre el punto de demanda "i" y el "j" que será la misma que habrá entre en punto de demanda "j" y el "i".
- α: factor de velocidad comprendido entre 0 y 1.
- T: matriz de demanda entre puntos de demanda. Esta matriz será de dimensiones [número de puntos de demanda x número de puntos de demanda].
  Por el mismo motivo de simetría que en D, podemos agrupar las demandas de i a j con las de j a i. Debido a esto, la matriz T solo será distinta de 0 en los elementos que queden por encima de la diagonal principal. De esta forma el elemento "T (i, j)" nos indicará la demanda que va a existir para el trayecto entre el punto de demanda "i" y el "j" más la que habrá entre en los puntos de demanda "j" y el "i".

Lo primero que tenemos que determinar es el número de puntos de demanda que vamos a tratar y sus coordenadas. Para ello vamos a calcular el tamaño de la matriz A mediante el comando **size**. Guardaremos el número de filas en una variable, la cual nos dirá el número de puntos de demanda, y en otra el número de coordenadas (que serán dos al tratarse de un problema bidimensional). Acto seguido representaremos mediante el comando **meshgrid** una gráfica que irá desde 0 hasta la longitud del segmento en ambos ejes del plano. Así podremos representar tanto el segmento como los puntos de demanda.

A continuación vamos a proceder a la representación de las curvas de nivel de cada par de puntos de

demanda. A pesar de no saber aún dónde estarán las dos estaciones deseadas usaremos los valores obtenidos como salida del comando **meshgrid** para poder realizar la representación. Necesitaremos crear un arreglo (f). Para ello tendremos que obtener una variable en la que almacenaremos el número de elementos que va a necesitar la tercera componente del arreglo, es decir, tantos como pares de puntos de demanda. Una vez conocida su dimensión ya la podremos inicializar.

El arreglo tendrá tres dimensiones. Las dos primeras serán usadas para indicar los valores '"x" e "y" de las coordenadas de las curvas de nivel, y la última como indicador del par de puntos de demanda. Para el índice del par de puntos de demanda nos ayudaremos de un elemento auxiliar que iremos incrementando cada vez que se guarde una operación entre pares de puntos de demanda. A modo de ejemplo, f (xa, xb, k) indicará el valor de la función distancia del par de índice k cuando las estaciones se localizan en los puntos x e y, cuyos índices son respectivamente xa y xb.

Haremos el cálculo de la distancia para lo que necesitaremos dos bucles **for**, uno por cada punto de demanda. Como vamos a asumir que la distancia para ir del punto de demanda "i" al punto de demanda "j" es la misma que hay para desplazarnos desde el punto de demanda "j" hasta el punto de demanda "i", sólo la calcularemos una vez (en un solo sentido del trayecto). Para ello impondremos la condición de que el primer punto de demanda del par tenga un índice menor que el del segundo. Lo siguiente que tenemos que analizar son los puntos de acceso y salida de la red, respectivamente. Para ello distinguiremos si la coordenada "x" del primer punto de demanda es menor o mayor que la del segundo. Supondremos que el viajero se montará siempre en el punto de demanda con menor coordenada "x" en el plano y saldrá por el otro. La distancia la calcularemos como la suma de tres sub distancias:

1. La existente entre el punto de demanda desde el que entramos y la estación en la que nos vamos a montar.
2. La distancia que hay entre las estaciones de inicio y fin del trayecto (multiplicada por el coeficiente de velocidad, ya que al ir montado sobre el medio de transporte suponemos que la velocidad del viaje será mayor).
3. La que hay entre la estación en la que nos vamos a bajar y el punto de demanda al que queremos ir.

Lo siguiente que realizaremos será representar las curvas de nivel que nos generarán dichas funciones de distancias. Vamos a imponer que cada distancia entre par de puntos de demanda no pueda ser mayor a un longitud dada (impuesta en la matriz D). Así veremos las curvas de nivel que podríamos permitirnos realmente. De esta forma toda el área que quede dentro de la curva de nivel de la función va a cumplir que su distancia será menor que la asignada como máxima permitida. También representaremos el segmento en el que tienen que estar las dos estaciones nuevas que queremos localizar.

Ahora vamos a pasar al cálculo de todas las intersecciones posibles que existan entre cada par de pares de puntos de demanda. Gráficamente esto sería encontrar todos los puntos dónde se corten las dos curvas de nivel, generadas por cada uno de los pares de puntos de demanda. Así nos aseguraremos de que esos puntos de corte cumplen que la distancia es menor que el valor máximo preestablecido en cada caso, de forma que podríamos validar ambos trayectos.

Lo primero que tenemos que hacer es saber el número de iteraciones que existirán teniendo en cuenta cada par de pares posibles. Para esto usaremos una variable auxiliar que iremos incrementando y que se encontrará dentro de 4 bucles **for**. Con los cuatro bucles podremos conseguir las distintas combinaciones entre cada par de pares de puntos de demanda. De esta forma ya tenemos el número de iteraciones que realizará esta parte del código.

En este momento creamos otro arreglo (mat) que estará formada por tantos planos como iteraciones

hayamos previsto con anterioridad. La idea es almacenar en cada plano todos los cortes que se produzcan entre las curvas de nivel de cada par de pares de puntos de demanda. Por eso en cada plano del arreglo tendremos una matriz de 12 filas y 6 columnas. Las 12 filas se deben a que como máximo van a existir 12 cortes reales y las columnas las usaremos para almacenar la información. En las 4 primeras columnas vamos a guardar los índices de los 4 puntos de demanda que estamos combinando en esa iteración, mientras que en las dos últimas vamos a guardar las posibles soluciones. Estas serán las candidatas a ser las dos estaciones que deseamos localizar.

A continuación, nuevamente vamos a realizar los mismos bucles y condiciones que hemos usado para saber el número de iteraciones necesarias. Dentro de ellos, esta vez mediante el comando **solve**, realizaremos la intersección del par de pares de puntos de demanda y guardaremos la solución calculada. Como sabemos que sólo nos interesan las soluciones que sean reales, vamos a pasarle un filtro de forma que descartemos cualquier posible estación que contenga algún elemento imaginario. Una vez depurado, guardamos cada elemento en su fila y columna correspondiente dentro del arreglo.

Existe la opción de que las curvas de nivel de pares de puntos de demanda no se corten. Gráficamente esto podría interpretarse de dos maneras:

1. La curva de nivel de un par de puntos de demanda está dentro de la otra o viceversa.
2. Ni se cortan ni están contenidas una dentro de la otra. Son independientes.

A continuación en el código vamos a tratar de solucionar el problema que aportarían dichas opciones. Comenzaremos detectando en qué plano del arreglo (mat) no hay corte entre pares de curvas. Esto lo realizaremos preguntando si la columnas 5 y 6 son iguales a 0. Ahora vamos a comprobar si nos encontramos en la primera opción. Para ello vamos a hacer un test de inclusión. Tomaremos como punto de partida un valor que pueda ser válido para la primera estación, de 0 a L. A partir de este calculamos, mediante la ecuación que define la primera curva de nivel, la otra estación usando el comando **solve**. Deberemos asegurarnos que el segundo punto calculado no tiene elementos imaginarios, es mayor que el primer y menor que la longitud del segmento en el que queremos colocar las dos estaciones nuevas. Sólo si pasa esta criba, comprobaremos si los puntos calculados de la primera curva de nivel se encuentran dentro de la otra simplemente introduciéndolos en la ecuación de esta última curva. En caso de que se cumpla este test de inclusión almacenaremos las estaciones candidatas en las columnas 5 y 6 del arreglo (mat). Este proceso se repetirá variando el punto de partida (estación 1) hasta que se encuentre un par de puntos válidos. Si no se encuentran, significará que una curva no está dentro de la otra y por tanto habrá que hacer la comprobación de forma viceversa.

Si tenemos un resultado negativo en las dos comprobaciones anteriores, nos hallaremos en la opción 2. En este caso, operando de forma análoga a la opción 1, vamos a calcular un punto perteneciente a cada una de las curvas de nivel de los pares de punto de demanda y las añadiremos como posibles soluciones a probar.

En este momento ya tenemos almacenadas en el arreglo (mat) todas las candidatas a soluciones en las columnas 5 y 6 de los diferentes planos. Tan solo nos queda evaluar nuestra función objetivo, es decir, evaluar cuál de los posibles pares de estaciones candidatas nos aportaría un viaje óptimo (una mayor demanda). Lo primero que tenemos que hacer es saber el número de iteraciones que existirán teniendo en cuenta cada par de pares posibles. Para esto usaremos una variable auxiliar que iremos incrementando y que se encontrará dentro de 4 bucles for. Con los cuatro bucles podremos conseguir las distintas combinaciones entre par de pares de puntos de demanda. De esta forma ya tenemos el número de iteraciones que realizará esta parte del código.

Nuevamente vamos a crear un arreglo (sol) que tendrá tantos planos como posibles soluciones. Cada plano tendrá 1 fila y 3 columnas, en las cuales guardaremos: estación 1, estación 2 y el valor que adquiere nuestra función objetivo para esas estaciones. Ahora pasamos a realizar un barrido por todos los planos, cada uno con una solución candidata asociada. En cada iteración vamos a comprobar si esa solución nos permitiría encontrarnos dentro de la curva de nivel asociada a cada uno de los pares de puntos de demanda. En caso afirmativo eso implicaría que esa solución nos permitiría validar el traslado entre ese par de puntos de demanda y añadiríamos su demanda asociada. Tras calcular la función de demandas de cada candidata, la solución final será aquella que nos permita cubrir la solicitud total mayor.

Vamos a dibujar todas las candidatas remarcando la solución final para tener una visualización global del problema. Para finalizar el código devolveremos como salidas de la función las dos soluciones obtenidas.

5 EXPLICACIÓN DEL CÓDIGO MATLAB PARA EL CASO DE UNA RED FORMADA POR DOS ARISTAS CONTIGUAS

El código que se va a explicar a continuación se utilizará para resolver el problema de la localización óptima de estaciones en redes de transporte ferroviario, en el caso de que las estaciones estén situadas en dos aristas distintas.

Para comenzar el código necesitaremos introducir como entrada algunos elementos que pasamos a describir:

- A: matriz formada por tantas filas como puntos de demanda tratemos y por 2 columnas, que serán las coordenadas "x" e "y" de cada uno de los puntos de demanda que estén en el plano.
- p1: coordenadas bidimensionales del punto inicial de la primera arista.
- d1: vector con la dirección que tomará la primera arista. A lo largo del código esta dirección va a ir multiplicada por un coeficiente, lambda1, que tomará valores entre 0 y 1. Cuando tome el valor 0 nos encontramos en el punto inicial de la arista mientras que si toma el valor 1 estaremos en el punto final del mismo.
- p2: coordenadas bidimensionales del punto inicial de la segunda arista.
- d2: vector con la dirección que tomará la segunda arista. Esta dirección va a ir multiplicada por un coeficiente lambda2, que tomará valores entre 0 y 1. Cuando toma el valor 0 nos encontramos en el punto inicial de la arista mientras que si toma el valor 1 estaremos en el punto final del mismo.
- D: matriz de distancias máximas permitidas entre puntos de demanda. Esta matriz será de dimensiones [número de puntos de demanda x número de puntos de demanda]. La matriz D sólo será distinta de 0 en los elementos que queden por encima de la diagonal principal ya que por la naturaleza del problema el par "ij" está cubierto si y sólo si lo está el "ji". De esta forma, el elemento "D(i,j)" nos indicará la distancia que podría existir entre el punto de demanda "i" y el "j" que será la misma que habrá entre en punto de demanda "j" y el "i".
- α: factor de velocidad comprendido entre 0 y 1.
- T: matriz de demanda entre puntos de demanda. Esta matriz será de dimensiones [número de puntos de demanda x número de puntos de demanda]. Por el mismo motivo de simetría que en D, podemos agrupar las demandas de i a j con las de j a i. Debido a esto, la matriz T solo será distinta de 0 en los elementos que queden por encima de la diagonal principal. De esta forma el elemento "T (i,j)" nos indicará la demanda que va a existir para el trayecto entre el punto de demanda "i" y el "j" más la que habrá entre en los puntos de demanda "j" y el "i".

A lo largo del código vamos a estudiar en paralelo las dos opciones de viaje posibles que tendríamos utilizando los puntos de demanda del conjunto A, i y j, como acceso y salida de la red respectivamente. Por simplicidad a lo largo de la explicación, llamaremos "opción a" a la alternativa en la que el viajero partirá desde "i", entrará al sistema por la estación que esté contenida en la primera arista ("e1") y salga del mismo por la que se sitúe dentro de la segunda arista ("e2") para acabar en el punto de demanda "j". Por el contrario, llamaremos "opción b" al caso en el que partiendo del punto "i" y terminando en el punto "j" accedamos al sistema de velocidad por la estación "e2" y salgamos por "e1".

Lo primero que tenemos que determinar es el número de puntos de demanda que vamos a tratar y sus coordenadas. Para ello vamos a calcular el tamaño de la matriz A mediante el comando **size**. Guardaremos el número de filas en una variable, la cual nos dirá el número de puntos de demanda, y en otra el número de coordenadas (que serán dos al tratarse de un problema bidimensional). Ahora vamos a calcular las longitudes de las dos aristas (L1 y L2) y reduciremos el número de los decimales con los que trataremos las longitudes para evitar un futuro colapso del código en los bucles. Acto seguido representaremos mediante el comando **meshgrid** una gráfica que irá desde 0 hasta la suma de las longitudes de las dos aristas en ambos ejes del plano. Así podremos representar las dos aristas independientemente de la orientación que tengan.

A continuación vamos a proceder a la representación de las curvas de nivel de cada par de puntos de demanda para las dos opciones anteriormente detalladas. A pesar de no saber aún dónde estarán las dos estaciones deseadas usaremos los valores obtenidos como salida del comando **meshgrid** para poder realizar la representación. Necesitaremos crear dos arreglos (f12 y f21), estando f12 asociado a la "opción a" y f21 a la "opción b". Para ello tendremos que obtener una variable en la que almacenaremos el número de elementos que va a necesitar la tercera componente del arreglo, es decir, tantos como pares de puntos de demanda. Una vez conocida su dimensión ya la podremos inicializar.

Los arreglos tendrán tres dimensiones cada uno. Las dos primeras serán usadas para indicar los valores '"x" e "y" de las coordenadas de las curvas de nivel, y la última como indicador del par de puntos de demanda. Para el índice del par de puntos de demanda nos ayudaremos de un elemento auxiliar que iremos incrementando cada vez que se guarde una operación entre pares de puntos de demanda. A modo de ejemplo, f12 (xa, xb, k) indicará el valor de la función distancia del par de índice k para la "opción a" cuando las estaciones se localizan en los puntos x e y, cuyos índices son respectivamente xa y xb.

Haremos el cálculo de la distancia para lo que necesitaremos dos bucles **for**, uno por cada punto de demanda. Como vamos a asumir que la distancia para ir del punto de demanda "i" al punto de demanda "j" es la misma que hay para desplazarnos desde el punto de demanda "j" hasta el punto de demanda "i", sólo la calcularemos una vez (en un solo sentido del trayecto). Para ello impondremos la condición de que el primer punto de demanda del par tenga un índice menor que el del segundo. Lo siguiente que tenemos que analizar es por cuál de los dos puntos de demanda se montará el usuario y por cuál se bajará. Aquí es donde distinguiremos entre las dos opciones y usaremos los dos arreglos. El primero de ellos (f12) lo usaremos en el caso de que partamos del punto de demanda "i" y entremos a la red por la estación 1, la que está en la primera arista. Recorreremos lo que resta de la primera arista y el tramo que va desde el inicio del segundo hasta donde se encuentre la estación 2 para acabar llegando al punto de demanda "j". El segundo arreglo (f21) lo vamos a usar cuando partamos del punto de demanda "i" y entremos a la red por la estación 2, la que está en la segunda arista. Ahora recorreremos el tramo que va desde la estación 2 hasta el inicio de la segunda arista y el tramo que va desde el final del primero hasta donde se encuentre la estación 1, por la que saldremos, para acabar llegando al punto de demanda "j". Calcularemos la distancia como la suma de tres sub

distancias:

4. La existente entre el punto de demanda desde el que salimos y la estación en la que nos vamos a montar.
5. La distancia que hay entre las estaciones de inicio y fin del trayecto recorriendo las aristas (multiplicada por el coeficiente de velocidad, ya que al ir montado sobre el medio de transporte suponemos que la velocidad del viaje será mayor).
6. La que hay entre la estación en la que nos vamos a bajar y el punto de demanda al que queremos ir.

Lo siguiente que realizaremos será representar las curvas de nivel que nos generarán dichas funciones de distancias. Por comodidad y sencillez colorearemos las generadas por el arreglo f12 de color verde (opción a) mientras que las de f21 irán en rojo (opción b). Vamos a imponer que cada distancia entre par de puntos de demanda no pueda ser mayor a un longitud dada (impuesta en la matriz D) pudiendo así ver las curvas de nivel que podríamos permitirnos realmente. De esta forma toda el área que quede dentro de la curva de nivel de la función va a cumplir que su distancia será menor que la asignada como máxima permitida.

Ahora vamos a pasar al cálculo de todas las intersecciones posibles que existan entre cada par de pares de puntos de demanda. Gráficamente esto sería encontrar todos los puntos dónde se corten las dos curvas de nivel, generadas por cada uno de los pares de puntos de demanda. Así nos aseguraremos de que esos puntos de corte cumplen que la distancia es menor que el valor máximo preestablecido en cada caso, de forma que podríamos validar ambos trayectos. Nuevamente seguiremos trabajando las dos opciones simultáneamente.

Lo primero que tenemos que hacer es saber el número de iteraciones que existirán teniendo en cuenta cada par de pares posibles. Para esto usaremos una variable auxiliar que iremos incrementando y que se encontrará dentro de 4 bucles **for**. Con los cuatro bucles podremos conseguir las distintas combinaciones entre par de pares de puntos de demanda. De esta forma ya tenemos el número de iteraciones que realizará esta parte del código.

En este momento creamos otro arreglo (mat) que estará formado por tantos planos como iteraciones hayamos previsto con anterioridad. La idea es almacenar en cada plano todos los cortes que se produzcan entre las curvas de nivel de cada par de pares de puntos de demanda. Por eso en cada plano del arreglo tendremos una matriz de 24 filas y 6 columnas. Las 24 filas se deben a que para esta parte del código vamos a usar un solo arreglo para las dos opciones para evitar repetir bucles innecesarios. Como máximo van a existir 24 cortes reales, 12 por cada opción. En las filas 1 a 12 almacenaremos los cortes producidos en la situación que refleja la "opción a", mientras que en las filas 13 a 24 vamos a almacenar los producidos en el caso de la "opción b". Las columnas las usaremos para almacenar la información. En las 4 primeras columnas vamos a guardar los índices de los 4 puntos de demanda que estamos combinando en esa iteración, mientras que en las dos últimas vamos a guardar las posibles soluciones. Estas serán las candidatas a ser las dos estaciones que deseamos localizar.

A continuación, nuevamente vamos a realizar los mismos bucles y condiciones que hemos usado para saber el número de iteraciones necesarias. Dentro de ellos, esta vez mediante el comando **solve**, realizaremos la intersección del par de pares de puntos de demanda y guardaremos la solución calculada. Primeros haremos la opción a y después la b. Como sabemos que sólo nos interesan las soluciones que sean reales, vamos filtrar las soluciones imponiendo las condiciones de resolución **Real** de forma que descartemos cualquier posible estación que contenga algún elemento imaginario ya que carecería de sentido físico. También descartaremos cualquier opción que sea menor que 0 o mayor que 1 ya que estamos calculando los coeficientes lamba1 y lambda2 que van multiplicando a

las direcciones de las aristas. Así ya tendríamos definido exactamente donde están las estaciones. Guardamos cada elemento en su fila y columna correspondiente dentro del arreglo.

Exist

e la opción de que las curvas de nivel de pares de puntos de demanda no se corten. Gráficamente esto podría interpretarse de dos maneras:

3. La curva de nivel de un par de puntos de demanda está dentro de la otra o viceversa.
4. Ni se cortan ni están contenidas una dentro de la otra. Son independientes.

A continuación en el código vamos a tratar de solucionar el problema que aportarían dichas opciones. Debido a la gran variedad de casos posibles haremos uso de diferentes *flags* con los que indicaremos si el caso que estamos analizando ha sido tratado o no. Comenzaremos detectando en qué plano del arreglo (mat) no hay corte entre pares de curvas. Esto lo realizaremos preguntando si la columnas 5 y 6 son iguales a 0. Ahora vamos a comprobar si nos encontramos en la primera opción. Para ello vamos a hacer un test de inclusión. Tomaremos como punto de partida un valor de lambda1 que pueda ser válido para la primera estación, de 0 a 1, de forma que esta se encuentre dentro de la primera arista. A partir de este calculamos, mediante la ecuación que define la primera curva de nivel, el otro coeficiente lambda2 que nos ubica la segunda estación dentro de la segunda arista. Para ello usaremos el comando **solve**. Deberemos asegurarnos que el segundo punto calculado no tiene elementos imaginarios, es menor que 1 y mayor que 0. Sólo si pasa esta criba, comprobaremos si los puntos calculados de la primera curva de nivel se encuentran dentro de la segunda, introduciéndolos en la ecuación de esta última curva y observando si la distancia que generaría estos dos lambdas es menor que la distancia máxima permitida que determina la segunda curva de nivel. En caso de que se cumpla este test de inclusión almacenaremos las estaciones candidatas en las columnas 5 y 6 del arreglo (mat). Este proceso se repetirá variando el punto de partida (lambda1) hasta que se encuentre un par de puntos válidos. Si no se encuentran, significará que una curva no está dentro de la otra y por tanto habrá que hacer la comprobación de forma recíproca.

Si tenemos un resultado negativo en las dos comprobaciones anteriores, nos hallaremos en la segunda opción. En este caso, operando de forma análoga a la anterior, vamos a calcular un punto perteneciente a cada una de las curvas de nivel de los pares de punto de demanda y los añadiremos como posibles soluciones a probar.

En este momento ya tenemos almacenadas en el arreglo (mat) todas las candidatas a soluciones en las columnas 5 y 6 de los diferentes planos. Tan solo nos queda evaluar nuestra función objetivo, es decir, evaluar cuál de los posibles pares de estaciones candidatas nos aportaría un viaje óptimo (una mayor demanda). Lo primero que tenemos que hacer es saber el número de iteraciones que existirán teniendo en cuenta cada par de pares posibles. Para esto usaremos una variable auxiliar que iremos incrementando y que se encontrará dentro de 4 bucles **for**. Con los cuatro bucles podremos conseguir las distintas combinaciones entre par de pares de puntos de demanda. De esta forma ya tenemos el número de iteraciones que realizará esta parte del código.

De nuevo vamos a crear dos arreglos (sol1 y sol2), estando sol1 asociado a la "opción a" y sol2 a la "opción b", que tendrán tantos planos como posibles soluciones de cada tipo. Cada plano tendrá 1 fila y 3 columnas, en las cuales guardaremos: lambda 1, lambda 2 y el valor que adquiere nuestra función objetivo para las estaciones que definen esas lambdas. Ahora pasamos a realizar un barrido por todos los planos de ambas opciones, cada uno con una solución candidata asociada. En cada iteración vamos a comprobar si esa solución nos permitiría encontrarnos dentro de la curva de nivel asociada a cada uno de los pares de puntos de demanda. En caso afirmativo eso implicaría que esa solución nos permitiría validar el traslado entre ese par de puntos de demanda y añadiríamos su

demanda asociada. Tras calcular la función objetivo de cada punto candidato la solución final será aquella que nos permita tener un valor total mayor dentro de las dos opciones.

Vamos a dibujar todos los puntos candidatos remarcando la solución final para tener una visualización global del problema. Para finalizar el código devolveremos como salidas de la función la función obtenida.

6 EXPLICACIÓN DEL CÓDIGO MATLAB PARA EL CASO DE UNA RED FORMADA POR MÁS DE DOS ARISTAS

El código que se va a explicar a continuación se utilizará para resolver el problema de la localización óptima de dos estaciones en redes de transporte ferroviario, en el caso de que las estaciones estén situadas en dos segmentos distintos en cualquiera de los segmentos que forman la red. Ambas estaciones podrán estar en el mismo segmento o cada una en un segmento diferente.

Para comenzar el código necesitaremos meter como entrada algunos elementos que pasamos a describir:

- A: matriz formada por tantas filas como puntos de demanda tratemos y por 2 columnas, que serán las coordenadas "x" e "y" de cada uno de los puntos de demanda que estén en el plano.
- p1, p2, p3,…, pn: coordenadas bidimensionales de los puntos que son nexo de unión entre segmentos o que inician o cierran la cadena.
- mat_puntos: matriz en la que guardamos las coordenadas de cada par de nexos que van consecutivos y forman un rama
- mat_puntos_repr: matriz en la que almacenamos los segmentos de forma que nos facilite su representación gráfica.
- D: matriz de distancias máximas permitidas entre puntos de demanda. Esta matriz será de dimensiones [número de puntos de demanda x número de puntos de demanda]. La matriz D solo será distinta de 0 en los elementos que queden por encima de la diagonal principal ya que por la naturaleza del problema el par "ij" está cubierto si y sólo si lo está el "ji". De esta forma, el elemento "D (i, j)" nos indicará la distancia que podría existir entre el punto de demanda "i" y el "j" que será la misma que habrá entre en punto de demanda "j" y el "i".
- *alpha*: factor de velocidad comprendido entre 0 y 1.
- T: matriz de demanda entre puntos de demanda. Esta matriz será de dimensiones [número de puntos de demanda x número de puntos de demanda]. Por el mismo motivo de simetría que en D, podemos agrupar las demandas de i a j con las de j a i. Debido a esto, la matriz T solo será distinta de 0 en los elementos que queden por encima de la diagonal principal. De esta forma el elemento "T (i, j)" nos indicará la demanda que va a existir para el trayecto entre el punto de demanda "i" y el "j" más la que habrá entre en los puntos de demanda "j" y el "i".

Lo primero que vamos a hacer es analizar mediante el comando **length** el número de segmentos que van a constituir el árbol. Guardaremos en una matriz (mat_direcc) los vectores que indican la dirección de cada segmento y en otra matriz (mat_long) la longitud de cada rama del árbol.

A continuación vamos representar en negro la situación de la que partimos inicialmente, es decir, los diferentes segmentos y los nexos. Añadiremos también en color verde la localización de todos los puntos de demanda.

En este caso en el que la red es tipo árbol vamos a necesitar un algoritmo el cual haga llamadas a otras dos funciones. Dentro del código principal, analizaremos la posibilidad de que las dos estaciones estén en el mismo segmento o que estén en segmentos diferentes. Nuestro objetivo será analizar ambas posibilidades para determinar cuál es la que aporta una solución óptima.

Empezaremos trabajando en el caso de que estén contenidas en la misma rama. Vamos a almacenar las soluciones propuestas en la matriz "candidatos_a" la cual estará formada por estas columnas: estación1, estación2, demanda cubierta, punto inicial del segmento y punto final del segmento (estos dos son útiles a la hora de representar). Realizaremos un bucle **for** tantas veces como número de ramas existan. Dentro de este almacenaremos en la variable L, la longitud del segmento en tratamiento. A continuación haremos una llamada a la función denominada: SEGMENTO_A_COMPLETO, dándole como entradas: matriz de puntos de demanda, longitud de la arista a tratar, coordenadas iniciales y finales de la arista, matriz de distancias, coeficiente de velocidad y matriz de demandas. Esta función se comentará más adelante pero adelantamos que devolverá como salida la solución óptima para esa arista y cuanta demanda cubriría.

A continuación realizaremos un bucle para analizar cuál de las aristas, previamente tratadas individualmente, cubre una mayor demanda. Esta será la que almacenemos como candidata de la opción a, que luego compararemos con la que se generará si se considera el grafo completo.

Ahora tendremos que analizar el caso en el que tratemos el grafo tipo árbol completo estando cada estación en una arista diferente, sin tener que ser estas contiguas. Vamos a hacer una única llamada a la función BORDES_A_COMPLETO, aportándole como elementos de entrada: matriz de puntos de demanda, número total de aristas que forman el grafo, matriz de coordenadas de las uniones de los segmentos, matriz de vectores directores de todas las aristas, matriz con las longitudes de todos los segmentos, matriz de distancias, coeficiente de velocidad, matriz de demandas y matriz de pesos necesaria en de algoritmo de dijkstra que después será utilizado. En este caso, dentro de la única llamada se analizarán todas las variantes posibles y se devuelve directamente la opción que cubriría más demanda en el caso de que las estaciones estén en aristas diferentes. BORDES_A_COMPLETO nos devolverá en una matriz: lambda1, lambda2, máxima cobertura y coordenadas de las dos ramas donde hay estación.

Finalmente se analizará cuál de las dos opciones, ambas estaciones en la misma rama o una estación en cada rama, aportará una solución óptima siendo esta la solución final que representaremos.

Cabe mencionar que las dos funciones a las que se hace llamada son muy parecidas a las que se han usado y explicado anteriormente para el análisis de una sola arista y a la que usamos para el caso de dos estaciones en aristas contiguas.

En el caso de la función para una arista (SEGMENTO_A_COMPLETO) la diferencia principal que existirá será que necesitaremos trasladar, mediante rotación y traslación, las coordenadas de los puntos de demanda. Esto se debe a que esta función es genérica y trabaja siempre con el segmento con una inclinación de 45 grados respecto a la horizontal empezando en el punto (0,0). Debido a que en el árbol general cada arista tendrá una orientación y localización distinta

deberemos trasladar tanto la arista como los puntos de demanda acordemente. Una vez tengamos el escenario genérico, el código es igual que el utilizado para la arista única

Por otra parte, en el caso de la segunda función utilizada, BORDES_A_COMPLETA, la única diferencia que existe con respecto al código usado para el caso de dos aristas contiguas es que existe un tramo intermedio entre las aristas que contienen estación que también es parte del trayecto. Esta longitud también va multiplicada por el factor alpha ya que se recorre a alta velocidad y la calcularemos buscando el camino más corto entre estaciones usando el algoritmo de dijkstra.

7 ANÁLISIS DEL CASO EN EL QUE SOLO LOCALIZAMOS UNA ESTACIÓN

A lo largo de este trabajo siempre hemos planteado la búsqueda de dos estaciones y cabría la opción de preguntarse por qué no nos centramos en buscar solo una.

Esto se puede justificar debido a que el caso genérico que hemos empleado para localizar dos estaciones en un segmento ya engloba la búsqueda en el caso que sólo queramos una estación. Se puede decir que el caso de una estación es un caso particular del problema de averiguar la ubicación de dos estaciones. Sólo bastaría fijar la primera estación a uno de los nodos que delimita el segmento.

Siendo N1, N2 los nodos que delimitan la arista, Ai, Aj los puntos de demanda y E1, E2 las estaciones, gráficamente tendríamos:

- Problema en el que buscamos dos estaciones:

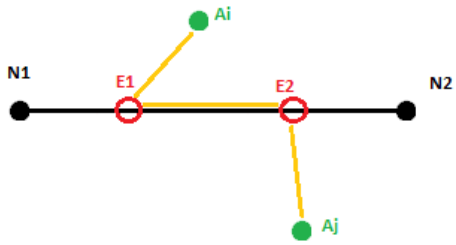

Figura 10

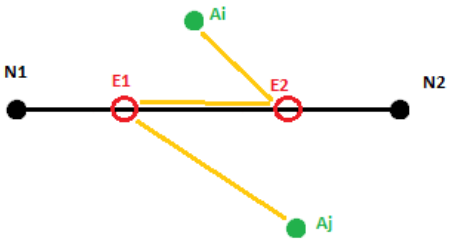

Figura 11

El viaje que haríamos sería el camino dibujado en amarillo y se observa como en ambos ejemplos entramos y salimos por las dos estaciones que hemos localizado, siendo estas diferentes a los nodos.

- Problema en el que buscamos una estación y la otra está fijada. Algunos casos:

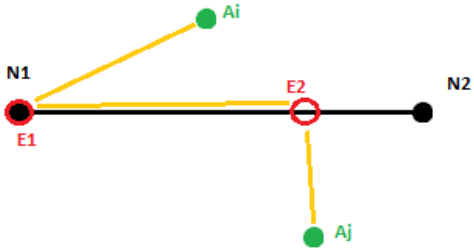

Figura 12

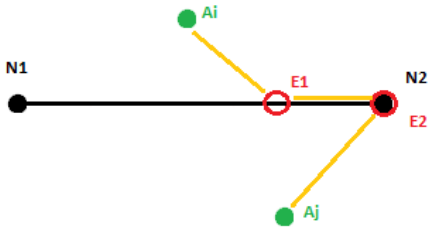

Figura 13

El viaje que haríamos sería el camino dibujado en amarillo y se observa como en ambos ejemplos entramos por un nodo del segmento (estación ya fijada) y salimos por la única estación que hemos calculado.

Matemáticamente se podrían analizar los 4 ejemplos anteriores simplemente viendo las curvas de nivel que se generarían. Para las figuras 10 y 11, nos fijaríamos en las intersecciones de las curvas de nivel que se generarían dentro del área formado por dos ejes que ambos van desde 0 a L (AZUL Y AMARILLO). Sin embargo, si quisiéramos analizar la figura 12 tendríamos que observar los cortes que existiesen entre las curvas de nivel y el primer eje vertical (ROJO). Por último, si estuviésemos interesados en analizar la figura 13 deberíamos mirar las intersecciones de las curvas con el eje vertical en x=L (VERDE).

Figura 14

8 EXPERIENCIA COMPUTACIONAL

Tras acabar los diseños y las comprobaciones de los códigos, hemos pasado a realizar varias experiencias computacionales para almacenar los tiempos necesarios de ejecución en cada caso. Se han hecho varios análisis para poder plasmar en estas tablas las medias de todos los tiempos obtenidos para cada apartado. Haremos un barrido por los diferentes valores que tomarán las variables significativas de los códigos y anotaremos de forma orientativa los tiempos. Así podremos adquirir información que podría resultar útil en posteriores usos del código.

A continuación se presentan los valores obtenidos para el algoritmo 1, el caso de una única arista:

Longitud de la arista (→) Nº Puntos de demanda (↓)	5	10
5	00h 16m 27s	00h 22m 51s
10	06h 53m 52s	06h 59m 54s

Tabla 1

Ahora plasmamos los resultados para el algoritmo 2, el caso de dos aristas contiguas:

Suma de longitudes de los dos segmentos (→) Nº Puntos de demanda (↓)	5	10
5	00h 45m 30s	00h 48m 32s
10	14h 28m 07s	15h 07m 48s

Tabla 2

Cabe destacar que en todas las simulaciones realizadas, aproximadamente el 85% del tiempo empleado se invierte en la resolución de las ecuaciones con el comando **solve.**

9 RESULTADOS

A modo de ilustración vamos a mostrar las gráficas de salidas que proporcinan los códigos MATLAB para cada caso. En ellas se observan todos los puntos de demanda que estemos tratando de color verde y el tipo de red en color negro.

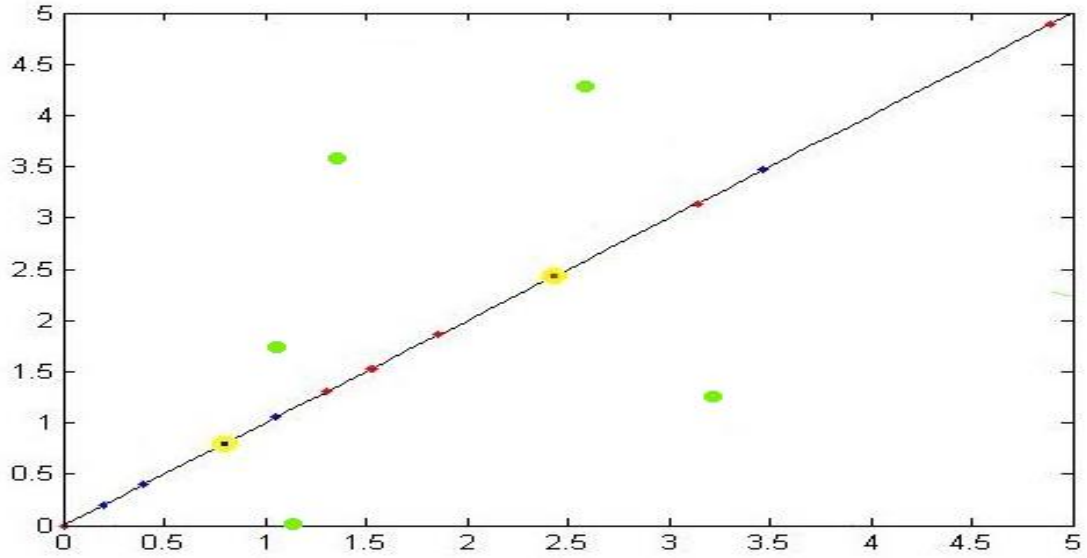

Figura 15

En esta imagen se muestra un segmento en el cual los puntos azules y rojos representan puntos candidatos a ser estaciones, los azules de entrada a la red y los rojos de salida. Remarcamos en amarillo la solución óptima.

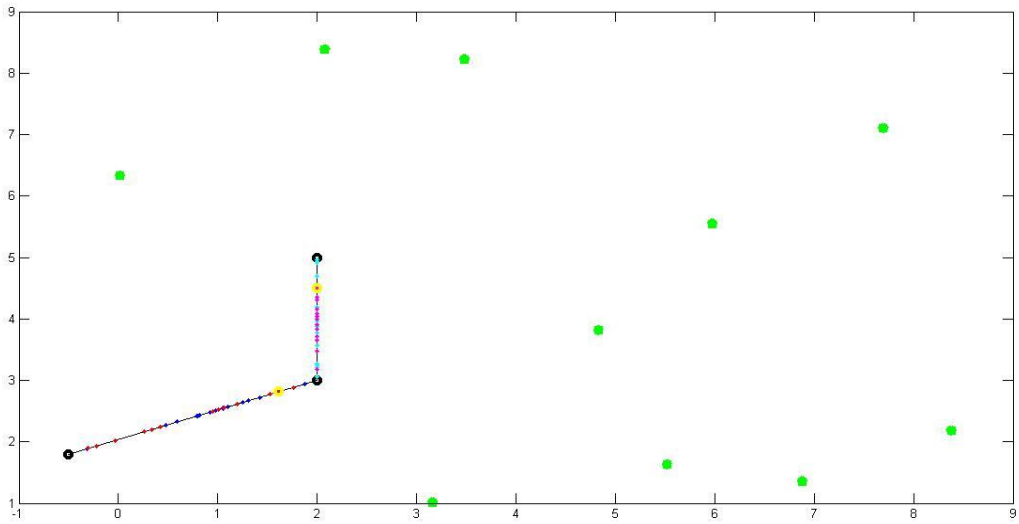

Figura 16

En esta segunda imagen se muestra el caso en el que la red esté formada por dos segmentos contiguos. En este caso tenemos dos opciones de entrada y salida de la red. La primera es viajando desde el primer punto de demanda pasando por una estación en el primer segmento, cuyas candidatas coloreamos de rojo, saliendo de la red por una estación contenida en el segundo segmento, cuyas candidatas coloreamos de fucsia y finalizando el viaje en el segundo punto de demanda. La otra opción es viajando desde el segundo punto de demanda pasando por una estación en el primer segmento, cuyas candidatas coloreamos de azul, saliendo de la red por una estación contenida en el segundo segmento, cuyas candidatas coloreamos de celeste y finalizando el trayecto en el segundo punto de demanda. Remarcamos en amarillo la solución final óptima.

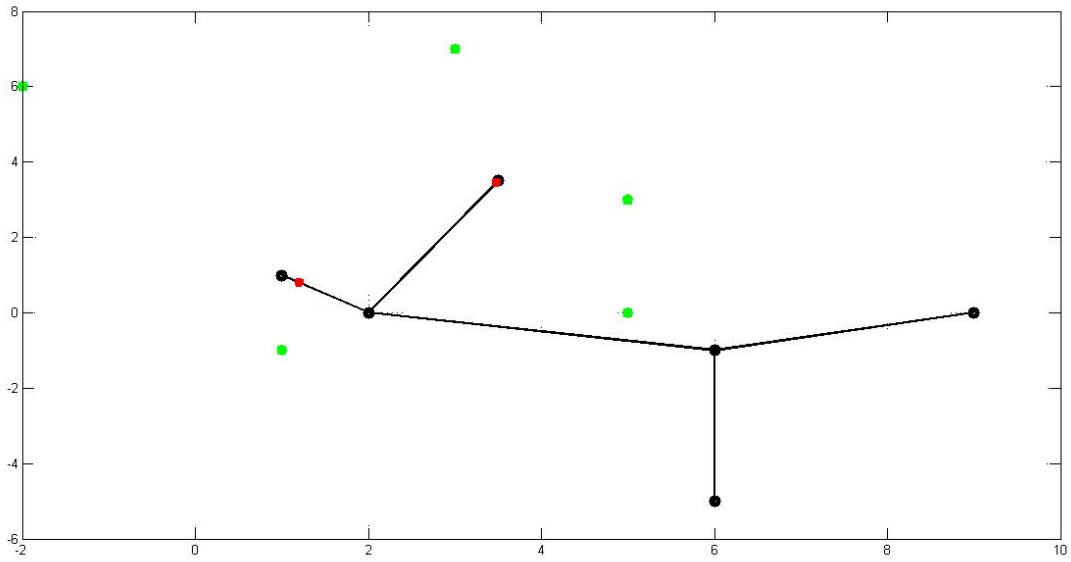

Figura 17

En este caso se ha representado el caso de una red tipo árbol con varias ramas, y por simplicidad solo mostramos la solución final óptima en rojo, cada una en una arista diferente.

10 REFERENCIAS BIBLIOGRÁFICAS

[1] Mark-Christoph Körner, Juan A.Mesa, Federico Perea, Anita Schöbel, Daniel Scholz, «A maximum trip covering location problem with an alternative mode of transportation on tree networks and segment» *Top (2014) 22: 227-253*

[2] Gilbert Laporte, Juan A.Mesa, Francisco A. Ortega, Ignacio Sevillano, «Maximizing trip coverage in the location of a single rapid transit alignment» *Annals of Operations Research (2005) 136: 49-63*

[3] Juan A.Mesa, Federico Perea, Gilbert Laporte, «Adding a new station and a road link to a road-rail network in the presence of a model competiton» *Transportation Research Part B (2014) 68: 1-16*

[4] Gilbert Laporte, Stefan Nickel, Francisco Saldanha da Gama, « Location Science, *Chapter 23*: *241-255*, The design of a rapid transit network», *(2015) Springer*

[5] Vukan R. Vuchic, «Urban transit: operations, planning and economics, *Chapter 5*: *268-291,* Planning of rail transit station locations», *(2005) Hoboken*

[6] Blas Pelegrín, Lázaro Cánovas, Pascual Fernández, «Algoritmos en grafos y redes, *Capítulo 1*: Definiciones y conceptos básicos en grafos, *Capítulo 3:* Árboles», *(1992) PPV*

[7] Vukan R. Vuchic, Gordon F. Newell, «Rapid transit interstation spacings for minimun travel time», *(1968) Trans. Science 2*

[8] J. de Cea, J. de D. Ortúzar, L.G. WIllumsen, «Evaluating marginal improvements to a transport network: An application to the Santiago underground», *(1986) Martinus Nijhoff Publishers*

[9] Gilbert Laporte, Juan A. Mesa, Francisco A. Ortega, «Locating stations on rapid transit lines», *(2002) Computers and Operations Research 29, Pergamon*

[10] Evangelos Kranakis, Paolo Penna, Konrad Schlude, David Scot Taylor, Peter Widmayer, «Improving customer proximityn to railway stations», *(2003) Springer-Verlag Berlin Heidelberg*

[11] Nikolaus Ruf, Anita Schöbel, «Set covering with almost consecutive ones property», *(2004) Discrete Optimization 1, 215-228, Elsevier*

[12] Anita Schöbel, «Locating stops along bus or railway lines», *(2005) Annals of Operations Research,136,211-227 Springer Science + Business media*

[13] Hugo M. Repolho, Antonio P. Antunes, Richard L. Church, «Optimal location of railway stations: the lisbon-Porto high-speed rail line», *(2013) Transportation Science 47, Nº3 August . Informs*

[14] Horst W. Hamacher, Simone Horn, Anita Schöbel, «Stop location design in public transportation networks: covering and accessibility objetives», (2008) *Top. Springer*

[15] Emilio Carrizosa, Jonas Harbering, Anita Schöbel, «The stop location problem with realistic traveling time», (2009) *ATMOS´13*

[16] Teorema de Bezout: *https://es.wikipedia.org/wiki/Teorema_de_B%C3%A9zout*

ANEXOS

ANEXO A: CÓDIGO MATLAB PARA UNA RED CON UNA ÚNICA ARISTA

```
clear all
close all
clc
format short

% A: matriz de puntos de demanda,
% D: matriz de distancias máximas válidas
% alpha: factor de velocidad
% T: matriz de demandas
% L: longitud del segmento

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%DATOS(Ejemplo para una A, L, D, T y alpha concreta.)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

A=[1  1;
    1 -0.5;
    4 -0.5;
    2 -2
    ];
L=5;
D= [0   4.24  3.04  3.95     ;
    0   0     3     3.4      ;
    0   0     0     2.02     ;
    0   0     0     0
    ];

T= [0   32    27    19      ;
    0   0     49    48      ;
    0   0     0     67      ;
    0   0     0     0
    ];
alpha=0.5;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FIN DATOS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[numpueblos,numcoord]=size(A);
xxa=0:0.1:L; yya=0:0.1:L;
[xm4,ym4]=meshgrid(xxa,yya);

  % variable que usaremos para saber la dimensión de f y d, de forma
que así sean igual que "x" e "y" y se pueda operar
var=[0:0.1:L];
var=length(var);
%%%%
```

```
%%
%2.CALCULO DE LAS fij:
f=zeros(var,var,var) ;

k=0;
for xa=1:numpueblos
    for xb=1:numpueblos
        k=k+1;


        if(xa<xb)
            if ( A(xa,1) < A(xb,1) )  %si la coordenada "x" del pueblo
i es menor que la  coordenada "x" del pueblo j-> entra por i sale por
j
                f(:,:,k)= sqrt((A(xa,1)-xm4).^2 +(A(xa,2)).^2 )  +
alpha* (ym4-xm4)  +  sqrt((A(xb,1)-ym4).^2 +(A(xb,2)).^2 )  - D(xa,xb)
;
            else                    %si la coordenada "x" del pueblo j
es menor o igual  que la  coordenada "x" del pueblo i-> entra por j
sale por i
                f(:,:,k)= sqrt((A(xb,1)-xm4).^2 +(A(xb,2)).^2 )  +
alpha* (ym4-xm4)  +  sqrt((A(xa,1)-ym4).^2 +(A(xa,2)).^2 )  - D(xa,xb)
;
            end
        end


    end
end


%DIBUJO DE LOS CONTORNOS

for k=1:var
    g=f(:,:,k);
    contour(xm4,ym4,g,[0,0]);
    hold on
end
hold on
plot(xxa,yya,'k')


%%
% 3.INTERSECCIONES

%calculamos el numero de intersecciones para poder dimensionar la
celda
%donde las almacenaremos
aux=0;
for k=1:numpueblos-2
    for kk=1:numpueblos
        if (k<kk)
            for i=1:numpueblos-1
                for ii=1:numpueblos
                    if( (i<ii && i>=k && ii>kk) || (i<ii && i>=k+1 )
)
                        aux=aux+1;
                    end
```

```
                end
            end
        end
    end
end
NUM_ITER=aux;


aux2=1;
aux4=1;
syms x y ya yb yc
xx=[];
yy=[];
xxx=[];
yyy=[];
mat=zeros(12,6,NUM_ITER);% los elementos son: k, kk, i, ii, x[], y[].

for k=1:numpueblos-2
    for kk=1:numpueblos
        if (k<kk)

            for i=1:numpueblos-1
                for ii=1:numpueblos
                    if( (i<ii && i>=k && ii>kk) || (i<ii && i>=k+1 )
)


                        [xx,yy]=solve(sqrt((A(k,1)-x)^2 +(A(k,2))^2 )
+  alpha*(y-x) +  sqrt((A(kk,1)-y)^2 +(A(kk,2))^2 ) == D(k,kk),...
                            sqrt((A(i,1)-x)^2 +(A(i,2))^2 )  +
alpha*(y-x) +  sqrt((A(ii,1)-y)^2 +(A(ii,2))^2 ) ==
D(i,ii),'Real',true);

                        dim=length(xx);
                        aux3=0;
                        for aux2=1:dim
                            imagx=imag(xx(aux2));
                            imagy=imag(yy(aux2));
                            if (imagx == 0.00 || imagy == 0.00)%solo
lo consideramos solucion si la x e y no son imaginarios
                                aux3=aux3+1;
                                xxx(aux3)=xx(aux2);
                                yyy(aux3)=yy(aux2);
                            end
                        end

                        mat(1,1,aux4)=k;
                        mat(1,2,aux4)=kk;
                        mat(1,3,aux4)=i;
                        mat(1,4,aux4)=ii;
                        for p=1:dim
                            mat(p,5,aux4)=xxx(p);
                            mat(p,6,aux4)=yyy(p);

                        end

                        aux4=aux4+1;
                    end
```

```
                end
            end

        end
    end
end



%%
% 4. SI NO INTERSECTAN:...es decir el elemento x e y es cero
for p=1:NUM_ITER
    if (mat(1,5,p)==0 &&  mat(1,6,p)==0);
        ek=mat(1,1,p);
        ekk=mat(1,2,p);
        ei=mat(1,3,p);
        eii=mat(1,4,p);
        %la interseccion que estudiaremos sera la de f(ek,ekk) vs
f(ei,eii)

        flag=0;
        flag1=0;
        flag2=0;
        flag3=0;
        flag4=0;
        flag5=0;

        %-----------------------------------4a.1
        if flag4==0
            for xaux=0:0.1:L
                ecu=sqrt((A(ek,1)-xaux).^2 +(A(ek,2)).^2 )  +  alpha*
(y-xaux)  +  sqrt((A(ekk,1)-y).^2 +(A(ekk,2)).^2 )  - D(ek,ekk);
                yg=solve(ecu,y,'Real',true);
                for id=1:length(yg)
                    if flag==0
                        yaux=yg(id,1);
                        yaux=double(yaux);
                        if imag(yaux)==0 && yaux<=L && xaux<yaux
                            %el punto (xaux,yaux) pertenece a la curva
de nivel asociada a f(ek,ekk)

                            if sqrt((A(ei,1)-xaux).^2+(A(ei,2)).^2 ) +
alpha* (yaux-xaux) + sqrt((A(eii,1)-yaux).^2 +(A(eii,2)).^2 ) <=
D(ei,eii)
                                % el punto xaux,yaux  de f (ek,ekk)
esta dentro de la curva de f(ei,eii)...meter xaux,yaux en la nube de
puntos de la curva
                                mat(1,5,p)=xaux;
                                mat(1,6,p)=yaux;
                                flag=1;
                                flag4=1;
                            end
                        end
                    end
                end
            end
        end
        %-----------------------------------4a.2
```

```
        if flag4==0
            for xaux1=0:0.1:L
                ecu1= sqrt((A(ei,1)-xaux1).^2 +(A(ei,2)).^2 )  +
alpha* (ya-xaux1)  +  sqrt((A(eii,1)-ya).^2 +(A(eii,2)).^2 )  -
D(ei,eii);
                yg1=solve(ecu1,ya,'Real',true);
                for id=1:length(yg1)
                    if flag1==0
                        yaux1=yg1(id,1); %por si tiene mas de una
solucion ,coger la primera

                        if imag(yaux1)==0 && yaux1<=L && xaux1<yaux1
                            %el punto (xaux,yaux) pertenece a la curva
de nivel asociada a f(ei,eii)
                            yaux1=double(yaux1);
                            if  sqrt((A(ek,1)-xaux1).^2 +(A(ek,2)).^2
)+alpha* (yaux1-xaux1)+sqrt((A(ekk,1)-yaux1).^2 +(A(ekk,2)).^2 )<=
D(ek,ekk)
                                % el punto xaux,yaux  de f (ei,eii)
esta dentro de la curva de f(ek,ekk)...meter xaux,yaux en la nube de
puntos de la curva
                                mat(1,5,p)=xaux1;
                                mat(1,6,p)=yaux1;
                                flag1=1;
                                flag4=1;
                            end
                        end
                    end
                end
            end
        end
        %-----------------------------------4b
        %si no es ninguno de los casos anteriores... intersectan o
esta contenidos alguno dentro del otro se añade un punto de cada
curva:


        if(flag4==0)
            for xaux2=0:0.1:L
                ecu2= sqrt((A(ek,1)-xaux2).^2 +(A(ek,2)).^2 )  +
alpha* (yb-xaux2)  +  sqrt((A(ekk,1)-yb).^2 +(A(ekk,2)).^2 )  -
D(ek,ekk);
                yg2=solve(ecu2,yb,'Real',true);
                for id=1:length(yg2)
                    if flag2==0
                        yaux2=yg2(id,1);
                        if imag(yaux2)==0 && yaux2<=L && xaux2<yaux2
                            yaux2=double(yaux2);
                            mat(1,5,p)=xaux2;
                            mat(1,6,p)=yaux2;
                            flag2=1;
                            flag5=1;% condicion para entrar en el otro
bucle
                        end
                    end
                end
            end
        end
```

```
        if flag5==1
            for xaux3=0:0.1:L
                ecu3= sqrt((A(ei,1)-xaux3).^2 +(A(ei,2)).^2 )  +
alpha* (yc-xaux3)  +  sqrt((A(eii,1)-yc).^2 +(A(eii,2)).^2 )  -
D(ei,eii);
                yg3=solve(ecu3,yc,'Real',true);
                for id=1:length(yg3)
                    if flag3==0
                        yaux3=yg3(id,1);%por si tiene mas de una
solucion ,coger la primera
                        if imag(yaux3)==0 && yaux3<=L && xaux3<yaux3

                            yaux3=double(yaux3);
                            %el punto (xaux1,yaux1) pertenece a la
curva de nivel asociada a f(ei,eii) y hay que añadirlo a la nube de
puntos
                            mat(2,5,p)=xaux3;
                            mat(2,6,p)=yaux3;
                            flag3=1;
                            flag4=1;
                        end
                    end

                end
            end
        end
    end


    %%
    % 5.EVALUAR LA FUNCION OBJETIVO

    num_sol=0;
    %bucle para calcular el numero de posibles soluciones (x,y)
    for aux5=1:NUM_ITER
        for jj=1:12
            if ((mat(jj,5,aux5)~=0) && ( mat(jj,6,aux5)~=0 ))
                num_sol=num_sol+1;
            end
        end
    end


    sol=zeros(1,3,num_sol+1); % siendo los elementos : x , y, suma de
ti
    for aux6=1:NUM_ITER
        for jj=1:12
            if mat(jj,5,aux6)~=0 &&  mat(jj,6,aux6)~=0 &&
imag(mat(jj,5,aux6))==0 &&  imag(mat(jj,6,aux6))==0
                sol(1,1,aux6)=mat(jj,5,aux6);
                sol(1,2,aux6)=mat(jj,6,aux6);
            end
        end
    end


    for pla=1:num_sol
        for i=1:numpueblos
```

```
            for j=1:numpueblos
                if(i<j)
                    if ( A(i,1) < A(j,1) )  %si la coordenada "x" del
pueblo i es menor que la  coordenada "x" del pueblo j-> entra por i
sale por j
                        if ( sqrt( (A(i,1)-sol(1,1,pla)).^2
+(A(i,2)).^2 )  +  alpha* (sol(1,2,pla)- sol(1,1,pla))  +...
                                sqrt( (A(j,1)-sol(1,2,pla)).^2
+(A(j,2)).^2 ) <= D(i,j) )

                            sol(1,3,pla)=sol(1,3,pla)+T(i,j);

                        end
                    else                    %si la coordenada "x" del
pueblo j es menor o igual  que la  coordenada "x" del pueblo i-> entra
por j sale por i
                        if ( sqrt( (A(j,1)-sol(1,1,pla)).^2
+(A(j,2)).^2 )  +  alpha* (sol(1,2,pla)-sol(1,1,pla))  + ...
                                sqrt( (A(i,1)-sol(1,2,pla)).^2
+(A(i,2)).^2 )  <= D(i,j) )

                            sol(1,3,pla)=sol(1,3,pla)+T(i,j);
                        end
                    end
                end
            end
        end
    end

end
for i=1:num_sol

    xdi=sol(1,1,i);
    ydi=sol(1,2,i);
    plot(xdi,xdi,'b.','LineWidth',3)
    plot(ydi,ydi,'r.','LineWidth',3)

end


maximo=0;
indice=0;
for pla=1:num_sol
    if sol(1,3,pla)> maximo
        maximo=sol(1,3,pla);
        indice=pla;
    end
end

display('LAS SOLUCIONES SON :')
X_SOLUCION= sol(1,1,indice)
Y_SOLUCION= sol(1,2,indice)


plot(X_SOLUCION,Y_SOLUCION,'+k')
plot(X_SOLUCION,X_SOLUCION,'oy','LineWidth',3)
plot(Y_SOLUCION,Y_SOLUCION,'oy','LineWidth',3)
```

ANEXO B: CÓDIGO MATLAB PARA UNA RED CON DOS ARISTAS CONTIGUAS

```
clear all
close all
clc
% A: matriz de pueblos ,
% p1: punto inicial del primer segmento (xp1,yp1)
% d1: direccion del primer eje ( (xp2-xp1),(yp2-yp1) )
% p2: punto inicial del segundo sgmento (xp2,yp2)
% d2: direccion del segundo eje ( (xp3-xp2),(yp3-yp2) )
% D: matriz de distancias máximas válidas
% alpha: factor de velocidad
% T: matriz de demandas


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%DATOS (Ejemplo para una A, p1, d1, p2, d2, D, T, alpha concretos.)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
p1=[-1,7];   d1=[3 -4];
p2=[2,3];   d2=[4 -2];

alpha=0.2;

A=[1  1;
    1 -0.5;
    4 -0.5;
    2 -2
    ];

D= [0   4.24  3.04  3.95     ;
    0   0     3     3.4      ;
    0   0     0     2.02     ;
    0   0     0     0
    ];

T= [0   32    27    19      ;
    0   0     49    48      ;
    0   0     0     67      ;
    0   0     0     0
    ];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FIN DATOS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[numpueblos,numcoord]=size(A);
p3=p2+d2;
```

```
L1=sqrt( (p2(1)-p1(1))^2 + (p2(2)-p1(2))^2 );L1=L1*1000;
Lo1=floor(L1); L1=Lo1/1000;  %Cogemos solo 4 cifras significativas y
asi evitamos que el programa se atasque en los bucles
L2=sqrt( (p3(1)-p2(1))^2 + (p3(2)-p2(2))^2 );L2=L2*1000;
Lo2=floor(L2); L2=Lo2/1000;  %Cogemos solo 4 cifras significativas y
asi evitamos que el programa se atasque en los bucles
L=L1+L2;

xxa=0:0.1:L; yya=0:0.1:L;
[la1,la2]=meshgrid(xxa,yya);

var=length(la1);  % variable que usaremos para saber la dimension de f
y d, de forma que asi sean igual que "x" e "y" y se pueda operar

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% 2.CALCULO DE LAS fij:
f12=zeros(var,var,var) ;
f21=zeros(var,var,var) ;

kit=0;

for xa=1:numpueblos
    for xb=1:numpueblos

        % la primera estación está en:
(p1(1)+la1*d1(1),p1(2)+la1*d1(2)) y la segunda en:
(p2(1)+la2*d2(1),p2(2)+la2*d2(2))
        if(xa<xb)
            kit=kit+1;
            % Ai -> e1 -> p2 -> e2 -> Aj
            f12(:,:,kit)= sqrt( ((p1(1)+ d1(1)*la1)-A(xa,1)).^2  +
((p1(2)+ d1(2)*la1)-A(xa,2)).^2  ) ...
                + alpha*(1-la1) * L1 + alpha*(la2) * L2 ...
                + sqrt( (A(xb,1)-(p2(1)+ la2*d2(1))) .^2  +  (A(xb,2)-
(p2(2)+la2*d2(2))).^2 )   -   D(xa,xb);

            % Ai -> e2 -> p2 -> e1 -> Aj
            f21(:,:,kit)= sqrt( ((p2(1)+ la2*d2(1))-A(xa,1)).^2  +
((p2(2)+ la2*d2(2))-A(xa,2)).^2  ) ...
                + alpha*(1-la1) * L1 + alpha*(la2) * L2 ...
                + sqrt( (A(xb,1)-(p1(1)+ la1*d1(1))) .^2  +  (A(xb,2)-
(p1(2)+la1*d1(2))).^2 )   -   D(xa,xb);
        end

    end
end

%DIBUJO DE LOS CONTORNOS
for k=1:kit
    g12=f12(:,:,k);
    contour(la1,la2,g12,[0,0],'g');
    hold on
    g21=f21(:,:,k);
    contour(la1,la2,g21,[0,0],'r')
end


%%
```

```
% 3.INTERSECCIONES

%calculamos el numero de intersecciones para poder dimensionar el
arreglo donde las almacenaremos
aux=0;
for k=1:numpueblos-2
    for kk=1:numpueblos
        if (k<kk)
            for i=1:numpueblos-1
                for ii=1:numpueblos
                    if( (i<ii && i>=k && ii>kk) || (i<ii && i>=k+1 )
)
                        aux=aux+1;
                    end
                end
            end
        end
    end
end
NUM_ITER=aux;

mat = zeros(24,6,NUM_ITER);% los elementos son: k, kk, i, ii,
la1_iter1[] y la1_iter2[] , de la fila 1-12 para ir de 1 a 2 ,de la
fila 13-24 de 2 a 1

e1x=[];     e1y=[];      e2x=[];     e2y=[];
xxx1=[];    yyy1=[];    xxx2=[];    yyy2=[];

aux4=0;
syms la1 la2

for k=1:numpueblos-2
    for kk=1:numpueblos
        if (k<kk)
            for i=1:numpueblos-1
                for ii=1:numpueblos
                    if( (i<ii && i>=k && ii>kk) || (i<ii && i>=k+1 )
)

                        aux4=aux4+1;

                        % Ai -> e1 -> p2 -> e2 -> Aj
                        [e1x,e1y]=solve(sqrt( ((p1(1)+ d1(1)*la1)-
A(k,1)).^2  +  ((p1(2)+ d1(2)*la1)-A(k,2)).^2  ) ...
                            + alpha*(1-la1) * L1 + alpha*(la2) * L2
...
                            + sqrt( (A(kk,1)-(p2(1)+ la2*d2(1))) .^2
+  (A(kk,2)-(p2(2)+la2*d2(2))).^2 )  ==  D(k,kk), ...
                            ...
                            sqrt( ((p1(1)+ d1(1)*la1)-A(i,1)).^2  +
((p1(2)+ d1(2)*la1)-A(i,2)).^2  ) ...
                            + alpha*(1-la1) * L1 + alpha*(la2) * L2
...
                            + sqrt( (A(ii,1)-(p2(1)+ la2*d2(1))) .^2
+  (A(ii,2)-(p2(2)+la2*d2(2))).^2 )  == D(i,ii),'Real',true);


                        % Ai -> e2 -> p2 -> e1 -> Aj
```

```
                        [e2x,e2y]=solve( sqrt( ((p2(1)+ la2*d2(1))-
A(k,1)).^2  +  ((p2(2)+ la2*d2(2))-A(k,2)).^2  ) ...
                            + alpha*(la2) * L2 + alpha*(1-la1) * L1
...
                            + sqrt( (A(kk,1)-(p1(1)+ la1*d1(1))) .^2
+  (A(kk,2)-(p1(2)+la1*d1(2))).^2 )  ==  D(k,kk),...
                            ...
                            sqrt( ((p2(1)+ la2*d2(1))-A(i,1)).^2  +
((p2(2)+ la2*d2(2))-A(i,2)).^2  ) ...
                            + alpha*(la2) * L2 + alpha*(1-la1) * L1
...
                            + sqrt( (A(ii,1)-(p1(1)+ la1*d1(1))) .^2
+  (A(ii,2)-(p1(2)+la1*d1(2))).^2 )  ==  D(i,ii),'Real',true) ;

                        dim1=length(e1x);
                        dim2=length(e2x);

                        aux3=0;
                        for aux2=1:dim1
                            if (e1x(aux2)<=1  &&  e1x(aux2)>0  &&
e1y(aux2)<=1  &&  e1y(aux2)>0) % solo las soluciones que nos permitan
tener la estacion dentro del primer segmento
                                aux3=aux3+1;
                                xxx1(aux3)=e1x(aux2);
                                yyy1(aux3)=e1y(aux2);
                            end
                        end

                        aux6=0;
                        for aux5=1:dim2
                            if (e2x(aux5)<=1  &&  e2x(aux5)>0  &&
e2y(aux5)<=1  &&  e2y(aux5)>0) % solo las soluciones que nos permitan
tener la estacion dentro del segundo segmento
                                aux6=aux6+1;
                                xxx2(aux6)=e2x(aux5);
                                yyy2(aux6)=e2y(aux5);
                            end
                        end

                        mat(1,1,aux4)=k;
                        mat(1,2,aux4)=kk;
                        mat(1,3,aux4)=i;
                        mat(1,4,aux4)=ii;

                        for p=1:aux3
                            mat(p,5,aux4)=xxx1(p);
                            mat(p,6,aux4)=yyy1(p);
                        end

                        for p=1:aux6
                            mat(12+p,5,aux4)=xxx2(p);
                            mat(12+p,6,aux4)=yyy2(p);
                        end

                    end
                end
            end
        end
```

```
    end
end

%% 4. SI NO INTERSECTAN:...es decir los elementos la1_iter1[],
la2_iter1[] son cero la1[] y la2 de alguno de los sentidos del
trayecto:

for p=1:NUM_ITER % cada plano
    for pp=1:12:13   % analizar el primer elemento ,1 y 13, de cada
tipo de interseccion
        if ((mat(pp,5,p)==0  &&  mat(pp,6,p)==0))

            ek=mat(1,1,p);
            ekk=mat(1,2,p);
            ei=mat(1,3,p);
            eii=mat(1,4,p);

          %% FLAGS QUE NECESITAMOS:
            A1A=0;  A1B=0;  A2A=0;  A2B=0;  BA1=0;   BB1=0;   BA2=0;
BB2=0;   FIN1=0; FIN2=0;

            %-----------------------------------4a.1

            % CASO A1A
            if pp==1 %ANALIZAMOS LAS POSIBLES SOLUCIONES DE LA PRIMERA
ITERACION
                for la1aux=0:0.025:1
                    if FIN1==0
                        %el punto que genera (la1aux,la2aux) pertenece
a la curva de nivel asociada a f12(ek,ekk)
                        syms  la2auxx
                        ecu= sqrt( ((p1(1)+la1aux*d1(1))-A(ek,1)).^2 +
((p1(2)+la1aux*d1(2))-A(ek,2)).^2 ) ...
                            + alpha*(1-la1aux) * L1 + alpha*(la2auxx)
* L2 ...
                            + sqrt( (A(ekk,1)-
(p2(1)+la2auxx*d2(1))).^2 + (A(ekk,1)-(p2(2)+la2auxx*d2(2))).^2 )-
D(ek,ekk);
                        yg=solve(ecu,la2auxx,'Real',true);

                        for id=1:length(yg)
                            if A1A==0
                                la2aux=yg(id,1);
                                la2aux=double(la2aux);
                                if la2aux>0 && la2aux<=1
                                    % si el punto que genera
(la1aux,la2aux)  de f12(ek,ekk) esta dentro de la curva de
f21(ei,eii)...meter (la1aux,la2aux)  en la nube de puntos de la curva
                                    if (sqrt( ((p1(1)+la1aux*d1(1))-
A(ei,1)).^2 + ((p1(2)+la1aux*d1(2))-A(ei,2)).^2 ) ...
                                        + alpha*(1-la1aux) * L1 +
alpha*(la2aux) * L2 ...
                                        + sqrt( (A(eii,1)-
(p2(1)+la2aux*d2(1))).^2 + (A(eii,1)-(p2(2)+la2aux*d2(2))).^2 ) <=
D(ei,eii))

                                        mat(pp,5,p)=la1aux;
                                        mat(pp,6,p)=la2aux;
```

```
                                    A1A=1;
                                    FIN1=1;
                                end
                            end
                        end
                    end
                end
            end

            % CASO A1B
            syms  la2auxx
            if pp==13 %ANALIZAMOS LAS POSIBLES SOLUCIONES DE LA
SEGUNDA ITERACION
                for la1aux=0:0.025:1
                    if FIN2==0
                        %el punto que genera (la1aux,la2aux) pertenece
a la curva de nivel asociada a f21(ek,ekk)
                        ecu= ( sqrt( ((p2(1)+la2auxx*d2(1))-
A(ek,1)).^2 + ((p2(2)+la2auxx*d2(2))-A(ek,2)).^2 ) ...
                            + alpha*(1-la1aux) * L1 + alpha*(la2auxx)
* L2 ...
                            + sqrt( (A(ekk,1)-(p1(1)+la1aux*d1(1))).^2
+ (A(ekk,1)-(p1(2)+la1aux*d1(2))).^2 )- D(ek,ekk));
                        yg=solve(ecu,la2auxx,'Real',true);

                        for id=1:length(yg)
                            if A1B==0
                                la2aux=yg(id,1);
                                la2aux=double(la2aux);
                                if la2aux>0 && la2aux<=1
                                    % si el punto que genera
(la1aux,la2aux)  de f21(ek,ekk) esta dentro de la curva de
f12(ei,eii)...meter (la1aux,la2aux)  en la nube de puntos de la curva
                                    if ( ((p1(1)+la1aux*d1(1))-
A(ei,1)).^2 + ((p1(2)+la1aux*d1(2))-A(ei,2)).^2 ) ...
                                            + alpha*(1-la1aux) * L1 +
alpha*(la2aux) * L2 ...
                                            + sqrt( (A(eii,1)-
(p2(1)+la2aux*d2(1))).^2 + (A(eii,1)-(p2(2)+la2aux*d2(2))).^2 ) <=
D(ei,eii)

                                        mat(pp,5,p)=la1aux;
                                        mat(pp,6,p)=la2aux;
                                        A1B=1;
                                        FIN2=1;
                                    end
                                end
                            end
                        end
                    end
                end
            end


            %-----------------------------------4a.2

            %CASO A2A
```

```
            syms  la2auxx
            if pp==1 %ANALIZAMOS LAS POSIBLES SOLUCIONES DE LA PRIMERA
ITERACION
                for la1aux=0:0.025:1
                    if FIN1==0
                        %el punto que genera (la1aux,la2aux) pertenece
a la curva de nivel asociada a f12(ei,eii)
                        ecu1= sqrt(((p1(1)+la1aux*d1(1))-A(ei,1)).^2 +
((p1(2)+la1aux*d1(2))-A(ei,2)).^2 ) ...
                            + alpha*(1-la1aux) * L1 + alpha*(la2auxx)
* L2 ...
                            + sqrt( (A(eii,1) -
(p2(1)+la2auxx*d2(1))).^2 + (A(eii,1)-(p2(2)+la2auxx*d2(2))).^2 ) -
D(ei,eii);
                        yg1=solve(ecu1,la2auxx,'Real',true);

                        for id=1:length(yg1)
                            if A2A==0
                                la2aux=yg1(id,1); %por si tiene mas de
una solucion ,coger la primera
                                la2aux=double(la2aux);
                                if la2aux>0 && la2aux<=1
                                    % si el punto que genera
(la1aux,la2aux)  de f21(ek,ekk) esta dentro de la curva de
f12(ei,eii)...meter (la1aux,la2aux)  en la nube de puntos de la curva
                                    if ( ((p1(1)+la1aux*d1(1))-
A(ek,1))^2 + ((p1(2)+la1aux*d1(2))-A(ek,2))^2 ) ...
                                        + alpha*(1-la1aux) * L1 +
alpha*(la2aux) * L2 ...
                                        + sqrt( (A(ekk,1)-
(p2(1)+la2aux*d2(1))).^2 + (A(ekk,1)-(p2(2)+la2aux*d2(2))).^2 ) <=
D(ek,ekk)

                                        mat(pp,5,p)=la1aux;
                                        mat(pp,6,p)=la2aux;
                                        A2A=1;
                                        FIN1=1;
                                    end
                                end
                            end
                        end
                    end
                end
            end

            %CASO A2B
            syms  la2auxx
            if pp==13 %ANALIZAMOS LAS POSIBLES SOLUCIONES DE LA
SEGUNDA ITERACION
                for la1aux=0:0.025:1
                    if FIN2==0
                        %el punto que genera (la1aux,la2aux) pertenece
a la curva de nivel asociada a f21(ei,eii)
                        ecu= ( sqrt( ((p2(1)+la2auxx*d2(1))-
A(ei,1)).^2 + ((p2(2)+la2auxx*d2(2))-A(ei,2)).^2 ) ...
                            + alpha*(1-la1aux) * L1 + alpha*(la2auxx)
* L2 ...
```

```
                            + sqrt( (A(eii,1)-(p1(1)+la1aux*d1(1))).^2
+ (A(eii,1)-(p1(2)+la1aux*d1(2))).^2 )- D(ei,eii));
                        yg=solve(ecu,la2auxx,'Real',true);

                        for id=1:length(yg)
                            if A2B==0
                                la2aux=yg(id,1);
                                la2aux=double(la2aux);
                                if la2aux>0 && la2aux<=1
                                    % si el punto que genera
(la1aux,la2aux)  de f21(ei,eii) esta dentro de la curva de
f12(ek,ekk)...meter (la1aux,la2aux)  en la nube de puntos de la curva
                                    if ( ((p1(1)+la1aux*d1(1))-
A(ek,1))^2 + ((p1(2)+la1aux*d1(2))-A(ek,2))^2 ) ...
                                            + alpha*(1-la1aux) * L1 +
alpha*(la2aux) * L2 ...
                                            + sqrt( (A(ekk,1)-
(p2(1)+la2aux*d2(1))).^2 + (A(ekk,1)-(p2(2)+la2aux*d2(2))).^2 ) <=
D(ek,ekk)

                                        mat(pp,5,p)=la1aux;
                                        mat(pp,6,p)=la2aux;
                                        A2B=1;
                                        FIN2=1;
                                    end
                                end
                            end
                        end
                    end
                end
            end


            %-----------------------------------4b
            %si no es ninguno de los casos anteriores... intersectan o
esta contenidos alguno dentro del otro se añade un punto de cada
curva:

            %CASO BA
            syms  la2auxx
            if pp==1
                for la1aux=0:0.025:1
                    if FIN1==0
                        %el punto que genera (la1aux,la2aux) pertenece
a la curva de nivel asociada a f12(ek,ekk)
                        ecu= sqrt( ((p1(1)+la1aux*d1(1))-A(ek,1)).^2 +
((p1(2)+la1aux*d1(2))-A(ek,2)).^2 ) ...
                            + alpha*(1-la1aux) * L1 + alpha*(la2auxx)
* L2 ...
                            + sqrt( (A(ekk,1)-
(p2(1)+la2auxx*d2(1))).^2 + (A(ekk,1)-(p2(2)+la2auxx*d2(2))).^2 )-
D(ek,ekk);
                        yg=solve(ecu,la2auxx,'Real',true);
                        for id=1:length(yg)
                            if BA1==0
                                la2aux=yg(id,1);
                                la2aux=double(la2aux);
                                if la2aux>0 && la2aux<=1
```

```
                                    mat(pp,5,p)=la1aux;
                                    mat(pp,6,p)=la2aux;
                                    BA1=1;
                                end
                            end
                        end
                        %el punto que genera (la1aux,la2aux) pertenece
a la curva de nivel asociada a f21(ei,eii)
                        if BA1==1
                            syms  la2auxx
                            ecu=sqrt( ((p2(1)+la2auxx*d2(1))-
A(ei,1)).^2 + ((p2(2)+la2auxx*d2(2))-A(ei,2)).^2 ) ...
                                + alpha*(1-la1aux) * L1 +
alpha*(la2auxx) * L2 ...
                                + sqrt( (A(eii,1)-
(p1(1)+la1aux*d1(1))).^2 + (A(eii,1)-(p1(2)+la1aux*d1(2))).^2 )-
D(ei,eii);
                            yg=solve(ecu,la2auxx,'Real',true);
                            for id=1:length(yg)
                                if BA2==0
                                    la2aux=yg(id,1);
                                    la2aux=double(la2aux);
                                    if la2aux>0 && la2aux<=1
                                        mat(pp+1,5,p)=la1aux;
                                        mat(pp+1,6,p)=la2aux;
                                        BA2=1;
                                        FIN1=1;
                                    end
                                end
                            end
                        end
                    end
                end
            end

            %CASO BB
            if pp==13
                for la1aux=0:0.025:1
                    if FIN2==0
                        syms  la2auxx
                        %el punto que genera (la1aux,la2aux) pertenece
a la curva de nivel asociada a f12(ei,eii)
                        ecu= sqrt( ((p1(1)+la1aux*d1(1))-A(ei,1)).^2 +
((p1(2)+la1aux*d1(2))-A(ei,2)).^2 ) ...
                            + alpha*(1-la1aux) * L1 + alpha*(la2auxx)
* L2 ...
                            + sqrt( (A(eii,1)-
(p2(1)+la2auxx*d2(1))).^2 + (A(eii,1)-(p2(2)+la2auxx*d2(2))).^2 )-
D(ei,ei);
                        yg=solve(ecu,la2auxx);
                        for id=1:length(yg)
                            if BB1==0
                                la2aux=yg(id,1);
                                la2aux=double(la2aux);
                                if la2aux>0 && la2aux<=1
                                    mat(pp,5,p)=la1aux;
                                    mat(pp,6,p)=la2aux;
                                    BB1=1;
```

```
                                end
                            end
                        end
                        %el punto que genera (la1aux,la2aux) pertenece
a la curva de nivel asociada a f21(ek,ekk)
                        if BB1==1
                            syms  la2auxx
                            ecu=sqrt( ((p2(1)+la2auxx*d2(1))-
A(ek,1)).^2 + ((p2(2)+la2auxx*d2(2))-A(ek,2)).^2 ) ...
                                + alpha*(1-la1aux) * L1 +
alpha*(la2auxx) * L2 ...
                                + sqrt( (A(ek,1)-
(p1(1)+la1aux*d1(1))).^2 + (A(ek,1)-(p1(2)+la1aux*d1(2))).^2 )-
D(ek,ekk);
                            yg=solve(ecu,la2auxx);
                            for id=1:length(yg)
                                if BB2==0
                                    la2aux=yg(id,1);
                                    la2aux=double(la2aux);
                                    if la2aux>0 && la2aux<=1
                                        mat(pp+1,5,p)=la1aux;
                                        mat(pp+1,6,p)=la2aux;
                                        BB2=1;
                                        FIN2=1;
                                    end
                                end
                            end

                        end
                    end
                end
            end


        end
    end
end

%% 5.EVALUAR LA FUNCION OBJETIVO

num_sol1=0;
num_sol2=0;
aux5=1;
%bucle para calcular el numero de posibles soluciones (la1,la2)
for aux5=1:NUM_ITER
    for jj=1:24
        if mat(jj,5,aux5)>0  &&  mat(jj,5,aux5)<=1  &&
mat(jj,6,aux5)>0  &&  mat(jj,6,aux5)<=1
            if jj < 13
                num_sol1=num_sol1+1;
            end
            if jj > 12
                num_sol2=num_sol2+1;
            end
        end
    end
end
```

```
if num_sol1==0
    num_sol1=1;
end
if num_sol2==0
    num_sol2=1;
end

sol1=zeros(1,3,num_sol1); % siendo los elementos : la1 , la2 , suma de
ti de la primera iteracion
sol2=zeros(1,3,num_sol2); % siendo los elementos : la1 , la2 , suma de
ti de la segunda iteracion

for aux6=1:NUM_ITER
    for jj=1:24
        if mat(jj,5,aux6)>0  &&  mat(jj,5,aux6)<=1  &&
mat(jj,6,aux6)>0  &&  mat(jj,6,aux6)<=1
            if jj<13
                sol1(1,1,aux6)=mat(jj,5,aux6);
                sol1(1,2,aux6)=mat(jj,6,aux6);
            end
            if jj>12
                sol2(1,1,aux6)=mat(jj,5,aux6);
                sol2(1,2,aux6)=mat(jj,6,aux6);
            end
        end
    end
end

for pla=1:num_sol1
    for i=1:numpueblos
        for j=1:numpueblos
            if(i<j)
                if sqrt( ((p1(1)+sol1(1,1,pla)*d1(1))-A(i,1)).^2 +
((p1(2)+sol1(1,1,pla)*d1(2))-A(i,2)).^2 ) ...
                        + alpha*(1-sol1(1,1,pla)) * L1 +
alpha*(sol1(1,2,pla)) * L2 ...
                        + sqrt( (A(j,1)-
(p2(1)+sol1(1,2,pla)*d2(1))).^2 + (A(j,1)-
(p2(2)+sol1(1,2,pla)*d2(2))).^2 ) <= D(i,j)
                    sol1(1,3,pla)=sol1(1,3,pla)+T(i,j);
                end

            end
        end
    end
end


for pla=1:num_sol2
    for i=1:numpueblos
        for j=1:numpueblos
            if(i<j)
                if  sqrt( ((p2(1)+sol2(1,2,pla)*d2(1))-A(i,1)).^2 +
((p2(2)+sol2(1,2,pla)*d2(2))-A(i,2)).^2 ) ...
                        + alpha*(1-sol2(1,1,pla)) * L1 +
alpha*(sol2(1,2,pla)) * L2 ...
```

```
                        + sqrt( (A(j,1)-
(p1(1)+sol2(1,1,pla)*d1(1))).^2 + (A(j,1)-
(p1(2)+sol2(1,1,pla)*d1(2))).^2 ) <= D(i,j)

                    sol2(1,3,pla)=sol2(1,3,pla)+T(i,j);
                end
            end
        end
    end
end

maximo=0;
indice=1;
maximo1=0;
indice1=1;
maximo2=0;
indice2=1;
xdi=[];
ydi=[];

for pla=1:num_sol1
    if sol1(1,3,pla)> maximo1
        maximo1=sol1(1,3,pla);
        indice1=pla;
    end
end
for pla=1:num_sol2
    if sol2(1,3,pla)> maximo2
        maximo2=sol2(1,3,pla);
        indice2=pla;
    end
end

if maximo1 >= maximo2
    maximo=maximo1;
    indice=indice1;
    sol=[sol1(1,1,indice),sol1(1,2,indice)];
end
if maximo1 < maximo2
    maximo=maximo2;
    indice=indice2;
     sol=[sol2(1,1,indice),sol2(1,2,indice)];
end




%% REPRESENTACIÓN

figure
tx=[p1(1),p2(1),p3(1)];
ty=[p1(2),p2(2),p3(2)];
plot(tx,ty,'k')
hold on
plot(p1(1),p1(2),'ok','LineWidth',3)
plot(p2(1),p2(2),'ok','LineWidth',3)
plot(p3(1),p3(2),'ok','LineWidth',3)
for i=1:num_sol1
```

```
    if (sol1(1,1,i)>0  &&  sol1(1,1,i)<=1)
        plot( (p1(1)+sol1(1,1,i)*d1(1)) , (p1(2)+sol1(1,1,i)*d1(2))
,'b.','LineWidth',3)
        plot( (p2(1)+sol1(1,2,i)*d2(1)) , (p2(2)+sol1(1,2,i)*d2(2))
,'c.','LineWidth',3)
    end


end
for i=1:num_sol2

    if (sol2(1,1,i)>0  &&  sol2(1,1,i)<=1)
        plot( (p1(1)+sol2(1,1,i)*d1(1)) , (p1(2)+sol2(1,1,i)*d1(2))
,'r.','LineWidth',3)
        plot( (p2(1)+sol2(1,2,i)*d2(1)) , (p2(2)+sol2(1,2,i)*d2(2))
,'m.','LineWidth',3)
    end


end
E1_SOL=[];
E2_SOL=[];
display('LAS LANDAS SON :')
LANDA1_SOL= sol(1,1)
LANDA2_SOL= sol(1,2)
 display('LAS ESTACIONES QUE MAXIMIZAN LA COBERTURA SON:')
E1_SOL(1)= p1(1)+LANDA1_SOL*d1(1);  E1_SOL(2)= p1(2)+LANDA1_SOL*d1(2);
E2_SOL(1)= p2(1)+LANDA2_SOL*d2(1);  E2_SOL(2)= p2(2)+LANDA2_SOL*d2(2);
E1_SOL
E2_SOL


plot(E1_SOL(1),E1_SOL(2),'oy','LineWidth',3)
plot(E2_SOL(1),E2_SOL(2),'oy','LineWidth',3)

%%
for j=1:numpueblos
    plot(A(j,1),A(j,2),'*g','LineWidth',7)
end
```

ANEXO C: CÓDIGO MATLAB PRINCIPAL PARA UNA RED TIPO ÁRBOL

```
clear all
close all
clc
format short
% A: matriz de pueblos ,
% D: matriz de distancias máximas válidas
% alpha: factor de velocidad
% T: matriz de demandas

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%DATOS (Ejemplo para una A, p1, p2, p3, ..., pn, D, T, alpha
concretos.)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

D= [ 0  4   6   8   10 ;
    0   0   6   5   6  ;
    0   0   0   4   9  ;
    0   0   0   0   4;
    0   0   0   0   0] ;

A=[1    3;
    3   -1;
    7.5 3;
    8   -1.5;
    7   -6];

T= [ 0   100    220    139   160     ;
    0   0    184   107    103   ;
    0   0    0    90     155;
    0   0    0    0      188;
    0     0   0   0   0

    alpha=0.2;

    [numpueblos,numcoord]=size(A);

    p1=[1 1];   p2=[2 0];   p3=[3.5 3.5];   p4=[6 -1];   p5=[ 6 -5];
p6=[9 0];
    puntos=[1 1;  2 0;   3.5 3.5;   6 -1;   6 -5;   9 0];
    mat_puntos= [p1 p2;  p2 p3;  p2 p4;  p4 p5;  p4 p6];
    mat_puntos_repr= [p1 p2;  p2 p3;  p3 p2;  p2 p4; p4 p5; p5 p4; p4
p6];
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %FIN DATOS
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    num_ramas= length (mat_puntos)
```

```
    for i=1:length(mat_puntos)
    mat_direcc(i,1) = (mat_puntos(i,3)-mat_puntos(i,1))  ;
    mat_direcc(i,2) = (mat_puntos(i,4)-mat_puntos(i,2))  ;
    mat_long(i,1) = sqrt(mat_direcc(i,1).^2 + mat_direcc(i,2).^2)  ;
    end
    % guardamos en mat_long por filas : la longitud  de cada rama

    longitudes=inf(num_ramas+1,num_ramas+1);% matriz que preparamos
con los pesos(Longitudes) que usaremos en el algoritmo de dijkstra
    n=size(longitudes,1);
    aaa=1;
    for i=1:n
        for j=1:n
            if i==j
                longitudes(i,j)=0;
            elseif i<j
                if mat_puntos(aaa,1)==puntos(i,1)  &&
mat_puntos(aaa,2)==puntos(i,2)       &&
mat_puntos(aaa,3)==puntos(j,1)  &&  mat_puntos(aaa,4)==puntos(j,2)
                    longitudes(i,j)=mat_long(aaa,1);
                    longitudes(j,i)=mat_long(aaa,1);

                    if aaa<num_ramas
                        aaa=aaa+1;
                    end
                end
            end
        end
    end

    %% Representacion de la situacion de los pueblos y las ramas
    tx=[];
    ty=[];
    for i=1:length(mat_puntos_repr)
        tx=[tx,mat_puntos_repr(i,1)];
        ty=[ty,mat_puntos_repr(i,2)];
        tx=[tx,mat_puntos_repr(i,3)];
        ty=[ty,mat_puntos_repr(i,4)];
    end
    plot(tx,ty,'k','LineWidth',1.5)
    grid
    hold on
    for i=1:length(puntos)
        plot(puntos(i,1),puntos(i,2),'ok','LineWidth',4)
    end
    for j=1:numpueblos
        plot(A(j,1),A(j,2),'*g','LineWidth',7)
    end
    %%

    %% A) OBTEMOS LA SOLUCION PARA CADA RAMA INVIVIDUALMENTE:
    candidatos_a=[];% x,y,demanda(T),pinicial, pfinal(estos dos
ultimos utiles para representacion)


    for i=1:num_ramas

        L=mat_long(i,1);
```

```
        pp1=[mat_puntos(i,1) , mat_puntos(i,2) ];
        pp2=[mat_puntos(i,3) , mat_puntos(i,4) ];

[candidatos_a(i,1),candidatos_a(i,2),candidatos_a(i,3)]=SEGMENTO_A_COM
PLETO(A,L,pp1,pp2,D,alpha,T);
        candidatos_a(i,4)=pp1(1);candidatos_a(i,5)=pp1(2);
        candidatos_a(i,6)=pp2(1);candidatos_a(i,7)=pp2(2);

    end

    % CALCULAR LA SOL DEL TIPO RAMA INDIVIDUAL QUE NOS DA UNA MAYOR
DEMANDA
    max_a=0;
    indi_a=1;
    for i=1:num_ramas
        if candidatos_a(i,3)>max_a
            indi_a= i;
        end
    end

    xa=candidatos_a(indi_a,1);
    ya=candidatos_a(indi_a,2);
    max_a=candidatos_a(indi_a,3);
    if xa>0 && ya>0
        est1_a(1)=candidatos_a(indi_a,4)+(candidatos_a(indi_a,6)-
candidatos_a(indi_a,4)) *xa/mat_long(indi_a,1);
        est1_a(2)=candidatos_a(indi_a,5)+(candidatos_a(indi_a,7)-
candidatos_a(indi_a,5)) *xa/mat_long(indi_a,1);
        est2_a(1)=candidatos_a(indi_a,4)+(candidatos_a(indi_a,6)-
candidatos_a(indi_a,4)) *ya/mat_long(indi_a,1);
        est2_a(2)=candidatos_a(indi_a,5)+(candidatos_a(indi_a,7)-
candidatos_a(indi_a,5)) *ya/mat_long(indi_a,1);
    end
    %% B) OBTEMOS LA SOLUCION PARA CADA PAR DE RAMAS QUE NO HAN DE SER
CONTIGUAS:


[mat_solu]=BORDES_A_COMPLETO(A,num_ramas,mat_puntos,mat_direcc,mat_lon
g,D,alpha,T,longitudes);

    % CALCULAR LA SOL DEL TIPO PAR DE RAMAS QUE NOS DA UNA MAYOR
DEMANDA
    num_casos=num_ramas*(num_ramas-1)/2;
    max_b=0;
    for auxi=1:num_casos
        if mat_solu(auxi,5)>=max_b ;%T maximo> maximo1
            max_b=mat_solu(auxi,5);
            la1=mat_solu(auxi,3);
            la2=mat_solu(auxi,4);
            p1x=mat_solu(auxi,6);% coord x del pto inicial de la rama
inicial
            p1y=mat_solu(auxi,7);% coord y del pto inicial de la rama
inicial
            d1x=mat_solu(auxi,8);% direcc x de la rama inicial
            d1y=mat_solu(auxi,9);% direcc y de la rama inicial
            p2x=mat_solu(auxi,10);% coord x del pto inicial de la rama
final
```

```
            p2y=mat_solu(auxi,11);% coord y del pto inicial de la rama
final
            d2x=mat_solu(auxi,12);% direcc x de la rama final
            d2y=mat_solu(auxi,13);% direcc y de la rama final
        end

    end
    est1_b=[(p1x +la1*d1x) , (p1y+la1*d1y)];
    est2_b=[(p2x +la2*d2x) , (p2y+la2*d2y)];


    %%
    if max_a>max_b
        plot( est1_a(1), est1_a(2) ,'r+','LineWidth',5)
        plot( est2_a(1), est2_a(2) ,'r+','LineWidth',5)
        display('LAS ESTACIONES QUE MAXIMIZAN LA COBERTURA SON:')
        est1_a
        est2_a
    end
    if max_a<=max_b
        plot( est1_b(1) , est1_b(2) ,'r+','LineWidth',5)
        plot( est2_b(1) , est2_b(2) ,'r+','LineWidth',5)
        display('LAS ESTACIONES QUE MAXIMIZAN LA COBERTURA SON:')
        est1_b
        est2_b
    end
```

ANEXO D: SUBCÓDIGO MATLAB PARA UNA RED TIPO ÁRBOL (UNA ARISTA)

```
function [X_SOL,Y_SOL,T_SOL] =SEGMENTO_A_COMPLETO(A,L,p1,p2,D,alpha,T)

%% Preparación de la rotación y translación de los pueblos
m1=(p2(2)-p1(2))/(p2(1)-p1(1));
if m1==Inf
    m1=99999;
elseif m1== -Inf
    m1=-99999;
end
m2=1;
tanbeta=abs((m2-m1)/(1+m1*m2));
beta1=(atan(tanbeta))*180/3.1415;
beta2=180-beta1;
if beta1<beta2
    p=beta1; g=beta2;
else
    p=beta2; g=beta1;
end

if p2(1)>=p1(1) && p2(2)>=p1(2) % 1er cuadrante
    if p2(1)>p2(2)
        beta=p;
        giro=beta;
    end
    if p2(1)<=p2(2)
        beta=p;
        giro=360-beta;
    end
end

if p2(1)<=p1(1) && p2(2)>=p1(2) % 2do cuadrante
    if abs(p2(1))<abs(p2(2))
        beta=p;
        giro=360-beta;
    end
    if abs(p2(1))>=abs(p2(2))
        beta=g;
        giro= 360-beta;
    end
end

if p2(1)<=p1(1) && p2(2)<=p1(2) % 3er cuadrante
    if abs(p2(1))>abs(p2(2))
        beta=g;
        giro=360-beta;
    end
    if abs(p2(1))<=abs(p2(2))
        beta=g;
```

```
        giro= beta;
    end
end

if p2(1)>=p1(1) && p2(2)<=p1(2) % 4to cuadrante
    if abs(p2(1))<abs(p2(2))
        beta=g;
        giro=beta;
    end
    if abs(p2(1))>=abs(p2(2))
        beta=p;
        giro= beta;
    end
end
giro;
traslx=-p1(1);
trasly=-p1(2);

[numpueblos,numcoord]=size(A);
girorad=giro*3.1415/180;
AA=zeros(numpueblos,2);
for i=1:numpueblos
    Axprima= A(i,1)*cos(girorad) - A(i,2)*sin(girorad)+traslx;% para
transformar la coordenadas de los pueblos en funcion del giro y
traslado q haga el segemnto a analizar
    Ayprima= A(i,1)*sin(girorad) + A(i,2)*cos(girorad)+trasly;
    AA(i,1)=Axprima;
    AA(i,2)=Ayprima;
end
%%
AA1=AA*10; AAo1=floor(AA1); AA=AAo1/10;
xxa=0:0.1:L; yya=0:0.1:L;
[xm4,ym4]=meshgrid(xxa,yya);

var=0;  % variable que usaremos para saber la dimension de f y d, de
forma que asi sean igual que "x" e "y" y se pueda operar
for i=0:0.1:L
    var=var+1;
end

%% 3.INTERSECCIONES

%calculamos el numero de intersecciones para poder dimensionar la
celda
%donde las almacenaremos
aux=1;
for k=1:numpueblos-2
    for kk=1:numpueblos
        if (k<kk)
            for i=1:numpueblos-1
                for ii=1:numpueblos
                    if( (i<ii && i>=k && ii>kk) || (i<ii && i>=k+1 )
)
                        aux=aux+1;
                    end
                end
            end
        end
```

```
    end
end
NUM_ITER=aux-1;


aux2=1;
aux4=1;
syms x y ya yb yc
xx=[];
yy=[];
xxx=[];
yyy=[];
mat=zeros(12,6,NUM_ITER);% los elementos son: k, kk, i, ii, x[], y[].

for k=1:numpueblos-2

    for kk=1:numpueblos
        if (k<kk)

            for i=1:numpueblos-1
                for ii=1:numpueblos
                    if( (i<ii && i>=k && ii>kk) || (i<ii && i>=k+1 )
)
                        disp([k kk i ii])
                        [xx,yy]=solve(sqrt((AA(k,1)-x)^2 +(AA(k,2))^2
)  +  alpha*(y-x) +  sqrt((AA(kk,1)-y)^2 +(AA(kk,2))^2 ) ==
D(k,kk),...
                            sqrt((AA(i,1)-x)^2 +(AA(i,2))^2 )  +
alpha*(y-x) +  sqrt((AA(ii,1)-y)^2 +(AA(ii,2))^2 ) ==
D(i,ii),'Real',true)
                        if  isempty(xx)
                            xx=0;   yy=0;
                        end
                        dim=length(xx);
                        aux3=0;

                        for aux2=1:dim
                            imagx=imag(xx(aux2));
                            imagy=imag(yy(aux2));
                            if (imagx == 0.00 && imagy == 0.00)%solo
lo consideramos solucion si la x e y no son imaginarios
                                if xx(aux2)>0 && xx(aux2)<L &&
yy(aux2)>0 && yy(aux2)<L
                                    aux3=aux3+1;
                                    xxx(aux3)=xx(aux2);
                                    yyy(aux3)=yy(aux2);
                                end
                            end
                        end

                        mat(1,1,aux4)=k;
                        mat(1,2,aux4)=kk;
                        mat(1,3,aux4)=i;
                        mat(1,4,aux4)=ii;
                        for p=1:aux3
                            mat(p,5,aux4)=xxx(p);
                            mat(p,6,aux4)=yyy(p);
```

```
                        end

                        aux4=aux4+1;
                    end
                end
            end

        end
    end
end



%% 4. SI NO INTERSECTAN:...es decir el elemento x e y es cero
for p=1:NUM_ITER
    if (mat(1,5,p)==0 &&  mat(1,6,p)==0);
        ek=mat(1,1,p);
        ekk=mat(1,2,p);
        ei=mat(1,3,p);
        eii=mat(1,4,p);
        %la interseccion que estudiaremos sera la de f(ek,ekk) vs
f(ei,eii)

        flag=0;
        flag1=0;
        flag2=0;
        flag3=0;
        flag4=0;
        flag5=0;

        %-----------------------------------4a.1
        if flag4==0
            for xaux=0:0.1:L
                ecu=sqrt((AA(ek,1)-xaux).^2 +(AA(ek,2)).^2 )  +
alpha* (y-xaux)  +  sqrt((AA(ekk,1)-y).^2 +(AA(ekk,2)).^2 )  -
D(ek,ekk);
                yg=solve(ecu,y,'Real',true);
                for id=1:length(yg)
                    if flag==0
                        yaux=yg(id,1);
                        yaux=double(yaux);
                        if imag(yaux)==0 && yaux<=L && xaux<yaux
                            %el punto (xaux,yaux) pertenece a la curva
de nivel asociada a f(ek,ekk)

                            if sqrt((AA(ei,1)-xaux).^2+(AA(ei,2)).^2 )
+ alpha* (yaux-xaux) + sqrt((AA(eii,1)-yaux).^2 +(AA(eii,2)).^2 ) <=
D(ei,eii)
                                % el punto xaux,yaux  de f (ek,ekk)
esta dentro de la curva de f(ei,eii)...meter xaux,yaux en la nube de
puntos de la curva
                                mat(1,5,p)=xaux;
                                mat(1,6,p)=yaux;
                                flag=1;
                                flag4=1;
                            end
                        end
                    end
```

```
                end
            end
        end
        %-----------------------------------4a.2
        if flag4==0
            for xaux1=0:0.1:L
                ecu1= sqrt((AA(ei,1)-xaux1).^2 +(AA(ei,2)).^2 )  +
alpha* (ya-xaux1)  +  sqrt((AA(eii,1)-ya).^2 +(AA(eii,2)).^2 )  -
D(ei,eii);
                yg1=solve(ecu1,ya,'Real',true);
                for id=1:length(yg1)
                    if flag1==0
                        yaux1=yg1(id,1); %por si tiene mas de una
solucion ,coger la primera

                        if imag(yaux1)==0 && yaux1<=L && xaux1<yaux1
                            %el punto (xaux,yaux) pertenece a la curva
de nivel asociada a f(ei,eii)
                            yaux1=double(yaux1);
                            if  sqrt((AA(ek,1)-xaux1).^2
+(AA(ek,2)).^2 )+alpha* (yaux1-xaux1)+sqrt((AA(ekk,1)-yaux1).^2
+(AA(ekk,2)).^2 )<= D(ek,ekk)
                                % el punto xaux,yaux  de f (ei,eii)
esta dentro de la curva de f(ek,ekk)...meter xaux,yaux en la nube de
puntos de la curva
                                mat(1,5,p)=xaux1;
                                mat(1,6,p)=yaux1;
                                flag1=1;
                                flag4=1;
                            end
                        end
                    end
                end
            end
        end
        %-----------------------------------4b
        %si no es ninguno de los casos anteriores... intersectan o
esta contenidos alguno dentro del otro se añade un punto de cada
curva:


        if(flag4==0)
            for xaux2=0:0.1:L
                ecu2= sqrt((AA(ek,1)-xaux2).^2 +(AA(ek,2)).^2 )  +
alpha* (yb-xaux2)  +  sqrt((AA(ekk,1)-yb).^2 +(AA(ekk,2)).^2 )  -
D(ek,ekk);
                yg2=solve(ecu2,yb,'Real',true);
                for id=1:length(yg2)
                    if flag2==0
                        yaux2=yg2(id,1);
                        if imag(yaux2)==0 && yaux2<=L && xaux2<yaux2
                            yaux2=double(yaux2);
                            mat(1,5,p)=xaux2;
                            mat(1,6,p)=yaux2;
                            flag2=1;
                            flag5=1;% condicion para entrar en el otro
bucle
                        end
```

```
                    end
                end
            end
        end
        if flag5==1
            for xaux3=0:0.1:L
                ecu3= sqrt((AA(ei,1)-xaux3).^2 +(AA(ei,2)).^2 )  +
alpha* (yc-xaux3)  +  sqrt((AA(eii,1)-yc).^2 +(AA(eii,2)).^2 )  -
D(ei,eii);
                yg3=solve(ecu3,yc,'Real',true);
                for id=1:length(yg3)
                    if flag3==0
                        yaux3=yg3(id,1);%por si tiene mas de una
solucion ,coger la primera
                        if imag(yaux3)==0 && yaux3<=L && xaux3<yaux3

                            yaux3=double(yaux3);
                            %el punto (xaux1,yaux1) pertenece a la
curva de nivel asociada a f(ei,eii) y hay que añadirlo a la nube de
puntos
                            mat(2,5,p)=xaux3;
                            mat(2,6,p)=yaux3;
                            flag3=1;
                            flag4=1;
                        end
                    end

                end
            end
        end
    end
end


%% 5.EVALUAR LA FUNCION OBJETIVO

num_sol=0;
aux5=1;
%bucle para calcular el numero de posibles soluciones (x,y)
for k=1:numpueblos-2
    for kk=1:numpueblos
        if (k<kk)

            for i=1:numpueblos-1
                for ii=1:numpueblos
                    if( (i<ii && i>=k && ii>kk) || (i<ii && i>=k+1 )
)

                        for jj=1:12
                            if ((mat(jj,5,aux5)~=0) && (
mat(jj,6,aux5)~=0 ))
                                num_sol=num_sol+1;
                            end
                        end
                        aux5=aux5+1;
                    end
                end
            end
```

```
        end
    end
end


sol=zeros(1,3,num_sol+1); % siendo los elementos : x , y, suma de ti
aux6=0;
for k=1:numpueblos-2
    for kk=1:numpueblos
        if (k<kk)

            for i=1:numpueblos-1
                for ii=1:numpueblos
                    if( (i<ii && i>=k && ii>kk) || (i<ii && i>=k+1 )
)
                        aux6=aux6+1;
                        for jj=1:12
                            if mat(jj,5,aux6)~=0 &&  mat(jj,6,aux6)~=0
&& imag(mat(jj,5,aux6))==0 &&  imag(mat(jj,6,aux6))==0
                                sol(1,1,aux6)=mat(jj,5,aux6);
                                sol(1,2,aux6)=mat(jj,6,aux6);
                            end

                        end
                    end
                end
            end
        end
    end
end

for pla=1:num_sol
    for i=1:numpueblos
        for j=1:numpueblos
            if(i<j)
                if ( AA(i,1) < AA(j,1) )  %si la coordenada "x" del
pueblo i es menor que la  coordenada "x" del pueblo j-> entra por i
sale por j
                    if ( sqrt( (AA(i,1)-sol(1,1,pla)).^2 +(AA(i,2)).^2
)  +  alpha* (sol(1,2,pla)- sol(1,1,pla))  +...
                            sqrt( (AA(j,1)-sol(1,2,pla)).^2
+(AA(j,2)).^2 ) <= D(i,j) )

                        sol(1,3,pla)=sol(1,3,pla)+T(i,j);

                    end
                else                    %si la coordenada "x" del
pueblo j es menor o igual  que la  coordenada "x" del pueblo i-> entra
por j sale por i
                    if ( sqrt( (AA(j,1)-sol(1,1,pla)).^2 +(AA(j,2)).^2
)  +  alpha* (sol(1,2,pla)-sol(1,1,pla))  + ...
                            sqrt( (AA(i,1)-sol(1,2,pla)).^2
+(AA(i,2)).^2 )  <= D(i,j) )

                        sol(1,3,pla)=sol(1,3,pla)+T(i,j);
                    end
                end
            end
```

```
        end
    end
end



if num_sol>=1
    for i=1:num_sol
        if ((sol(1,1,i)~=0) && ( mat(1,2,i)~=0 ))
            xdi=sol(1,1,i);
            ydi=sol(1,2,i);
        end
        % % % % % % % % % % % % % % %
plot(xdi,xdi,'b.','LineWidth',3)
        % % % % % % % % % % % % % % %
plot(ydi,ydi,'r.','LineWidth',3)
    end
    maximo=0;
    indice=1;
    for pla=1:num_sol
        if sol(1,3,pla)> maximo
            maximo=sol(1,3,pla);
            indice=pla;
        end
    end

    X_SOL= sol(1,1,indice);
    Y_SOL= sol(1,2,indice);
    T_SOL= sol(1,3,indice);

end
clear xx yy xxx yyy
clear x y ya yb yc

if num_sol<1 % este segmento no aporta soluciones
    X_SOL=0;
    Y_SOL=0;
    T_SOL=0;
end
```

ANEXO E: SUBCÓDIGO MATLAB PARA UNA RED TIPO ÁRBOL (DOS ARISTAS)

```
function [mat_solu,mat] =
BORDES_A_COMPLETO(A,num_ramas,mat_puntos,mat_long,D,alpha,T,longitudes
)
format short
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[numpueblos,numcoord]=size(A);

%%% calculo de la longitud intermedia
num_casos=num_ramas*(num_ramas-1)/2;
mat_L=zeros(num_casos,5) ;%(a,b, L_interm,L1,L2) caso en el que
analizemos: inicio en rama a y fin en rama b ... long interm obtenida
lrama1 y lrama2
aux=0;
cont=1;

%%   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% calculo de la longitud intermedia mas corta a partir del algoritmo
de dijkstra
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
conti=0;
for a=1:num_ramas -1
    for b=1:num_ramas
        if a<b
            conti=conti+1;
            if a==b-1
                mat_L(conti,3)=0;
            else

                n=size(longitudes,1);

                S(1:n) = 0;     % vector de nudos visitados :  0 no; 1
si
                dist(1:n) = inf;   % distancia mas corta desde el nudo
de inicio a al nudo de destino b
                prev(1:n) = n+1;    %se guarda el nudo del vertice
anterior

                dist(a) = 0;

                while sum(S)~=n % mientras quede algun nodo por
visitar
                    candidate=[];
                    for i=1:n
                        if S(i)==0
                            candidate=[candidate dist(i)];
                        else
```

```
                            candidate=[candidate inf];
                        end
                    end
                    [u_index u]=min(candidate);

                    S(u)=1;


                    for i=1:n
                        if(dist(u)+longitudes(u,i))<dist(i)
                            dist(i)=dist(u)+longitudes(u,i);
                            prev(i)=u;
                        end
                    end
                end

                ruta = [b];

                while ruta(1) ~= a
                    if prev(ruta(1))<=n
                        ruta=[prev(ruta(1)) ruta];
                    end
                end;

                mr=min(ruta);
                valor= dist(b)-mat_long(mr,1)-mat_long(b-1,1)

                if valor <=0
                    mat_L(conti,3)=0;
                else
                    mat_L(conti,3)=valor;
                end
            end
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%calculo de L1 y L2
contad=1;
for i=1:num_ramas-1
    for j=1:num_ramas
        if i<j
            mat_L(contad,1)=i;
            mat_L(contad,2)=j;
            mat_L(contad,4)=mat_long(i,1);
            mat_L(contad,5)=mat_long(j,1);
            contad=contad+1;
        end
    end
end


%% 3.INTERSECCIONES

%calculamos el numero de intersecciones para poder dimensionar la
```

```
%híper matriz donde las almacenaremos
mat_solu=zeros(num_casos,13); % todos los elementos estan explicados
en el momento de la asignacion ... aprox linea 710
conta=1;
for indi=1:num_ramas-1
    for indj=1:num_ramas

        if indi<indj
            p1ini=[mat_puntos(indi,1),mat_puntos(indi,2)];
            p1=p1ini;
            p1fin=[mat_puntos(indi,3),mat_puntos(indi,4)];
            d1=p1fin-p1ini;
            p2ini=[mat_puntos(indj,1),mat_puntos(indj,2)];
            p2=p2ini;
            p2fin=[mat_puntos(indj,3),mat_puntos(indj,4)];
            d2=p2fin-p2ini;


            L1=mat_L(conta,4)*1000; Lo1=floor(L1); L1=Lo1/1000
%Cogemos solo 4 cifras significativas y asi evitamos que el programa
se atasque en los bucles
            L2=mat_L(conta,5)*1000; Lo2=floor(L2); L2=Lo2/1000
            L_interm=mat_L(conta,3)*1000; Lointerm=floor(L_interm);
L_interm=Lointerm/1000%Cogemos solo 4 cifras significativas y asi
evitamos que el programa se atasque en los bucles
            L=L1+L2+L_interm;

            aux=0;
            for k=1:numpueblos-2
                for kk=1:numpueblos
                    if (k<kk)
                        for i=1:numpueblos-1
                            for ii=1:numpueblos
                                if( (i<ii && i>=k && ii>kk) || (i<ii
&& i>=k+1 )  )
                                    aux=aux+1;
                                end
                            end
                        end
                    end
                end
            end
            NUM_ITER=aux;

            mat = zeros(24,6,NUM_ITER);% los elementos son: k, kk, i,
ii, la1_iter1[] y la1_iter2[] , de la fila 1-12 para ir de 1 a 2 ,de
la fila 13-24 de 2 a 1

            e1x=[];     e1y=[];
            e2x=[];     e2y=[];
            xxx1=[];    yyy1=[];
            xxx2=[];    yyy2=[];

            aux4=0;
            syms la1 la2
            for k=1:numpueblos-2
                for kk=1:numpueblos
                    if (k<kk)
```

```
                        for i=1:numpueblos-1
                            for ii=1:numpueblos
                                if( (i<ii && i>=k && ii>kk) || (i<ii
&& i>=k+1 )  )

                                    aux4=aux4+1;

                                    [e1x,e1y]=solve(sqrt( ((p1(1)+
d1(1)*la1)-A(k,1)).^2  +  ((p1(2)+ d1(2)*la1)-A(k,2)).^2  ) ...
                                        + alpha*(1-la1) * L1 +
alpha*(la2) * L2 + alpha*L_interm...
                                        + sqrt( (A(kk,1)-(p2(1)+
la2*d2(1))) .^2  +  (A(kk,2)-(p2(2)+la2*d2(2))).^2 )  ==  D(k,kk), ...
                                        ...
                                        sqrt( ((p1(1)+ d1(1)*la1)-
A(i,1)).^2  +  ((p1(2)+ d1(2)*la1)-A(i,2)).^2  ) ...
                                        + alpha*(1-la1) * L1 +
alpha*(la2) * L2+ alpha*L_interm...
                                        + sqrt( (A(ii,1)-(p2(1)+
la2*d2(1))) .^2  +  (A(ii,2)-(p2(2)+la2*d2(2))).^2 )  ==
D(i,ii),'Real',true);

                                    % Ai -> e2 -> p2 -> e1 -> Aj
                                    [e2x,e2y]=solve( sqrt( ((p2(1)+
la2*d2(1))-A(k,1)).^2  +  ((p2(2)+ la2*d2(2))-A(k,2)).^2  ) ...
                                        + alpha*(la2) * L2 + alpha*(1-
la1) * L1 + alpha*L_interm ...
                                        + sqrt( (A(kk,1)-(p1(1)+
la1*d1(1))) .^2  +  (A(kk,2)-(p1(2)+la1*d1(2))).^2 )  ==  D(k,kk),...
                                        ...
                                        sqrt( ((p2(1)+ la2*d2(1))-
A(i,1)).^2  +  ((p2(2)+ la2*d2(2))-A(i,2)).^2  ) ...
                                        + alpha*(la2) * L2 + alpha*(1-
la1) * L1 + alpha*L_interm...
                                        + sqrt( (A(ii,1)-(p1(1)+
la1*d1(1))) .^2  +  (A(ii,2)-(p1(2)+la1*d1(2))).^2 )  ==
D(i,ii),'Real',true) ;

                                    dim1=length(e1x);
                                    dim2=length(e2x);

                                    aux3=0;
                                    for aux2=1:dim1
                                        if (e1x(aux2)<=1  &&
e1x(aux2)>0  &&  e1y(aux2)<=1  &&  e1y(aux2)>0) % solo las soluciones
que nos permitan tener la estacion dentro de nuestro segmento
                                            aux3=aux3+1;
                                            xxx1(aux3)=e1x(aux2);
                                            yyy1(aux3)=e1y(aux2);
                                        end
                                    end

                                    aux6=0;
                                    for aux5=1:dim2
                                        if (e2x(aux5)<=1  &&
e2x(aux5)>0  &&  e2y(aux5)<=1  &&  e2y(aux5)>0) % solo las soluciones
que nos permitan tener la estacion dentro de nuestro segmento
```

```
                                        aux6=aux6+1;
                                        xxx2(aux6)=e2x(aux5);
                                        yyy2(aux6)=e2y(aux5);
                                    end
                                end

                                mat(1,1,aux4)=k;
                                mat(1,2,aux4)=kk;
                                mat(1,3,aux4)=i;
                                mat(1,4,aux4)=ii;

                                for p=1:aux3
                                    mat(p,5,aux4)=xxx1(p);
                                    mat(p,6,aux4)=yyy1(p);
                                end

                                for p=1:aux6
                                    mat(12+p,5,aux4)=xxx2(p);
                                    mat(12+p,6,aux4)=yyy2(p);
                                end



                            end
                        end
                    end

                end
            end
        end


        %% 4. SI NO INTERSECAN:...es decir los elementos
la1_iter1[], la2_iter1[]
        % son cero la1[] y la2 de alguno de los sentidos del
trayecto:

        for p=1:NUM_ITER % cada plano
            for pp=1:12:13   % analizar el primer elemento ,1 y
13, de cada tipo de interseccion
                if ((mat(pp,5,p)==0  &&  mat(pp,6,p)==0))

                    ek=mat(1,1,p);
                    ekk=mat(1,2,p);
                    ei=mat(1,3,p);
                    eii=mat(1,4,p);

                    %% FLAGS QUE NECESITAMOS:
                    A1A=0;  A1B=0;  A2A=0;  A2B=0;  BA1=0;
BB1=0;   BA2=0;   BB2=0;   FIN1=0; FIN2=0;

                    %----------------------------------4a.1

                    % CASO A1A
                    if pp==1 %ANALIZAMOS LAS POSIBLES SOLUCIONES
DE LA PRIMERA ITERACION
                        for la1aux=0:0.025:1
                            if FIN1==0
```

```
                                    %el punto que genera
(la1aux,la2aux) pertenece a la curva de nivel asociada a f12(ek,ekk)
                                    clear la2auxx
                                    syms  la2auxx
                                    ecu= sqrt( ((p1(1)+la1aux*d1(1))-
A(ek,1)).^2 + ((p1(2)+la1aux*d1(2))-A(ek,2)).^2 ) ...
                                        + alpha*(1-la1aux) * L1 +
alpha*(la2auxx) * L2+ alpha*L_interm...
                                        + sqrt( (A(ekk,1)-
(p2(1)+la2auxx*d2(1))).^2 + (A(ekk,1)-(p2(2)+la2auxx*d2(2))).^2 )-
D(ek,ekk);
                                    yg=solve(ecu,la2auxx,'Real',true);

                                    for id=1:length(yg)
                                        if A1A==0
                                            la2aux=yg(id,1);
                                            la2aux=double(la2aux);
                                            if la2aux>0 && la2aux<=1
                                                % si el punto que
genera (la1aux,la2aux)  de f12(ek,ekk) esta dentro de la curva de
f21(ei,eii)...meter (la1aux,la2aux)  en la nube de puntos de la curva
                                                if (sqrt(
((p1(1)+la1aux*d1(1))-A(ei,1)).^2 + ((p1(2)+la1aux*d1(2))-A(ei,2)).^2
) ...
                                                    + alpha*(1-
la1aux) * L1 + alpha*(la2aux) * L2+ alpha*L_interm...
                                                    + sqrt(
(A(eii,1)-(p2(1)+la2aux*d2(1))).^2 + (A(eii,1)-
(p2(2)+la2aux*d2(2))).^2 ) <= D(ei,eii))


mat(pp,5,p)=la1aux;

mat(pp,6,p)=la2aux;
                                                    A1A=1;
                                                    FIN1=1;
                                                end
                                            end
                                        end
                                    end
                                end
                            end
                        end

                        % CASO A1B

                        clear  la2auxx
                        syms  la2auxx
                        if pp==13 %ANALIZAMOS LAS POSIBLES SOLUCIONES
DE LA SEGUNDA ITERACION
                            for la1aux=0:0.025:1
                                if FIN2==0
                                    %el punto que genera
(la1aux,la2aux) pertenece a la curva de nivel asociada a f21(ek,ekk)
                                    ecu= ( sqrt(
((p2(1)+la2auxx*d2(1))-A(ek,1)).^2 + ((p2(2)+la2auxx*d2(2))-
A(ek,2)).^2 ) ...
```

```
                                        + alpha*(1-la1aux) * L1 +
alpha*(la2auxx) * L2 + alpha*L_interm...
                                        + sqrt( (A(ekk,1)-
(p1(1)+la1aux*d1(1))).^2 + (A(ekk,1)-(p1(2)+la1aux*d1(2))).^2 )-
D(ek,ekk));
                                    yg=solve(ecu,la2auxx,'Real',true);

                                    for id=1:length(yg)
                                        if A1B==0
                                            la2aux=yg(id,1);
                                            la2aux=double(la2aux);
                                            if la2aux>0 && la2aux<=1
                                                % si el punto que
genera (la1aux,la2aux)  de f21(ek,ekk) esta dentro de la curva de
f12(ei,eii)...meter (la1aux,la2aux)  en la nube de puntos de la curva
                                                if (sqrt(
((p2(1)+la2aux*d2(1))-A(ei,1)).^2 + ((p2(2)+la2aux*d2(2))-A(ei,2)).^2
) ...
                                                    + alpha*(1-
la1aux) * L1 + alpha*(la2aux) * L2 + alpha*L_interm...
                                                    + sqrt(
(A(eii,1)-(p1(1)+la1aux*d1(1))).^2 + (A(eii,1)-
(p1(2)+la1aux*d1(2))).^2 )<= D(ei,eii))


mat(pp,5,p)=la1aux;

mat(pp,6,p)=la2aux;
                                                    A1B=1;
                                                    FIN2=1;
                                                end
                                            end
                                        end
                                    end
                                end
                            end
                        end


                        %-----------------------------------4a.2

                        %CASO A2A
                        clear la2auxx
                        syms  la2auxx
                        if pp==1 %ANALIZAMOS LAS POSIBLES SOLUCIONES
DE LA PRIMERA ITERACION
                            for la1aux=0:0.025:1
                                if FIN1==0
                                    %el punto que genera
(la1aux,la2aux) pertenece a la curva de nivel asociada a f12(ei,eii)
                                    ecu1= sqrt(((p1(1)+la1aux*d1(1))-
A(ei,1)).^2 + ((p1(2)+la1aux*d1(2))-A(ei,2)).^2 ) ...
                                        + alpha*(1-la1aux) * L1 +
alpha*(la2auxx) * L2 + alpha*L_interm...
                                        + sqrt( (A(eii,1) -
(p2(1)+la2auxx*d2(1))).^2 + (A(eii,1)-(p2(2)+la2auxx*d2(2))).^2 ) -
D(ei,eii);
```

```
yg1=solve(ecu1,la2auxx,'Real',true);

                                    for id=1:length(yg1)
                                        if A2A==0
                                            la2aux=yg1(id,1); %por si
tiene mas de una solucion ,coger la primera
                                            la2aux=double(la2aux);
                                            if la2aux>0 && la2aux<=1
                                                % si el punto que
genera (la1aux,la2aux)  de f21(ek,ekk) esta dentro de la curva de
f12(ei,eii)...meter (la1aux,la2aux)  en la nube de puntos de la curva
                                                if (
sqrt(((p1(1)+la1aux*d1(1))-A(ek,1)).^2 + ((p1(2)+la1aux*d1(2))-
A(ek,2)).^2 ) ...
                                                        + alpha*(1-
la1aux) * L1 + alpha*(la2aux) * L2 + alpha*L_interm...
                                                        + sqrt(
(A(ekk,1) -(p2(1)+la2aux*d2(1))).^2 + (A(ekk,1)-
(p2(2)+la2aux*d2(2))).^2 ) <= D(ek,ekk));


mat(pp,5,p)=la1aux;

mat(pp,6,p)=la2aux;
                                                    A2A=1;
                                                    FIN1=1;
                                                end
                                            end
                                        end
                                    end
                                end
                            end
                        end

                        %CASO A2B
                        clear la2auxx
                        syms  la2auxx
                        if pp==13 %ANALIZAMOS LAS POSIBLES SOLUCIONES
DE LA SEGUNDA ITERACION
                            for la1aux=0:0.025:1
                                if FIN2==0
                                    %el punto que genera
(la1aux,la2aux) pertenece a la curva de nivel asociada a f21(ei,eii)
                                    ecu= ( sqrt(
((p2(1)+la2auxx*d2(1))-A(ei,1)).^2 + ((p2(2)+la2auxx*d2(2))-
A(ei,2)).^2 ) ...
                                        + alpha*(1-la1aux) * L1 +
alpha*(la2auxx) * L2 + alpha*L_interm...
                                        + sqrt( (A(eii,1)-
(p1(1)+la1aux*d1(1))).^2 + (A(eii,1)-(p1(2)+la1aux*d1(2))).^2 )-
D(ei,eii));
                                    yg=solve(ecu,la2auxx,'Real',true);

                                    for id=1:length(yg)
                                        if A2B==0
                                            la2aux=yg(id,1);
                                            la2aux=double(la2aux);
```

```
                                            if la2aux>0 && la2aux<=1
                                                % si el punto que
genera (la1aux,la2aux)  de f21(ei,eii) esta dentro de la curva de
f12(ek,ekk)...meter (la1aux,la2aux)  en la nube de puntos de la curva
                                                if ( sqrt(
((p2(1)+la2aux*d2(1))-A(ek,1)).^2 + ((p2(2)+la2aux*d2(2))-A(ek,2)).^2
) ...
                                                    + alpha*(1-
la1aux) * L1 + alpha*(la2aux) * L2 + alpha*L_interm...
                                                    + sqrt(
(A(ekk,1)-(p1(1)+la1aux*d1(1))).^2 + (A(ekk,1)-
(p1(2)+la1aux*d1(2))).^2 )<= D(ek,ekk))


mat(pp,5,p)=la1aux;

mat(pp,6,p)=la2aux;
                                                    A2B=1;
                                                    FIN2=1;
                                                end
                                            end
                                        end
                                    end
                                end
                            end
                        end


                        %-----------------------------------4b
                        %si no es ninguno de los casos anteriores...
intersectan o esta contenidos alguno dentro del otro se añade un punto
de cada curva:

                        %CASO BA
                        clear la2auxx
                        syms  la2auxx
                        if pp==1
                            for la1aux=0.025:0.025:1
                                if FIN1==0
                                    %el punto que genera
(la1aux,la2aux) pertenece a la curva de nivel asociada a f12(ek,ekk)
                                    ecu= sqrt( ((p1(1)+la1aux*d1(1))-
A(ek,1)).^2 + ((p1(2)+la1aux*d1(2))-A(ek,2)).^2 ) ...
                                        + alpha*(1-la1aux) * L1 +
alpha*(la2auxx) * L2 + alpha*L_interm...
                                        + sqrt( (A(ekk,1)-
(p2(1)+la2auxx*d2(1))).^2 + (A(ekk,1)-(p2(2)+la2auxx*d2(2))).^2 )-
D(ek,ekk);
                                    yg=solve(ecu,la2auxx,'Real',true);
                                    for id=1:length(yg)
                                        if BA1==0
                                            la2aux=yg(id,1);
                                            la2aux=double(la2aux);
                                            if la2aux>0 && la2aux<=1
                                                mat(pp,5,p)=la1aux;
                                                mat(pp,6,p)=la2aux;
                                                BA1=1;
                                            end
```

```
                                        end
                                    end
                                    %el punto que genera
(la1aux,la2aux) pertenece a la curva de nivel asociada a f21(ei,eii)
                                    if BA1==1
                                        clear la2auxx
                                        syms  la2auxx

                                        ecu= sqrt(
((p1(1)+la1aux*d1(1))-A(ei,1)).^2 + ((p1(2)+la1aux*d1(2))-A(ei,2)).^2
) ...
                                            + alpha*(1-la1aux) * L1 +
alpha*(la2auxx) * L2 + alpha*L_interm...
                                            + sqrt( (A(eii,1)-
(p2(1)+la2auxx*d2(1))).^2 + (A(eii,1)-(p2(2)+la2auxx*d2(2))).^2 )-
D(ei,eii);


yg=solve(ecu,la2auxx,'Real',true);
                                        for id=1:length(yg)
                                            if BA2==0
                                                la2aux=yg(id,1);
                                                la2aux=double(la2aux);
                                                if la2aux>0 &&
la2aux<=1

mat(pp+1,5,p)=la1aux;

mat(pp+1,6,p)=la2aux;
                                                    BA2=1;
                                                    FIN1=1;
                                                end
                                            end
                                        end
                                    end
                                end
                            end
                        end

                        %CASO BB
                        if pp==13
                            for la1aux=0.025:0.025:1
                                if FIN2==0
                                    clear la2auxx
                                    syms  la2auxx
                                    %el punto que genera
(la1aux,la2aux) pertenece a la curva de nivel asociada a f12(ei,eii)


                                    ecu=sqrt( ((p2(1)+la2auxx*d2(1))-
A(ek,1)).^2 + ((p2(2)+la2auxx*d2(2))-A(ek,2)).^2 ) ...
                                        + alpha*(1-la1aux) * L1 +
alpha*(la2auxx) * L2+ alpha*L_interm...
                                        + sqrt( (A(ekk,1)-
(p1(1)+la1aux*d1(1))).^2 + (A(ekk,1)-(p1(2)+la1aux*d1(2))).^2 )-
D(ek,ekk);
                                    yg=solve(ecu,la2auxx,'Real',true);
                                    for id=1:length(yg)
```

```
                                        if BB1==0
                                            la2aux=yg(id,1);
                                            la2aux=double(la2aux);
                                            if la2aux>0 && la2aux<=1
                                                mat(pp,5,p)=la1aux;
                                                mat(pp,6,p)=la2aux;
                                                BB1=1;
                                            end
                                        end
                                    end
                                    %el punto que genera
(la1aux,la2aux) pertenece a la curva de nivel asociada a f21(ek,ekk)
                                    if BB1==1
                                        clear la2uxx
                                        syms  la2auxx
                                        ecu=sqrt(
((p2(1)+la2auxx*d2(1))-A(ek,1)).^2 + ((p2(2)+la2auxx*d2(2))-
A(ek,2)).^2 ) ...
                                            + alpha*(1-la1aux) * L1 +
alpha*(la2auxx) * L2+ alpha*L_interm...
                                            + sqrt( (A(ekk,1)-
(p1(1)+la1aux*d1(1))).^2 + (A(ekk,1)-(p1(2)+la1aux*d1(2))).^2 )-
D(ek,ekk);

yg=solve(ecu,la2auxx,'Real',true);
                                        for id=1:length(yg)
                                            if BB2==0
                                                la2aux=yg(id,1);
                                                la2aux=double(la2aux);
                                                if la2aux>0 &&
la2aux<=1

mat(pp+1,5,p)=la1aux;

mat(pp+1,6,p)=la2aux;
                                                    BB2=1;
                                                    FIN2=1;
                                                end
                                            end
                                        end

                                    end
                                end
                            end
                        end


                    end
                end
            end


            %%
            % 5.EVALUAR LA FUNCION OBJETIVO

            num_sol1=0;
            num_sol2=0;
```

```
            %bucle para calcular el numero de posibles soluciones
(la1,la2)
            for aux5=1:NUM_ITER
                for jj=1:24
                    if ((mat(jj,5,aux5)>0  &&  mat(jj,5,aux5)<=1  &&
mat(jj,6,aux5)>0  &&  mat(jj,6,aux5)<=1 ) || ...
                                (mat(jj,5,aux5)>=0  &&  mat(jj,5,aux5)<=1
&&  mat(jj,6,aux5)>0  &&  mat(jj,6,aux5)<=1 ) ||...
                                (mat(jj,5,aux5)>0  &&  mat(jj,5,aux5)<=1
&&  mat(jj,6,aux5)>=0  &&  mat(jj,6,aux5)<=1 ))
                        if jj < 13
                            num_sol1=num_sol1+1;
                        end
                        if jj > 12
                            num_sol2=num_sol2+1;
                        end
                    end
                end
            end



            sol1=zeros(1,3,num_sol1); % siendo los elementos : la1 ,
la2 , suma de ti de la primera iteracion
            sol2=zeros(1,3,num_sol2); % siendo los elementos : la1 ,
la2 , suma de ti de la segunda iteracion

            aux66=1;
            for aux6=1:NUM_ITER
                for jj=1:24
                    if (mat(jj,5,aux6)>0  &&  mat(jj,6,aux6)>0  ||...
                            (mat(jj,5,aux5)>=0   &&  mat(jj,6,aux5)>0)
||...
                            (mat(jj,5,aux5)>0  &&  mat(jj,6,aux5)>=0
))
                        if jj<13
                            sol1(1,1,aux66)=mat(jj,5,aux6);
                            sol1(1,2,aux66)=mat(jj,6,aux6);
                        end
                        if jj>12
                            sol2(1,1,aux66)=mat(jj,5,aux6);
                            sol2(1,2,aux66)=mat(jj,6,aux6);
                        end
                        aux66=aux66+1;
                    end
                end
            end

            if num_sol1~=0
                for pla=1:num_sol1
                    for i=1:numpueblos
                        for j=1:numpueblos
                            if(i<j)
                                if sqrt( ((p1(1)+sol1(1,1,pla)*d1(1))-
A(i,1)).^2 + ((p1(2)+sol1(1,1,pla)*d1(2))-A(i,2)).^2 ) ...
                                        + alpha*(1-sol1(1,1,pla)) * L1
+ alpha*(sol1(1,2,pla)) * L2+ alpha*L_interm....
```

```
                                        + sqrt( (A(j,1)-
(p2(1)+sol1(1,2,pla)*d2(1))).^2 + (A(j,1)-
(p2(2)+sol1(1,2,pla)*d2(2))).^2 ) <= D(i,j)

sol1(1,3,pla)=sol1(1,3,pla)+T(i,j);
                                end
                            end
                        end
                    end
                end
            end

            if num_sol2~=0
                for pla=1:num_sol2
                    for i=1:numpueblos
                        for j=1:numpueblos
                            if(i<j)
                                if  sqrt(
((p2(1)+sol2(1,2,pla)*d2(1))-A(i,1)).^2 +
((p2(2)+sol2(1,2,pla)*d2(2))-A(i,2)).^2 ) ...
                                        + alpha*(1-sol2(1,1,pla)) * L1
+ alpha*(sol2(1,2,pla)) * L2 + alpha*L_interm...
                                        + sqrt( (A(j,1)-
(p1(1)+sol2(1,1,pla)*d1(1))).^2 + (A(j,1)-
(p1(2)+sol2(1,1,pla)*d1(2))).^2 ) <= D(i,j)


sol2(1,3,pla)=sol2(1,3,pla)+T(i,j);
                                end
                            end
                        end
                    end
                end
            end

            maximo=0;
            indice=0;
            maximo1=0;
            indice1=0;
            maximo2=0;
            indice2=0;
            xdi=[];
            ydi=[];

            if num_sol1~=0
                for pla=1:num_sol1
                    if sol1(1,3,pla)> maximo1
                        maximo1=sol1(1,3,pla);
                        indice1=pla;
                    end
                end
            end


            if num_sol2~=0
                for pla=1:num_sol2
                    if sol2(1,3,pla)> maximo2
                        maximo2=sol2(1,3,pla);
```

```
                    indice2=pla;
                end
            end
        end

        if maximo1 > maximo2
            maximo=maximo1;
            indice=indice1;
            if indice ==0
                sol=[0,0];
            else

                sol=[sol1(1,1,indice),sol1(1,2,indice)]
            end
        end
        if maximo1 <= maximo2
            maximo=maximo2;
            indice=indice2;
            if indice ==0
                sol=[0,0];
            else
                sol=[sol2(1,1,indice),sol2(1,2,indice)]
            end
        end

        mat_solu(conta,1)=indi;% rama de inicial
        mat_solu(conta,2)=indj;% ultima rama
        mat_solu(conta,3)=sol(1,1);% la1
        mat_solu(conta,4)=sol(1,2);% la2
        mat_solu(conta,5)=maximo;%T maximo
        mat_solu(conta,6)=p1(1,1);% coord x del pto inicial de la
rama inicial
        mat_solu(conta,7)=p1(1,2);% coord y del pto inicial de la
rama inicial
        mat_solu(conta,8)=d1(1,1);% direcc x de la rama inicial
        mat_solu(conta,9)=d1(1,2);% direcc y de la rama inicial
        mat_solu(conta,10)=p2(1,1);% coord x del pto inicial de la
rama final
        mat_solu(conta,11)=p2(1,2);% coord y del pto inicial de la
rama final
        mat_solu(conta,12)=d2(1,1);% direcc x de la rama final
        mat_solu(conta,13)=d2(1,2);% direcc y de la rama final
        conta=conta+1;

    end
  end
end
```