FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO
MAESTRÍA EN OPTIMIZACIÓN Y CÓMPUTO APLICADO

DISEÑO DE UN ALGORITMO EN 2 FASES PARA UN MTSP

T E S I S

QUE PARA OBTENER EL GRADO DE MAESTRO EN
OPTIMIZACIÓN Y CÓMPUTO APLICADO

PRESENTA:

VÍCTOR HUGO PACHECO VALENCIA

DIRECTOR DE TESIS:

DR. NODARI VAKHANIA MAISURADZE

CUERNAVACA MOR. SEPTIEMBRE DE 2019

Resumen

El Problema de los Vendedores Viajeros, MTSP, consiste en construir o encontrar una secuencia de clientes por visitar para cada vendedor, partiendo y concluyendo todos ellos su recorrido en el depósito; de tal forma que todos los clientes sean visitados una vez y la suma de las distancias que los vendedores deben recorrer sea mínima.

El diseño del algoritmo para resolver el MTSP está formado por 2 fases: *i)* obtener k particiones del conjunto de los clientes V; y, *ii)* para cada partición obtenida en la fase 1, construir un recorrido.

La fase 1 se considera la más importante, ya que la calidad de las soluciones depende mucho de las particiones que esta genera. En ella se observa que la distribución de los vértices en las diferentes particiones es mejor, si el depósito se encuentra en el centro con respecto a los demás vértices.

La fase 2 obtiene recorridos con un margen de error entre 6 % y 22 %, sobre los mejores resultados conocidos reportados en la biblioteca TSPLIB para el problema TSP, con una complejidad temporal $O(n^3)$.

Los resultados para 22 instancias disponibles en la biblioteca TSPLIB, son los siguientes: *i)* El 45 % de los costos de las soluciones, es menor que los mejores costos conocidos para el MTSP Bounded ; y, *ii)* El 59.09 % de los costos de las soluciones obtenidas por el algoritmo 2 fases, tiene una diferencia menor al 5 % con respecto a los mejores costos conocidos, reportados en la literatura para el MTSP Bounded.

Abstract

The Multiple Traveling Salesman Problem, MTSP, consists in construct or finding a sequence of clients to visit for each salesman, starting and ending all of them in the depot; so that all clients are visited once and the sum of the distances that salesman must travel be the minimum.

The design of the algorithm to solve the MTSP consists of 2 phases: i) obtain k partitions of the set of clients V; and, ii) for each partition obtained in phase 1, construct a tour.

Phase 1 is considered the most important since the quality of the solutions depends a lot on the partitions it generates. It shows that the distribution of the vertices in the different partitions is better if the depot is in the center with respect to the other vertices.

Phase 2 obtains tours with a margin of error between 6 % and 22 %, on the best-known solutions reported in the TSPLIB library for the TSP problems, with time complexity $O(n^3)$.

The results for 22 instances available in the TSPLIB library are the following: *i)* 45 % of the costs of the solutions is less than the cost of the best-known solutions for the MTSP Bounded; and, *ii)* 59.09 % of the costs of the solutions obtained by the 2-phase algorithm, have a difference of less than 5 % compared to the cost of the best-known solutions, reported in the literature for the MTSP Bounded.

Agradecimientos

A mis padres: María Eulalia y Víctor Bonifacio Martín; mi esposa Elvira y mi hijo Adrián.

A la Universidad Autónoma del Estado de Morelos (UAEM), que a través de la Facultad de Contaduría, Administración e Informática (FCAeI), el Centro de Investigación en Ciencias (CInC) y el Centro de Investigación en Ingeniería y Ciencias Aplicadas (CIICAP); que me facilitaron los medios e instalaciones.

Al Conacyt, que me ha brindado la confianza y el apoyo económico.

A la Universidad Nacional Autónoma de México (UNAM); que en tres ocasiones nos proporcionó sus instalaciones: el Instituto de Matemáticas y el Instituto de BioTecnología), equipamiento y profesores: Dr. David Romero y la Dra. Blanca Taboada para tomar materias en esa institución.

A mi asesor de tesis, el Dr. Nodari Vakhania, quién me enseñó las técnicas de aprendizaje que se requieren para el desarrollo de algoritmos heurísticos; así como la asesoría para el desarrollo de este proyecto.

A mi Comité tutoral, el Dr. Federico Alonso Pesina, el Dr. Crispín Zavala, el Dr. Martín Martínez Rangel y el Dr. José Alberto Hernández; quienes dieron seguimiento y asesoría a mis avances durante todo el tiempo en el que este trabajo fue realizado.

A la Dra. Lorena Díaz González, el Dr. Crispín Zavala, el Dr. Luis Gaggero, el Dr. Outmane Oubram, el Dr. Noureddine Lakouari y el Dr. José Alberto Hernández; quienes me han motivado a continuar con optimismo y entusiasmo este proyecto.

A mi amigo José Luis, la Dra. Elisa Chinos, Alejandro Reynoso, el Dr. José Alberto Hernández y la Biól. Ivonne Miranda, quienes me apoyaron y asesoraron para la escritura de mi tesis.

A mis compañeros de la Maestría en Optimización y Cómputo Aplicado (MOCA): Alhelí, Ana, Berenice, Jaqueline, Javier, Mohammed, Rocío, Siquem y Uriel.

A mis compañeros de clases en el CInC: Alejandro, Andrés, David, Edmundo, Elisa, Eliseo, Francisco, Fredy, Gabriel, José, Rey David y Valeria.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

En esta investigación, se presenta el diseño de un algoritmo heurístico en 2 fases para resolver instancias del Problema de los Vendedores Viajeros [1], ***Multiple Traveling Salesman Problem (MTSP)***, en donde las distancias entre las ciudades son euclidianas en un plano 2D.

La tesis está organizada del siguiente modo:

En el capítulo 1, se describen el planteamiento del problema, la justificación y los objetivos que serán cubiertos en este proyecto de investigación.

En el capítulo 2, se describen aspectos importantes encontrados en la literatura acerca del MTSP; tal como la descripción, las variantes, la relación que tiene con otros problemas y la clasificación de los algoritmos que se han utilizado para resolverlo. También se incluye una descripción detallada del algoritmo en 2 fases de [Vakhania *et al.* 2016], debido a que se usó como base para la elaboración de este diseño.

En el capítulo 3, se describe el planteamiento matemático, se definen los conceptos y variables; y se describen los algoritmos de forma verbal y detallada en pséudocódigo para el diseño de esta heurística.

En el capítulo 4, se comparan los resultados del algoritmo diseñado en este proyecto de investigación con los encontrados en la literatura.

En las conclusiones se emiten los comentarios y observaciones que resultaron de las comparaciones realizadas en el capítulo 4.

[1] ***Multiple Traveling Salesman Problem*** se ha traducido al español como el Problema de los Vendedores Viajeros, debido a que la palabra ***multiple*** en inglés denota que son muchos

1.1. Planteamiento del problema

Existen muchos problemas de optimización que requieren ser resueltos. La gran mayoría con implicaciones prácticas en ciencias, ingeniería, economía y negocios; que son muy complejos y difíciles de resolver [Cuevas *et al.*, 2016].

En la vida real, la optimización se presenta en: *i)* la minimización del tiempo invertido para la ejecución de una tarea, el costo de un producto y el riesgo en una inversión o, *ii)* la maximización de las ganancias, la calidad de un producto y la eficiencia de un dispositivo [Cuevas *et al.*, 2016].

El Problema del Vendedor Viajero, ***Traveling Salesman Problem (TSP)***, es uno de los problemas típicos en optimización combinatoria y es conocido por ser de la clase NP-duro; probado por Karp en 1972 [Jünger *et al.* 1995]. Consiste en un vendedor que debe visitar a todos sus clientes una vez, y regresar con el cliente en donde inició su recorrido, de tal forma que la distancia total sea mínima.

El MTSP, es importante desde el punto de vista teórico y práctico, ya que es una variante del TSP [Bektas 2006]; que incluye más de un vendedor y un vértice especial denominado *depósito*. Consiste en construir o encontrar una secuencia de clientes por visitar para cada vendedor, partiendo y concluyendo todos ellos su recorrido en el depósito; de tal forma que todos los clientes sean visitados una vez y la suma de las distancias que los vendedores deben recorrer sea mínima (figura 1.1). Es conocido por ser un problema NP-Duro [Yousefikhoshbakht y Sedighpour 2012] y su importancia consiste en el hecho de que es un problema de optimización combinatoria [Harrath *et al.* 2019]. Es usado para resolver problemas en las que exista la necesidad de reducir costos, tiempos o distancias; mediante la planeación y calendarización de procesos. Por ello, es que se requiere del desarrollo continuo de algoritmos cuyas capacidades permitan encontrar mejores soluciones en tiempos razonables.

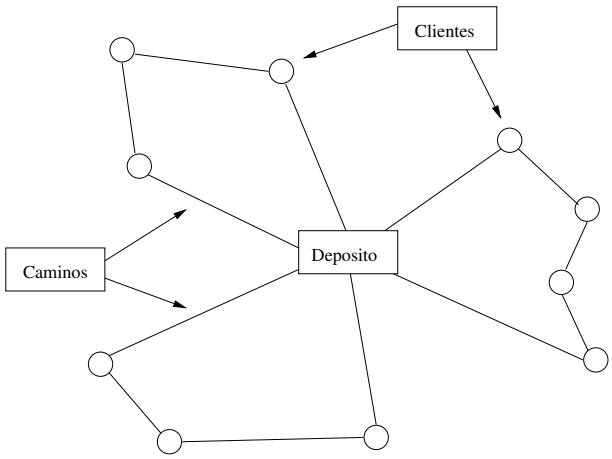

Figura 1.1: Ejemplo de una solución para un MTSP con 10 clientes, 1 depósito y 3 vendedores [Yousefikhoshbakht y Sedighpour 2012].

Algunas de las aplicaciones encontradas en la literatura, las cuales han sido resueltas por un MTSP, son las siguientes:

- **Calendarización de impresiones en una imprenta (*Print press Scheduling*)**

  Una *edición* es un conjunto de *páginas* de *anuncios publicitarios*, y las ediciones pueden tener páginas en común. Si dos ediciones son idénticas, entonces pueden ser combinadas y consideradas como una sola edición.

  Una prensa con 5 pares de *rodillos* es usada para imprimir diferentes ediciones de una revista, cuyos rodillos están divididos en 3 *líneas*, en las cuales se coloca una *placa*, que contiene un *formato* de 4, 6 u 8 páginas que incluyen anuncios publicitarios.

  Para imprimir un conjunto de páginas se requieren 1, 2 ó 4 placas; dependiendo si son blanco y negro (BW), 2 colores (2C) ó 4 colores (4C) respectivamente. El proceso consiste en pasar el papel blanco entre los 5 pares de rodillos, en donde cada par de ellos imprime un color en ambos lados del papel simultáneamente.

  El objetivo del problema, consiste en calendarizar las impresiones de los formatos según la cantidad de páginas requeridas de cada edición, tal que el costo por el número de cambios de placas, las veces que la prensa se detiene, las veces que se duplica una placa, la cantidad excedida de impresiones de una página y el cambio de color para el par de rodillos que imprime el segundo color en las impresiones (2C); sea mínimo.

  La solución consiste en resolver un MTSP con 3 agentes viajeros solamente, en donde cada agente representa a una línea de los tres que tiene cada rodillo; los vértices representan a las placas que serán usadas para imprimir las páginas en una ejecución y la distancia entre los vértices representa a la suma de los costos ya mencionados.

  Como observación tenemos que el costo entre dos ejecuciones son el cambio de placas de los rodillos, las veces que la prensa se detiene y el cambio de color en el caso de que las páginas sean impresas en dos colores. Para incluir los costos por duplicar una placa y el exceso de impresiones de una página, estos deben exceder la mitad del costo por detener la prensa, de lo contrario no se incluyen en el costo.

  Este problema fue una de las primeras aplicaciones de un MTSP presentado por [Gorenstein 1970] [Bektas 2006].

- **Calendarización de personal (*Crew scheduling*)**

  El problema consiste en un conjunto de k personas que deben salir de una oficina central, visitar n diferentes lugares una vez para desempeñar alguna actividad y después, regresar a la oficina central. El objetivo es determinar el recorrido para cada persona, tal que la suma de las distancias de los recorridos sea mínima.

  Como ejemplos tenemos: *i)* Las personas son mensajeros que deben recoger depósitos en diferentes sucursales bancarias [Svestka y Huckfeldt 1973]; *ii)* Las personas son técnicos que deben visitar casetas telefónicas una o dos veces a la semana para vaciar las monedas de los equipos telefónicos, y en caso de ser necesario, realizar mínimas reparaciones a los equipos [Lenstra y Kan 1975]; y, *iii)* Las personas son carteros que deben visitar buzones de correo para vaciarlos diariamente [Lenstra y Kan 1975].

Como observación, se tiene que en el planteamiento de cada problema de los 3 ejemplos del parrafo anterior, no se menciona que existe una capacidad de carga del vehículo que usa cada persona, ni tampoco la cantidad de carga que recoge en cada lugar que visita. Por lo tanto, es claro que se tiene un MTSP.

- **Calendarización de entrevistas** (***Interview scheduling***)

  Se tienen como objetos: n oficinas de vendedores de servicios en la industria turística y k ejecutivos de compras. Se define T como el número de periodos de tiempo para realizar las entrevistas entre los vendedores y los ejecutivos de compra, donde $T \leq k \leq n$. Por lo anterior, un vendedor puede entrevistarse con T de los n vendedores de servicios disponibles.

  Inicialmente, cada ejecutivo de compra y vendedor de servicio exponen en una lista una clasificación para expresar las preferencias para sus entrevistas; esa clasificación es usada para establecer el interés que hay entre ellos para una futura reunión.

  El objetivo, es generar el calendario para que cada ejecutivo de compra visite a los T vendedores de servicios, tal que un ejecutivo de compra solamente puede reunirse con un vendedor de servicios en un periodo de tiempo.

  Para resolver este problema a través de un MTSP, cada ejecutivo de compra es un vendedor viajero y los vendedores de servicios son las diferentes ciudades por visitar. Los costos de las aristas se calculan con las preferencias dadas por los ejecutivos de compra y los vendedores de servicios. La solución está compuesta por los k recorridos tal que la suma de sus costos sea mínima.

  Este problema fue descrito por [Gilbert y Hofstra 1992], y como observación se tiene que pudiera haber vendedores de servicios que no sean entrevistados y también, otros vendedores que sean entrevistados por más de un ejecutivo de ventas; por lo tanto es una variante de un MTSP.

- **Planificación de misiones** (***Mision Planning***)

  El problema presentado por [Brumitt y Stentz 1996], consiste en que n diferentes localidades sean visitadas por cualquiera de los k robots iguales, que hayan sido destinados dentro de una misión; tal que se eviten colisiones entre ellos, se maximice la seguridad de ellos y principalmente, se minimice la distancia de los recorridos. Una vez que se han alcanzado todas las localidades, los robots deben regresar a un sitio en donde se encuentra la base para su recarga de energía.

  Para que la misión sea completada, adicionalmente se incluyó dentro del diseño del algoritmo un sistema de planeación dinámica, que consiste en ir modificando en cualquier momento los recorridos para cada robot durante la misión; con base en algunos factores como lo son: la obtención de nueva información, las condiciones técnicas de los robots y las condiciones cambiantes del medio ambiente.

- **Drones para entregas** (***Drone delivery***)

Este problema puede ser modelado como un MTSP en 3 dimensiones [Lo *et al.* 2018]

En el comercio electrónico y compañías de ventas, se busca las formas para mejorar los costos y tiempos de entregas, mediante el uso de drones [Kitjacharoenchai *et al.* 2019].

La estrategia consiste en combinar las entregas mediante camiones y drones, quienes parten del depósito, realizan sus entregas y regresan nuevamente al depósito.

- **Diseño de redes de observación por medio de un sistema satelital de navegación global (*Design of global navigation satellite system surveying networks (NGSS)*)**

  Para este problema, investigado por [Saleh y Chelouah 2004], tenemos n *estaciones* terrestres, k *receptores* y l *sesiones de observación*; donde: *i)* una estación es un lugar físico en tierra con coordenadas (x, y, z) en donde se desea tener una medición satelital; *ii)* un receptor es el equipo que recibe las señales satelitales para obtener y guardar las mediciones; *iii)* una sesión de observación es un período de tiempo durante el cual, dos o más receptores guardan señales satelitales durante un tiempo fijo; y, *iv)* una *red de observación* es una lista, que determina en que estaciones debe estar un receptor para realizar una sesión de observación.

  El objetivo de este problema, es determinar el orden en que deben realizarse las sesiones especificadas en la red de observación, tal que el costo de las distancias recorridas por los receptores para visitar las estaciones sea mínima.

1.2. Justificación

Este proyecto de investigación, busca ofrecer un algoritmo para la solución del MTSP, el cuál se pretende que encuentre buenas soluciones factibles, y cuyos resultados puedan obtenerse en un tiempo razonablemente corto.

Debido a que la solución de este problema minimiza la distancia recorrida por los vendedores, también existe una disminución en el consumo de combustible y el tiempo requerido para realizar los recorridos; lo que trae como consecuencia beneficios como: ganancias para las empresas, y mayor satisfacción para los clientes y/o consumidores de sus productos y servicios. Por otro lado, también existen beneficios como la disminución en la emisión de contaminación y desgaste de las máquinas y/o vehículos involucrados.

La solución para este tipo de problemas, continúa siendo estudiado por los investigadores en el área de la optimización, dada la necesidad para encontrar mejores soluciones en menores tiempos de procesamiento.

1.3. Objetivo general

Diseñar e implementar un algoritmo heurístico en dos fases, que genere y evalúe soluciones factibles para un MTSP en un tiempo razonable; de tal forma que esas soluciones tengan un costo mínimo.

1.4. Objetivos específicos

- Realizar una revisión en la literatura para conocer cuales son las principales variantes y metodologías con las cuales se resuelve un MTSP.
- Diseñar una heurística en dos fases, que construya una solución factible que resuelva instancias para un MTSP, cuya suma de las distancias de los recorridos de la solución sea mínima.
- Implementar la heurística diseñada en C++, que construya una solución factible que resuelva instancias para un MTSP, cuya suma de las distancias de los recorridos de la solución sea mínima.
- Realizar un comparativo del desempeño del algoritmo propuesto y los resultados mejores conocidos reportados en la literatura.

1.5. Hipótesis

H1: Si un algoritmo heurístico en dos fases, es una buena alternativa que genera y evalua soluciones factibles para un MTSP; entonces el algoritmo entrega como resultado una solución tal que su costo sea mínimo en un tiempo razonable.

H0: Si un algoritmo heurístico en dos fases, no es una buena alternativa que genera y evalua soluciones factibles para un MTSP; entonces el algoritmo no entrega como resultado una solución tal que su costo sea mínimo en un tiempo razonable.

1.6. Alcances

El algoritmo en dos fases implementado en este proyecto de investigación, será evaluado solo con 22 instancias de la biblioteca [TSPLIB], mencionadas en el capítulo 2.6.

1.7. Limitaciones

En el diseño del algoritmo en dos fases de este proyecto de investigación, solo serán utilizados como datos de entrada: el conjunto de los vértices y la cantidad de recorridos por construir; por lo anterior el límite inferior m_{min} y límite superior m_{max}, que son parte de una restricción para un problema MTSP Bounded, no serán considerados.

Capítulo 2

Marco teórico

2.1. Descripción del problema

Sea $G = (V, E)$ un grafo no dirigido ponderado, donde V es el conjunto de los vértices y E el conjunto de las aristas; y k el número de vendedores que definen un MTSP.

2.1.1. Objetos

k, es el número de vendedores.

$d \in V$ es el depósito, del cual cada vendedor inicia y concluye su recorrido.

$i \in V \setminus \{d\}$, $1 \leq i \leq n$; son los vértices que deben ser visitados por uno de los vendedores una sola vez.

$(i, j) \in E$; $i \neq j$; $i, j \in V$; son las aristas.

2.1.2. Parámetros

$w(i, j) \in E$, $w(i, j) > 0$; es el costo (distancia, tiempo, entre otros) de la arista (i, j).

La matriz W, es simétrica si $w(i, j) = w(j, i)$; de lo contrario es asimétrica.

2.1.3. Función objetivo

Se define una matriz binaria X, donde $x_{ij} = 1$ si la arista (i, j) está incluida en la solución; y, $x_{ij} = 0$ si la arista (i, j) no lo está.

La función objetivo, es la minimización de la suma de los pesos de todas las aristas incluidas en la solución (ecuación 2.1), sujeto a las restricciones que se muestran a continuación.

$$\text{mín} \sum_{(i,j)\in E} w(i,j)x_{ij} \tag{2.1}$$

2.1.4. Restricciones

- Todos los vendedores inician su recorrido en el depósito.

$$\sum_{j\in V\setminus\{d\}} x_{dj} = k$$

- Todos los vendedores regresan al depósito al concluir su recorrido.

$$\sum_{j\in V\setminus\{d\}} x_{jd} = k$$

- Para todo vértice j, existe solo un vértice i que está ubicado atrás del vértice j dentro de un recorrido; i, j pertenecen al conjunto de los vértices con excepción del depósito.

$$\sum_{i\in V\setminus\{d\}} x_{ij} = 1;\ \forall j \in V \setminus \{d\}$$

- Para todo vértice i, existe solo un vértice j que está ubicado adelante del vértice i dentro de un recorrido; i, j pertenecen al conjunto de los vértices con excepción del depósito.

$$\sum_{j\in V\setminus\{d\}} x_{ij} = 1;\ \forall i \in V \setminus \{d\}$$

- Restricciones para la eliminación de subrecorridos. Son usados para evitar que existan recorridos que no inician y terminan en el depósito.

2.1.5. Solución factible

Una solución factible, es un conjunto de k recorridos que inician y concluyen en el depósito d, de tal forma que sean visitados los vértices que pertenecen al conjunto de los vértices, con excepción del depósito, una sola vez.

Por lo anterior, un recorrido para cada vendedor que debe visitar m vértices, está formado del siguiente modo: la 1^{er} arista dentro de su recorrido para ir del depósito d al vértice i_1, es (d, i_1); la 2^a arista dentro de su recorrido para ir del vértice i_1 al vértice i_2, es (i_1, i_2); y así sucesivamente hasta llegar a la arista (i_{m-1}, i_m), para ir del vértice i_{m-1} al vértice i_m; y, para concluir el recorrido, la última arista es (i_m, d), para ir del vértice i_m al depósito d.

$$(d, i_1), (i_1, i_2), ..., (i_{m-1}, i_m), (i_m, d)$$

donde el costo $c(t)$, para este recorrido es:

$$c(t) = w(d, i_1) + w(i_1, i_2) + ... + w(i_{m-1}, i_m) + w(i_m, d)$$

La minimización de la suma de los costos para los k recorridos, t_1, t_2, ..., t_k, es igual a la dada en la ecuación 2.1.

$$\text{mín}\{c(t_1) + c(t_2) + ... + c(t_k)\} = \text{mín} \sum_{(i,j)\in E} w(i, j)x_{ij}$$

2.2. Variantes del problema

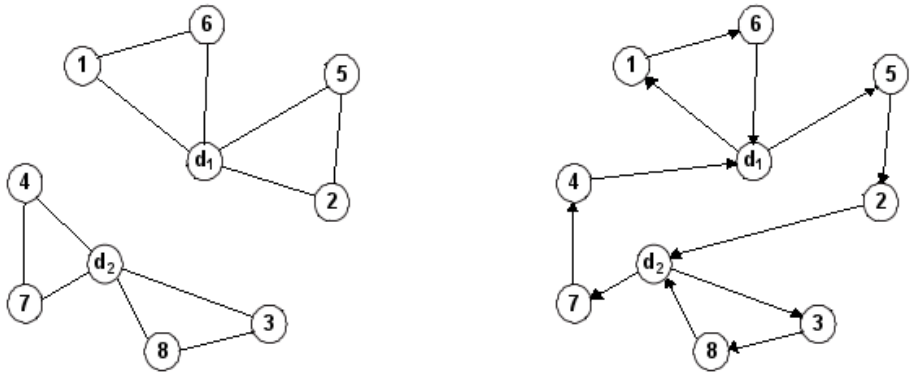

Figura 2.1: *izquierda)* MTSP con múltiples depósitos con destino fijo, *derecha)* MTSP con múltiples depósitos con destino no fijo.

Múltiples depósitos: En esta variante, se tiene más de un depósito en los que inicialmente hay un número de vendedores. Se le conoce como *destino fijo*, cuando todos los vendedores deben regresar al depósito del cual partieron al finalizar sus recorridos; y, *destino no fijo*, cuando los vendedores pueden llegar a cualquier depósito al finalizar sus recorridos, con la condición de que los depósitos tengan el mismo número de vendedores que inicialmente tenían [Bektas 2006].

En la figura 2.1, se muestran dos soluciones que corresponden a un problema MTSP con mútiples depósitos: *izquierda*) de destino fijo; y, *derecha*) de destino no fijo.

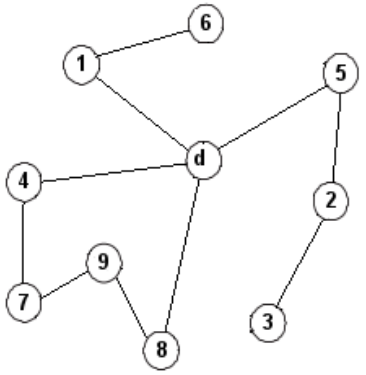

Figura 2.2: MTSP con dos recorridos abiertos y uno cerrado.

Recorridos abiertos y/o cerrados: En esta variante, se definen 2 tipos de vendedores: *i)* internos; y, *ii)* externos. Todos los vendedores inician su recorrido en el depósito, pero solo los internos deben regresar al depósito después de visitar todos sus vértices (***closed paths***), mientras que los externos no regresan (***open path***) [Wang *et al.* 2013] [Thenepalle y Singamsetty 2019].

En la figura 2.2, se muestra una solución para un MTSP con dos recorridos abiertos y uno cerrado.

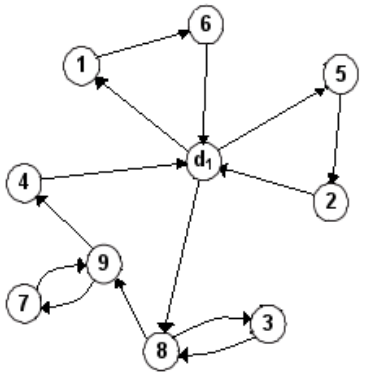

Figura 2.3: MTSP con dos vértices visitados más de una vez en un recorrido.

Vértices incluídos más de una vez en un recorrido: En esta variante no existe la restricción de visitar una sola vez a cada vértice, esto es conveniente en problemas en donde la matriz de costos es asimétrica, y también podrían encontrarse recorridos con menor costo.

En la figura 2.3, se muestra una solución para un MTSP, en la que el vértice 9 es visitado dos veces en uno de los tres recorridos.

Ventanas de tiempo: Esta variante es conocido como MTSPTW (***Multiple Travel Salesman Problem Time Windows***). En este problema, debe considerarse que existe un intervalo de tiempo en el cual, el vendedor puede visitar a un vértice dentro de su recorrido [Bektas 2006].

Intervalo de vértices por visitar en los recorridos: En esta variante, el número de vértices que pueden agregarse a cada recorrido, está dentro del intervalo: *Límite Inferior* (***Lower Bound***) y *Límite Superior* (***Upper Bound***) [Bektas 2006].

Intervalo de distancia mínima y máxima en los recorridos: Análogamente a la variante anterior, en este problema se limita a cada recorrido la distancia mínima y máxima que un vendedor puede recorrer [Bektas 2006].

Número de vendedores: El número de vendedores es fijo o acotado [Bektas 2006].

Costos ajustados: Si el número de vendedores en el problema es variable, usualmente cada vendedor tiene un costo fijo; ese costo fijo es agregado al costo total por cada vendedor que es requerido en la solución [Bektas 2006].

2.3. Relación del MTSP con otros problemas

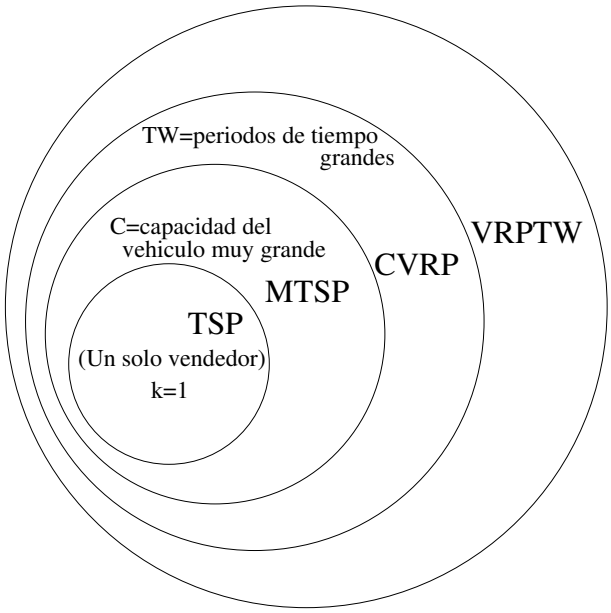

Figura 2.4: Relación del MTSP con otros problemas

Problema del Vendedor Viajero (***Traveling Salesman Problem (TSP)***)

Consiste en obtener el recorrido con el camino más corto para un vendedor que debe visitar cada una de los n vértices sucesivamente, empezando y terminando su recorrido en el vértice 1 [Gilbert y Hofstra 1992].

Un TSP puede considerarse un caso particular de un MTSP, cuando el número de vendedores es igual a uno ($k = 1$).

Así mismo, un problema MTSP puede ser transformado a un problema TSP, descomponiendo al depósito d en k vértices $d1, d2, ..., dk$. Las aristas (i, dj) y (dj, i) tienen un peso $w(i, dj) = w(i, d)$ y $w(dj, i) = w(d, i)$ respectivamente, y $w(dj, dl) = M$, donde M es un valor demasiado grande; $1 \leq i \leq n;\ 2 \leq j, l \leq k$ [Jünger *et al.* 1995].

Problema de Enrutamiento de Vehículos con Capacidad (***Capacitated Vehicle Routing Problem (CVRP)***)

Este problema se define de manera similar que un MTSP, agregando a los n vértices una demanda q_i, que debe ser cubierta por uno de los k vendedores, cuyo vehículo que utiliza tiene una capacidad C, debiendo empezar y terminar su recorrido en el depósito d, donde los vértices deben ser visitados una sola vez y $C \ll \sum_{i=1}^{n} q_i$ [Dantzig y Ramser 1929]. En este problema, adicionalmente a las restricciones que se tienen en un MTSP, se tiene que en cada recorrido, la suma de las demandas de vértices incluidos no exceda la capacidad del vehículo [Jünger *et al.* 1995][Pecin *et al.* 2017].

El objetivo de este problema, al igual que en un MTSP, es obtener un conjunto de k recorridos cuyo costo (distancias, tiempo, entre otros) sea mínimo.

Un MTSP puede considerarse entonces como un caso partícular de un CVRP cuando $C \geq \sum_{i=1}^{n} q_i$ y un CVRP puede ser considerado como un caso partícular de un VRPTW (definido a continuación) donde la ventana de tiempo es arbitrariamente grande [Pecin *et al.* 2017].

Problema de Enrutamiento de Vehículos con Ventanas de Tiempo (***Vehicle Routing Problem with Time Windows (VRPTW)***)

El problema se define de manera similar al CVRP, agregando una ventana de tiempo como un parámetro para los n vértices, donde $[a_i, b_i]$, es el período de tiempo en el cual se puede visitar al vértice i; $a_i < b_i$; y también se agrega una restricción, la cuál consiste en que un vendedor solo debe visitar al vértice i dentro de la ventana de tiempo $[a_i, b_i]$.

Por lo anterior, un CVRP es un caso partícular de un VRPTW, en donde las ventanas de tiempo de los n vértices es muy grande.

2.4. Clasificación de los algoritmos que solucionan el MTSP

2.4.1. Algoritmos Exactos

Son aquellos que son diseñados de tal manera que, es garantizado que encontrarán la solución óptima en un tiempo finito; y por ello deben probar que el algoritmo encuentra la mejor solución. [Sörensen 2013]. Estos algoritmos, examinan cada solución factible dentro del espacio de soluciones para el problema que resuelven; guardando en cada iteración la solución en la que la función objetivo de como resultado un menor o mayor valor, para problemas de minimización o maximización respectivamente.

La principal característica de estos algoritmos es que solo pueden resolver instancias relativamente pequeñas y el tiempo para encontrar la solución es demasiado grande [Cordeau *et al.* 2005]; tal es el caso de *ramas y cortes (**branch and bound**)*, que para un MTSP, el tiempo de procesamiento se incrementa de manera exponencial conforme aumenta el número de vértices en la instancia [Svestka y Huckfeldt 1973].

Finalmente, cabe señalar que el desarrollo de estos algoritmos no es trivial, incluso para aquellos que pueden considerarse como fáciles [Sörensen 2013].

2.4.2. Algoritmos Heurísticos

Debido a la imposibilidad de encontrar soluciones exactas para muchos problemas, sea por la complejidad del problema o del tamaño de la instancia; es que los investigadores a menudo se enfocaron en la intuición, comprensión y aprendizaje humano como una forma para resolver problemas [Simon y Newell 1958] [Sörensen 2013] a través de los algoritmos heurísticos.

Estos algoritmos ofrecen soluciones factibles no óptimas en un tiempo razonablemente corto comparado con los algoritmos exactos; pero a pesar de eso, su aceptación en el campo de la investigación ha sido lenta, debido a que ha sido mal visto por la comunidad académica [Sörensen 2013].

Dentro de esta categoría, se tienen las siguientes:

Algoritmos voraces: pueden ser aplicados a una gran variedad de problemas, consiste en construir y evaluar una solución factible. El algoritmo inicia con una solución sin elementos, y en cada iteración subsecuente agrega un elemento a la solucion, con base en una decisión que será definitiva.

Algoritmos de aproximación: garantizan que la solución que construyen se encuentra dentro de un cierto porcentaje de la solución óptima [Sörensen 2013].

Transformaciones: Consiste en aplicar algunas consideraciones al problema original para que sea cambiado a otro problema. Particularmente para un MTSP, como se explicó en la sección anterior, puede ser transformado a un TSP. Esto es útil porque al resolver el problema transformado mediante cualquier algoritmo, se resuelve también el problema que originalmente se tiene interés por resolver.

2.4.3. Algoritmos Metaheurísticos

Una metaheurística, es un conjunto de pautas o estrategias generales independientes del problema, para el desarrollo de algoritmos utilizados en la búsqueda de la solución de una amplia variedad de problemas de optimización [Sörensen 2013]; en donde la mayoría de ellos usan como inspiración fenómenos biológicos o sociales [Cuevas *et al.*, 2016].

En términos generales, estos algoritmos son una implementación que inicialmente generan una solución factible S de manera aleatoria y posteriormente de manera iterativa: *i)* genera una solución S' mediante una transformación de S, tal que satisfaga las restricciones del problema; *ii)* si la evaluación de la solución S' mejora a la evaluación de S con base en la función objetivo, entonces S' reemplaza a S, de lo contrario, se establece algún criterio de aceptación de la solución S' para salir de aquellos mínimos locales que el algoritmo haya encontrado; y *iii)* el algoritmo concluye hasta

que se cumpla con algún criterio de paro [Lin y Kernighan 1973], que generalmente es sintonizado en el algoritmo.

Por lo anterior, estos algoritmos resuelven problemas de optimización complejos, mediante una forma muy eficiente de generar soluciones factibles; y, en caso de ser alcanzado un mínimo local, busca una solución factible alternativa que ayude a encontrar mejores soluciones. Por ello, es que las metaheurísticas se han popularizado tanto en la industria como en la academia, ya que han demostrado ser herramientas eficientes para la solución de un amplio rango de problemas en dominios tales como la logistica, bio-informática, diseño estructural, minería de datos, finanzas, entre otros [Cuevas *et al.*, 2016].

2.5. Algoritmo en 2 fases para un MTSP

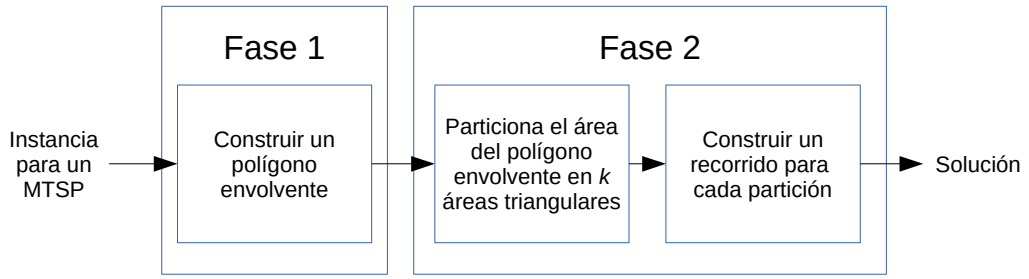

Figura 2.5: Diagrama que muestra las fases del algoritmo en 2 fases

El desarrollo del diseño del algoritmo y la programación en dos fases diseñados en esta investigación, está basado en la publicación de [Vakhania *et al.* 2016], que pretende encontrar buenas soluciones factibles no óptimas para un MTSP, y cuyos resultados puedan obtenerse en un tiempo razonablemente corto. El algoritmo ahí descrito muestra para cada etapa, una interesante forma para resolver el problema con base en la manera en que los vendedores pueden desplazarse, iniciando sus recorridos en el depósito, visitando los vértices que deben visitar y finalmente regresar al depósito.

Para la etapa de agrupamiento (fase 1), el algoritmo descrito considera que, al partir todos los vendedores del depósito, estos debieran desplazarse hacia diferentes direcciones en forma radial, tomando como centro al depósito; por lo que aquellos vértices que se encuentren cercanos a cada dirección, serán parte de un agrupamiento.

Para la etapa de generación de recorridos (fase 2), el algoritmo construye un recorrido para cada agrupamiento realizado en la fase 1; considerando dos direcciones a partir del depósito: *i)* del depósito hacia el vértice más alejado del depósito dentro del agrupamiento; y, *ii)* del vértice más alejado del depósito dentro del agrupamiento, alcanzado en la anterior dirección, hacia al depósito.

En la figura 2.5 se muestran las fases de este algoritmo: *i)* Construir un polígono envolvente; y *ii)* Particiona el polígono envolvente en k áreas triangulares para construir un recorrido por cada partición.

Sea un Grafo completo no dirigido ponderado $G = (V, E)$ cuyas aristas $(i, j) \in E$ tienen un peso, $w(i, j)$, donde el vértice i conecta al vértice j. Un vértice distinguido d (llamado depósito) y un número entero positivo k.

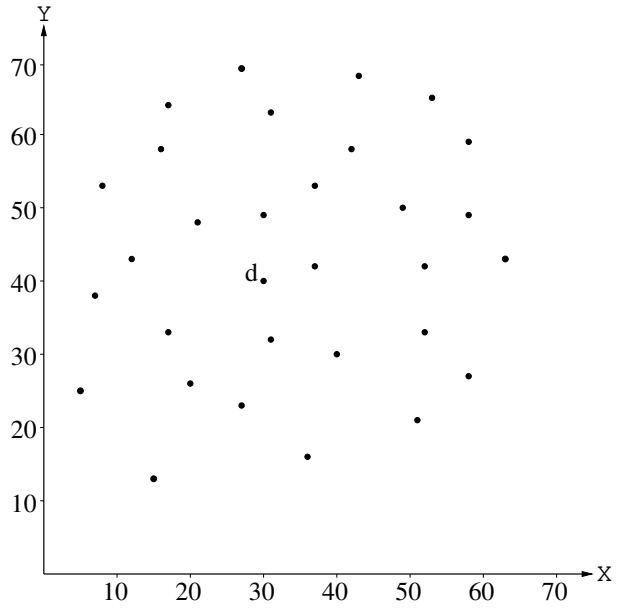

Figura 2.6: Conjunto de vértices que pertenecen al grafo $G = (V, E)$, que será usado de ejemplo en esta sección. (La instancia está disponible en el Apéndice A)

Se define como un recorrido, T_Y, a un ciclo que comienza en el vértice d, visita todos los vértices de la partición $Y \subseteq V$ exactamente una vez y regresa al vértice d (Ecuación 2.2). El costo de ese recorrido, $c(T_Y)$, es la suma de los pesos de las aristas incluidas en el recorrido (Ecuación 2.3).

$$T_Y = (i_1, i_2, ..., i_l, i_1);\ i_1 = d,\ i_2, ..., i_l \in Y \setminus \{d\} \tag{2.2}$$

$$c(T_Y) = w(i_1, i_2) + w(i_2, i_3) + ... + w(i_l, i_1) \tag{2.3}$$

El objetivo del algoritmo, es encontrar la partición del conjunto $V \setminus \{d\}$ en k subconjuntos $V_1, \cdots, V_k$; tal que la suma de los costos de los recorridos, $\sum_{i=1,\cdots,k} c(V_i)$, sea mínima (Ecuación 2.4).

$$\text{mín} \sum_{i=1}^{k} c(V_i) \tag{2.4}$$

La fase 1 consiste en construir un polígono convexo, $P = P(V)$, dentro del espacio euclidiano 2D que encierre a todos los vértices de V (figura 2.7).

El procedimiento inicialmente obtiene los vértices extremos: v_1 (con coordenada y máxima), v_l (con coordenada x mínima), v_o (con coordenada y mínima) y v_r (con coordenada x máxima).

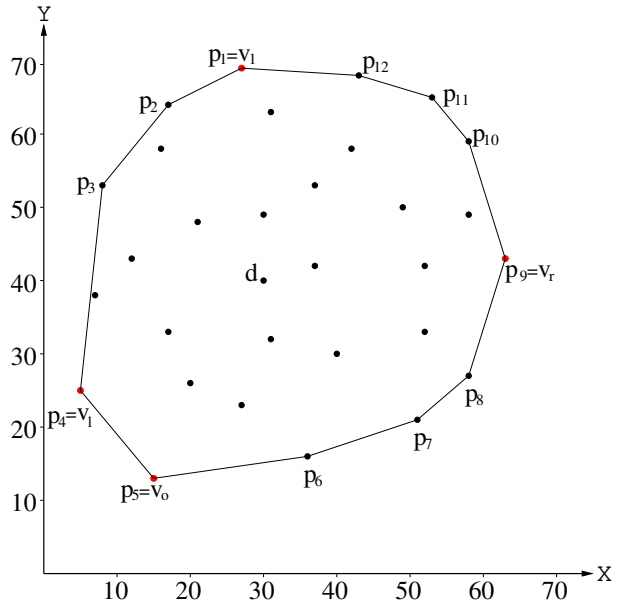

Figura 2.7: Polígono envolvente

Posteriormente, para la construcción del polígono se realizan 4 etapas: *i)* busca los vértices que se encuentran entre v_1 y v_l; *ii)* busca los vértices que se encuentran entre v_l y v_o; *iii)* busca los vértices que se encuentran entre v_o y v_r; y, *iv)* busca los vértices que se encuentran v_r y v_1. (Definición y procedimiento descritos en el capítulo 3, en la definición 3.2.14 y en el procedimiento 3.3.4 respectivamente).

En el ejemplo de la figura 2.7, podemos observar el resultado del algoritmo *construcción_del_polígono* en donde $p_1 = v_1$. Posteriormente, los vértices p_2, p_3 y p_4 son encontrados en la *etapa 1.* El vértice p_5 es encontrado en la *etapa 2.* Los vértices p_6, p_7, p_8 y p_9 son encontrados en la *etapa 3.* Y por último, los vértices p_{10}, p_{11} y p_{12} son encontrados en la *etapa 4.*

La fase 2 está compuesta por 2 etapas: *i)* particiona el área del polígono en k áreas triangulares; y, *ii)* se construye un recorrido para cada una de esas áreas.

Etapa 1: Sean p_x un vértice de P y m el número de vértices que tiene el polígono envolvente; definimos como una *arista auxiliar* a la arista (d, p_x).

Para particionar el área del polígono envolvente, inicialmente se generan las aristas $(d, p_x), 1 \leq x \leq m$; lo que da como resultado m aristas auxiliares y m áreas triángulares.

En el ejemplo de la figura 2.8, podemos observar que el polígono de la figura 2.7 está particionado por 12 aristas auxiliares (d, p_x), con los cuales se forman 12 áreas triángulares.

Para una arista auxiliar e_x, se define un conjunto $B(e_x)$, cuyos elementos son aquellos que están a una distancia menor o igual a b_x (un valor aleatorio que debe estar dentro de un intervalo, mismo que debe ser sintonizado para el problema) de la arista e_x.

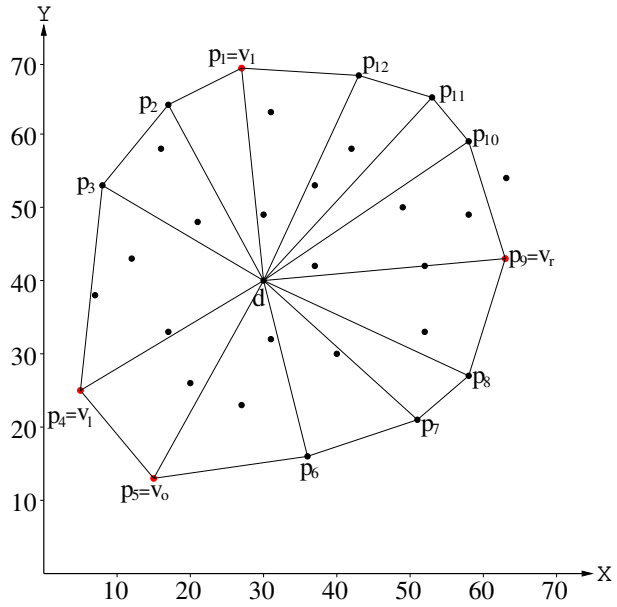

Figura 2.8: Polígono envolvente particionado por m aristas auxiliares

$$B(e_x) = \{v_i | v_i \textit{ está a una distancia menor o igual a } b_x \textit{ de la arista } e_x; v_i \in V \setminus \{d\}\} \quad (2.5)$$

También se define como el *factor de densidad*, $\delta(e_x)$, como el número de vértices que pertenecen al conjunto $B(e_x)$ dividido por la longitud de la arista auxiliar e_x

$$\delta(e_x) = \frac{|B(e_x)|}{w(e_x)} \quad (2.6)$$

En la figura 2.9, se muestran los 12 subconjuntos representados por rectángulos, para cada arista auxiliar (d, p_x). Cada rectángulo está formado por dos líneas paralelas, que están separadas de la arista auxiliar a una distancia b_x, $1 \leq x \leq 12$ y dos lineas que unen a los extremos de esas dos líneas paralelas. En la tabla 2.1, se muestran los valores correspondientes a cada arista.

Debido a que se requieren k particiones, es que nuestro problema puede caer en alguno de los siguientes casos:

Si $m = k$, entonces esta etapa de partición ha concluido.

Si $m > k$, entonces deben eliminarse aquellas $m - k$ aristas auxiliares, que tengan el menor factor de densidad.

Si $m < k$, entonces deben agregarse $k - m$ aristas auxiliares en aquellas regiones, en donde el factor de densidad es mayor.

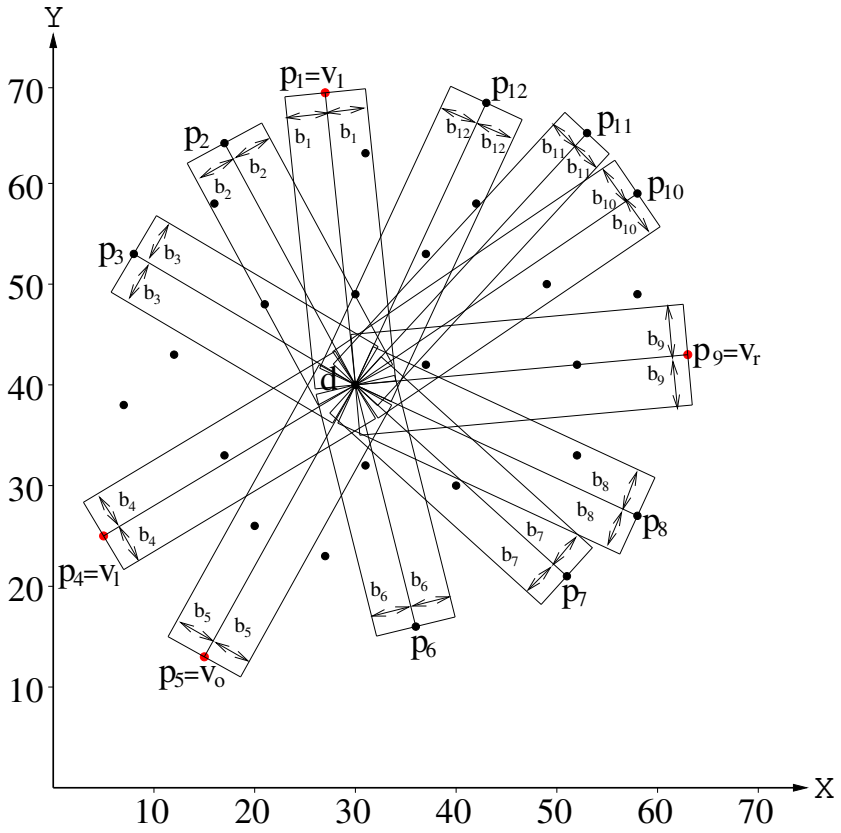

Figura 2.9: Áreas que definen los subconjuntos $B(e_x)$

x	arista auxiliar	$w(d, p_x)$	b_x	$\lvert B(d, p_x)\rvert$	$\delta(d, p_x)$
1	(d, p_1)	29,15	4	3	0,10
2	(d, p_2)	27,29	4,2	4	0,15
3	(d, p_3)	25,55	4,4	2	0,08
4	(d, p_4)	29,15	3,9	2	0,07
5	(d, p_5)	30,89	4,1	2	0,06
6	(d, p_6)	24,74	4	2	0,08
7	(d, p_7)	28,32	3,8	2	0,07
8	(d, p_8)	30,87	4,2	2	0,06
9	(d, p_9)	33,14	5	3	0,09
10	(d, p_{10})	33,84	4	3	0,09
11	(d, p_{11})	33,97	3	1	0,03
12	(d, p_{12})	30,87	3,9	4	0,13

Tabla 2.1: Valores del ejemplo de la figura 2.9

Para nuestro problema de ejemplo tenemos los siguientes casos (ver valores en la tabla 2.1):

- Si tuvieramos que eliminar una arista auxiliar, esta sería (d, p_{11}), dado que tiene el menor factor de densidad, igual a 0,03.
- Si tuvieramos que agregar una arista auxiliar, se agregaría en la zona en la que se encuentra (d, p_2), dado que tiene el mayor factor de densidad, igual a 0,15.

Una vez que ya se tienen k áreas triangulares, las cuales deben incluir al menos un elemento, se ha concluido la *etapa 1* de la *fase 2*; y, se procede con la *etapa 2* para construir un recorrido para cada una de esas áreas triangulares.

Etapa 2: Dada un área triangular, de las k formadas en la etapa anterior; sean v y u los vértices sobre el borde del polígono de esa área con aristas auxiliares $e = (d, v)$ y $e' = (d, u)$. El recorrido se compone con la construcción de los subrecorridos, por las aristas auxiliares e (iniciando en d hacia v) y e' (iniciando en u hacia d), en donde se incluyen a los vértices que pertenecen a $B(e)$ y $B(e')$ respectivamente.

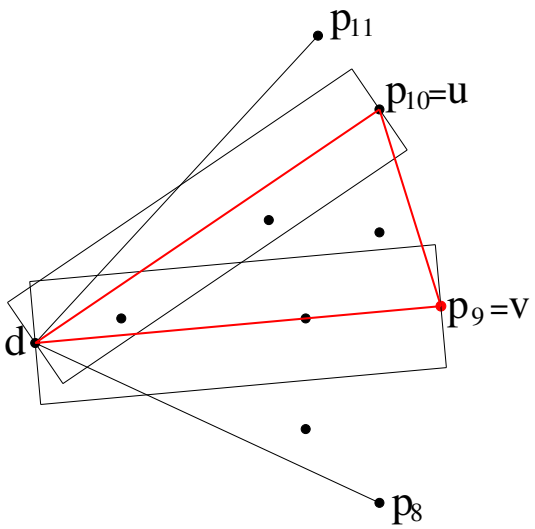

Figura 2.10: Una de las k áreas triangulares del polígono envolvente

Como observación, tenemos que pueden existir los casos en que

$$B(e) \cap B(e') \neq \emptyset$$

$$B(e) \cup B(e') \neq \{v_i | v_i \textit{ está dentro del área triangular } d - v - u - d\}$$

.

Para esos vértices que están en los dos subconjuntos o en ninguno de los subconjuntos, Se debe decidir a cuál de los dos subconjuntos deben pertenecer, con base en la cercanía con otros vértices que cada subconjunto tiene, o con las aristas auxiliares asociadas con esos subconjuntos.

En la figura 2.10, se muestra el área triangular que usaremos como ejemplo para construir su recorrido, donde $v = p_9$ y $u = p_{10}$. Se observa que inicialmente $|B(e)| = 3$ y $|B(e')| = 3$, que estos dos subconjuntos tienen un vértice en común; y, que existe un vértice que no pertenece a ninguno de los dos subconjuntos. Ambos vértices deben incluirse en $B(e)$, dado que ambos vértices están más cercanos a e. Después de esta decisión $|B(e)| = 4$ y $|B(e')| = 2$.

Ahora es turno de iniciar con la construcción del recorrido, que inicialmente incluye al vértice d. Después de manera iterativa, se agregan los vértices que pertenecen a $B(e) \setminus \{v\}$ y finalmente se incluye a v (ecuación 2.7).

$$T = (d, i_1, ..., i_{m-1}, v); \quad i_j, v \in B(e); m = |B(e)| \tag{2.7}$$

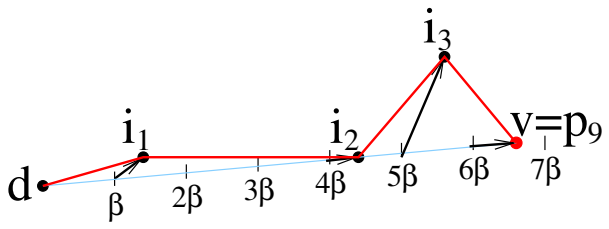

Figura 2.11: Construcción del recorrido del vector e

Para el procedimiento que incluye los vértices de $B(e)$ en el recorrido, se definen β como un factor de grosor y un número real α, $0 < \alpha < 1$; de tal modo que $\beta = \alpha b$, donde b es la distancia que se usó anteriormente para obtener el conjunto $B(e)$.

Inicialmente declaramos un entero $i = 1$ y una variable $v' = d$. Iterativamente se revisa si existe un vértice cuya coordenada (x_i, y_i), es la más alejada desde v' sobre e, que no sea mayor a la distancia $i\beta$ también sobre e, formando un vector $(v', (x_i, y_i))$; de lo contrario no se forma ese vector. El vector formado es un esqueleto para construir el recorrido, sea $\beta(v', (x_i, y_i))$ los vértices que pertenecen al conjunto $B(e)$, tal que están entre las distancias v' y (x_i, y_i); el recorrido se construye agregando los vértices incluidos en $\beta(v', (x_i, y_i))$, desde v' hasta el vértice ubicado en (x_i, y_i), según su cercanía con los vértices con el vector $(v', (x_i, y_i))$. Posteriormente, se incrementa en uno el valor de i y $v' = (i-1)\beta$. El proceso se repite hasta que $i\beta$ rebase el borde del polígono.

En la figura 2.11, se muestra con una línea roja la construcción del recorrido para el vector e. El vector e tiene identificadas las distancias $\beta, 2\beta, ..., 7\beta$. Los únicos vectores, que pudieron generarse son (β, i_1), $(4\beta, i_2)$, $(5\beta, i_3)$ y $(6\beta, v)$.

Una vez que se ha alcanzado al vértice v que está sobre el borde del polígono, ahora de manera análoga se incluyen los vértices del conjunto $B(e')$, desde el vértice u hasta d (ecuación 2.8).

$$T = (d, i_1, ..., i_{m-1}, v, u, {i'}_2, ..., {i'}_{m'}, d) \tag{2.8}$$
$$i_j, v \in B(e);\ m = |B(e)|;\ {i'}_j, u \in B(e');\ m' = |B(e')|$$

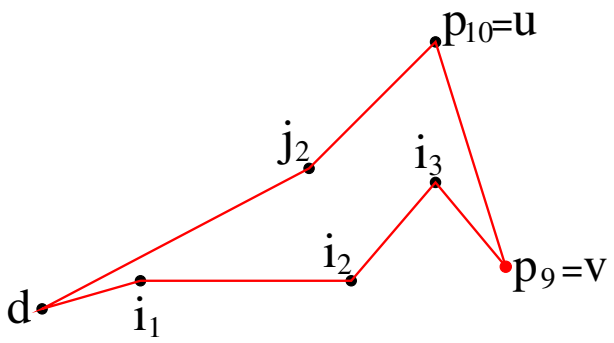

Figura 2.12: Recorrido para el área del triángulo $d - p_9 - p_{10} - d$

En la figura 2.12, se muestra el recorrido resultante para el área del triángulo $d - p_9 - p_{10} - d$

Una vez que se hayan construido los k recorridos para cada área triangular, tendremos completada la solución para la instancia dada.

Como observación podemos citar que existen algunos vértices que estén justo entre dos áreas triangulares, como por ejemplo los vértices que forman al polígono envolvente. Se sugiere para este caso, establecer algún criterio para decidir a que área pertenecerán esos vértices de tal modo que el recorrido que sea generado, dé como resultado un costo total de menor distancia. También debe considerarse que, si fueron agregados puntos para generar aristas auxiliares para la división del poligono envolvente; entonces esos puntos no deben ser incluidos en la solución.

Cabe mencionar que para obtener diferentes resultados, las variables α y b deben ser sintonizadas con la finalidad de obtener menores costos.

2.6. Resultados obtenidos de la literatura para un MTSP

Los resultados presentados en esta sección, corresponden a soluciones de instancias para el *MTSP Bounded*. Por lo tanto, además del conjunto de los vértices V y un número k; en la instancia se incluye un intervalo $[m_{min}, m_{max}]$, que indica la cantidad mínima y máxima de vértices que deben ser incluidos en cada recorrido.

La razón por la cual se están presentando resultados para el *MTSP Bounded*, es debido a que no se encontraron en la literatura resultados para el mismo problema para el cuál estamos diseñando el algoritmo; ya que nuestro diseño no incluye la restricción que limita al algoritmo a incluir una cantidad mínima o máxima de vértices a cada recorrido.

Nomenglatura	Referencia	Nombre del algoritmo
$AC2OptGA$	Colonia de Hormigas, 2Opt y Algoritmos Genéticos	[Harrath *et al.* 2019]
GAL	Algoritmos Genéticos con operadores Locales	[Lo *et al.* 2018]
$M-GELS$	Búsqueda Local Emulación Gravitacional Modificado	[Rostami *et al.* 2015]
$GELS-GA$	Búsqueda Local Emulación Gravitacional y Algoritmo Genético	[Hosseinabadi *et al.* 2014]
$NMACO$	Nueva Modificación al Algoritmo Colonia de Hormigas	[Yousefikhoshbakht y Sedighpour 2012]
$SW+AS_{elite}$	Algoritmo Barrido y Optimización Colonia de Hormigas Elite	[Yousefikhoshbakht y Sedighpour 2012]
ACO	Optimización Colonia de Hormigas	[Junjie and Dingwei 2006]
MGA	Algoritmo Genético Modificado	[Junjie and Dingwei 2006]
$CPLEX$ *optimizer*	Algoritmo que soluciona problemas de programación lineal y relacionados.	[Necula *et al.* 2015] [IBM 2019]

Tabla 2.2: Nomenglatura utilizada para identificar a los algoritmos publicados en la literatura.

	$M-GELS$	$NMACO$	$SW+AS_{elite}$	ACO	MGA
$AC2OptGA$	•	•	•		•
GAL			•	•	
$M-GELS$			•	•	
$GELS-GA$			•	•	
$NMACO$			•	•	•
$SW+AS_{elite}$				•	•
ACO					•

Tabla 2.3: Comparaciones realizadas entre algoritmos en la literatura

En la tabla 2.2, se presenta la nomenglatura y nombre utilizado en los artículos para referirse a sus algoritmos. Como puede observarse, la mayoría de ellos incluye a una metaheurística, ya sea Colonia de Hormigas, ***Ant Colony (ACO)*** o Algoritmos Genéticos ***Genetic Algorithm (GA)***.

En la tabla 2.3, las 6 instancias mostradas en la tabla 2.2, son comparadas a partir de la publicación realizada por [Junjie and Dingwei 2006]. Las conclusiones en estas comparaciones, son las siguientes:

- MGA es un poco superior para problemas con pocos vértices que ACO (instancias pr76, pr152 y pr226); por el contrario, para problemas con mayor cantidad de vértices (pr299, pr439 y pr1002) los resultados favorecen a ACO [Junjie and Dingwei 2006].
- $SW+AS_{elite}$ no mejoró el resultado obtenidos por MGA para las instancias pr76 y pr226; por el contrario, encontró mejores resultados para las instancias pr152, pr299, pr439 y pr1002. Se concluye que para futuras investigaciones, este algoritmo combinado con otras metaheurísticas como Algoritmos Genéticos, Búsqueda Tabú y Recocido Simulado darán mejores resultados [Yousefikhoshbakht *et al.* 2013].
- $NMACO$ compara sus resultados con $SW+AS_{elite}$, ACO y MGA, mejora los resultados para las instancias pr76, pr152, pr299, pr439 y pr1002; se concluye que $NMACO$ produce mejores resultados que MGA [Yousefikhoshbakht y Sedighpour 2012].

- El promedio de los resultados obtenidos por $GELS - GA$ son comparados con los promedios de los resultados obtenidos por $SW + AS_{elite}$ y ACO, obteniéndose una mejora del 11.7 % y 14.5 % respectivamente. Por lo anterior, se concluye que este algoritmo es superior sobre otros algoritmos que resuelven el MTSP, no solo porque obtiene mejores resultados, sino tambień porque el tiempo de ejecución es rápido [Hosseinabadi *et al.* 2014]. Como puede observarse, se comparan los promedios de los resultados debido a que no fueron publicados los valores de la mejor solución obtenida.

- Del mismo modo que $GELS - GA$, $M - GELS$ el promedio de sus resultados fueron comparados con los promedios de los resultados obtenidos por $SW + AS_{elite}$ y ACO; donde se obtiene una mejora de 10.2 % y 13.7 % respectivamente [Rostami *et al.* 2015]. Sin embargo, no mejora a los resultados obtenidos por $GELS - GA$.

- GAL es comparado con $SW + AS_{elite}$ y ACO y obtiene mejores resultados con respecto a esos algoritmos [Lo *et al.* 2018]. Puede observarse que obtiene mejores resultados con respecto al promedio de los valores obtenidos por $GELS - GA$ para las instancias pr226, pr299, pr439 y pr1002.

- $AC2OptGA$ fue comparado con $M - GELS$, $NMACO$ y $SW + AS_{elite}$; solo logró mejorar sus resultados de las instancias pr439 y pr1002 con respecto a los resultados de $M - GELS$ [Harrath *et al.* 2019].

Instancia	Nombre de la instancia
$\|V\| + 1$	es el número de vértices, incluido el depósito, que se encuentran en el conjunto de los vértices de la instancia que define al MTSP
m_{min}	es el número de vértices mínimo que el algoritmo debe incluir en un recorrido sin considerar al depósito
m_{max}	es el número de vértices máximo que el algoritmo debe incluir en un recorrido sin considerar al depósito

Tabla 2.4: La simbología del encabezado de la tabla 2.5.

Instancia					Algoritmo								
Instancia	$\|V\|+1$	k	m_{min}	m_{max}	$CPLEX$	$AC2OptGA$	GAL	$M-GELS$	$GELS-GA$	$NMACO$	$SW+AS_{elite}$	ACO	MGA
pr76	76	5	1	20		159289	153390	147734	**132784**	157413	157495	178597	157444
pr152	152	5	1	40		127520	115874	119206	**105205**	127781	127791	130953	127839
pr226	226	5	1	50		161084	**148051**	160350	152135	167239	167665	167646	166827
pr299	299	5	1	70		77810	**72949**	76654	76554	81261	81998	82106	82176
pr439	439	5	1	100		149675	**143785**	155524	146523	160298	161725	161955	173839
pr1002	1002	5	1	220		351371	**334351**	356341	354341	379042	379871	382198	427269
eil51	51	2	23	27	**442**								
eil51	51	3	15	20	**464**								
eil51	51	5	7	12	**530**								
eil51	51	7	5	10	**605**								
berlin52	52	2	10	41	**7754**								
berlin52	52	3	10	27	**8107**								
berlin52	52	5	6	17	**9126**								
berlin52	52	7	4	17	**9870**								
eil76	76	2	36	39	**559**								
eil76	76	3	21	30	**579**								
eil76	76	5	12	17	**681**								
eil76	76	7	7	15	**760**								
rat99	99	2	46	52	**1351**								
rat99	99	3	27	36	**1519**								
rat99	99	5	13	30	**1856**								
rat99	99	7	9	22	**2292**								

Tabla 2.5: Comparación entre los costos obtenidos por los algoritmos: $CPLEX$, $AC2OptGA$, GAL, $M-GELS$, $GELS-GA$, $NMACO$, $SW+AS_{elite}$, ACO, MGA; para 22 instancias de [TSPLIB]

En la tabla 2.5, se muestran los resultados de los diferentes algoritmos descritos en la literatura, en donde los mejores costos para cada instancia están escritos con letra negrita.

Capítulo 3

Diseño del algoritmo en 2 fases para resolver un MTSP

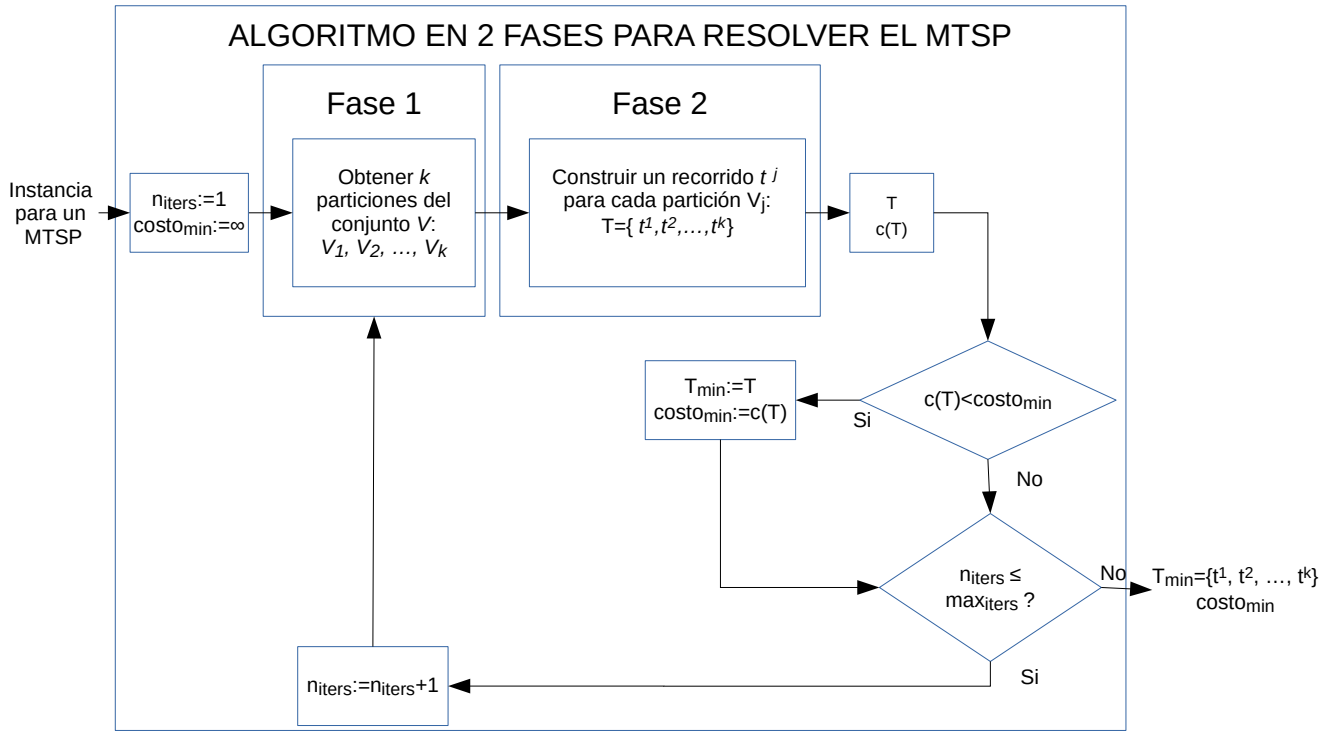

Figura 3.1: Diagrama que muestra las fases del algoritmo en 2 fases

El diagrama a bloques del algoritmo en dos fases para la construcción de k recorridos, diseñado en este proyecto de investigación, se muestra en la figura 3.1; el cual en la fase 1 obtiene k particiones del conjunto V; y en la fase 2, la construcción de un recorrido por cada partición generada en la fase 1. Este algoritmo genera n_{max} soluciones diferentes, dentro de las cuales, se guarda en la variable T_{min} la que menor costo haya obtenido.

La descripción en forma verbal y detallada (pseudocódigo) de cada fase,y posteriormente del algoritmo diseñado en este proyecto, son presentadas en las secciones 3.4 y 3.5 de este capítulo; después de haber descrito el planteamiento del problema (sección 3.1), se hayan descrito las definiciones utilizadas para la solución del problema (sección 3.2); y, descrito los procedimientos que son utilizados en cada fase (sección 3.3).

La principales diferencias que hay entre ambos algoritmos y la aportación de este proyecto son:

- **Fase de construcción de agrupamientos**: se utiliza el concepto de vector auxiliar para construir k particiones en ambos algoritmos. Solo que en nuestro diseño no se utilizan para dividir un polígono envolvente, que encierra a los vértices del conjunto V, en k triángulos; en donde cada área triangular es una partición V_j.

  En vez de eso, como se verá más adelante, se crean inicialmente k vectores auxiliares $\vec{a}_j$ que iterativamente cambian de dirección, para agregar a cada partición V_j al vértice $i \in V \setminus \{d\}$; tal que la distancia angular entre $\vec{a}_j$ e i sea mínima.

  Debido a la posibilidad de tener vectores auxiliares que cambien de dirección, se espera tener mejores agrupaciones que aquellas que se generan a través de la partición de un polígono envolvente.

- **Fase de construcción de recorridos**: Para cada partición generada por la fase de construcción de agrupamientos: *i)* se utiliza el concepto de polígono envolvente para comenzar a construir el recorrido; y, *ii)* de manera iterativa hasta que todos los vértices la partición hayan sido incluidos; se agrega al vértice aún no incluido en el recorrido, de tal forma que incremente el mínimo costo.

  Debido al cambio de algoritmo para esta fase, se mejoró la forma de construir un recorrido para una partición dada.

3.1. Planteamiento matemático del problema

Sea $G = (V, E)$ un grafo completo no dirigido ponderado y k un número entero, que indica la cantidad de vendedores que definen un problema MTSP.

donde:

- $d \in V$ es el vértice identificado como depósito.
- $i \in V \setminus \{d\}$; $1 \leq i \leq n$ son los vértices por incluir dentro de los k recorridos.
- $(i, j) \in E$ es una arista; $i, j \in V$, $i \neq j$.
- $w(i, j) > 0 \in E$ es la distancia euclidiana en un plano 2D de la arista (i, j).

Definición 3.1.1 Un *recorrido j*, t^j, es una secuencia de vértices que comienza en d, luego visita m_j vértices del subconjunto $V \setminus \{d\}$, y termina en el vértice d ; $1 \leq m_j \leq n-k+1$.

$$t^j = (d, i_1, i_2, \cdots, i_{m_j}, d) \tag{3.1}$$

donde:

- j representa al número de recorrido; $1 \leq j \leq k$
- m_j es la cantidad vértices incluidos en el recorrido j; $1 \leq m_j \leq n-k+1$
- $i_l \in V \setminus \{d\}$ son sitios incluidos dentro del recorrido j; $1 \leq l \leq m_j$

Definición 3.1.2 El *costo del recorrido j*, $c(t^j)$, es la suma de los costos de las aristas del conjunto E consideradas en el recorrido j.

$$c(t^j) = w(d, i_1) + w(i_1, i_2) + \cdots + w(i_{m_j-1}, i_{m_j}) + w(i_{m_j}, d)$$

$$c(t^j) = w(d, i_1) + \sum_{l=1}^{m_j-1} w(i_l, i_{l+1}) + w(i_{m_j}, d) \tag{3.2}$$

$$1 \leq j \leq k;\ 1 \leq m_j \leq n-k+1;\ d, i_l \in V$$

Definición 3.1.3 El *costo total de los recorridos*, $c(T)$, es la suma de los costos de los k recorridos:

$$c(T) = c(t^1) + \cdots + c(t^k)$$

$$c(T) = \sum_{j=1}^{k} c(t^j);\ 1 \leq j \leq k \tag{3.3}$$

La *función objetivo* es

$$\text{mín}\ \ c(T) \tag{3.4}$$

sujeta a las siguientes restricciones:

- Todos los vértices, con excepción del depósito, deben ser visitados una sola vez.
- Todos los vendedores inician y concluyen sus recorridos en el depósito, y deben visitar al menos un vértice.

3.2. Definiciones

Definición 3.2.1 El *ángulo de una arista*, $\theta(d,i)$, es el ángulo que hay entre la arista (d,i) y la línea $y = y_d$ tal que $x \geq x_d$.

$$\theta(d,i) = \begin{cases} \arccos \dfrac{x_i - x_d}{w(d,i)} & si \ \arcsin \dfrac{y_i - y_d}{w(d,i)} >= 0 \\ -\arccos \dfrac{x_i - x_d}{w(d,i)} & si \ \arcsin \dfrac{y_i - y_d}{w(d,i)} < 0 \end{cases} \tag{3.5}$$

$$1 \leq i \leq n; \ -\pi < \theta(d,i) \leq \pi; \ d,i \in V$$

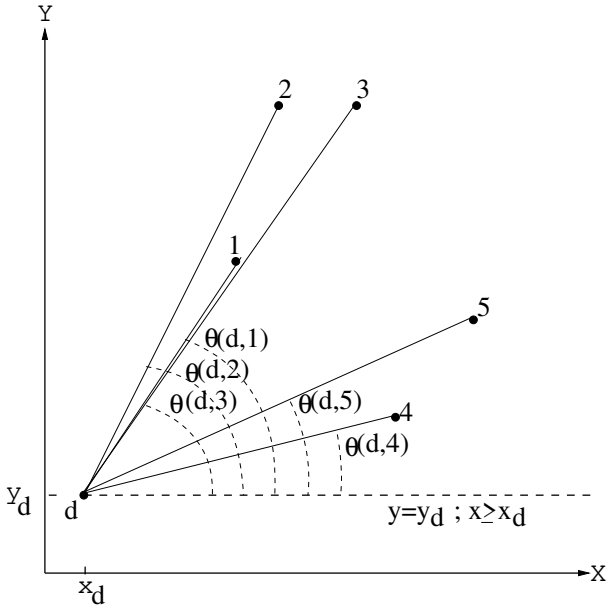

Figura 3.2: Ángulos de las aristas

En el ejemplo de la figura 3.2, se muestran los ángulos $\theta(d,i)$, que forman las aristas (d,i) con respecto a la línea $y = y_d, x \geq x_d$; para un conjunto $V = \{d,1,2,3,4,5\}$; $i \in V \setminus \{d\}$

Definición 3.2.2 Una *partición* j, V_j, es un subconjunto del conjunto V [Rosen 2004] tal que

$$V_1 \cup V_2 \cup ... \cup V_k = V \setminus \{d\} \tag{3.6}$$

$$V_1 \cap V_2 \cap ... \cap V_k = \emptyset \tag{3.7}$$

Definición 3.2.3 Sea A un conjunto de elementos, la *función máximo*[Rosen 2004], máx A, es aquella que devuelve el valor del elemento que pertenece a A con mayor valor.

Un ejemplo para la función máximo es: máx$\{10, 2, 5, 4, 7\} = 10$

Definición 3.2.4 Sea A un conjunto de elementos, la *función mínimo*, mín A, es aquella que devuelve el valor del elemento que pertenece a A con menor valor.

Un ejemplo para la función mínimo es: mín$\{10, 2, 5, 4, 8\} = 2$

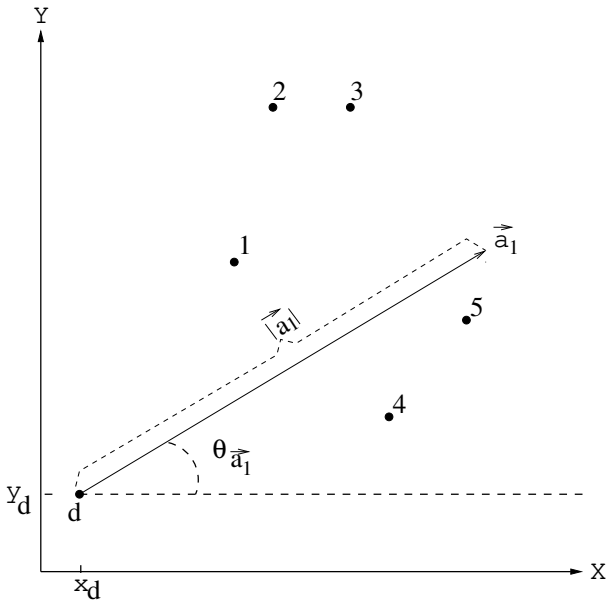

Figura 3.3: Vector auxiliar

Definición 3.2.5 Un *vector auxiliar*, $\vec{a}_j$, es una línea que tiene su origen en d, una magnitud $|\vec{a}_j|$ y una dirección $\theta_{\vec{a}_j}$, que es el ángulo que hay entre él y el eje de las abscisas. [Vakhania *et al.* 2016]

$$x_{\vec{a}_j} = x_d,\ y_{\vec{a}_j} = y_d \tag{3.8}$$

$$|\vec{a}_j| = \begin{cases} \text{máx}\{\ w(d, i_l)\ \} & si \quad V_j \neq \emptyset \\ 0 & si \quad V_j = \emptyset \end{cases} \tag{3.9}$$

$$\theta_{\vec{a}_j} = (parámetro\ variable) \tag{3.10}$$

$$1 \leq j \leq k;\ -\pi < \theta_{\vec{a}_j} \leq \pi;\ i_l \in V_j$$

El valor de $\theta_{\vec{a}_j}$ es modificado por el algoritmo de manera iterativa, con base en la dirección inicial cuando se crea al vector, o la ubicación de los vértices i_l incluidos en la partición V_j.

En el ejemplo de la figura 3.3, se muestra el vector auxiliar $\vec{a}_1$ con coordenadas de origen $x_{\vec{a}_1} = x_d$, $y_{\vec{a}_1} = y_d$, dirección $\theta_{\vec{a}_1}$ y magnitud $|\vec{a}_1| = w(d,3)$; debido a que la arista $(d,3)$ es la de mayor magnitud entre las aristas (d,i_l), donde i_l pertenece a $V_1 = \{1,2,3,4,5\}$.

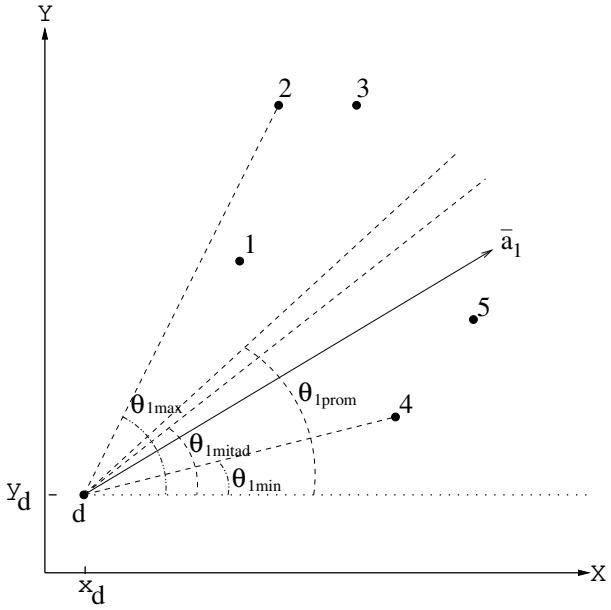

Figura 3.4: Ángulo máximo, mínimo, mitad y promedio de una partición

Definición 3.2.6 El *ángulo máximo de la partición j*, θ_{jmax}, es el ángulo de mayor valor entre las aristas (d,i_l), donde i_l pertenece a la partición V_j.

$$\theta_{jmax} = \text{máx}\{\ \theta(d,i_l)\ \} \tag{3.11}$$
$$-\pi < \theta_{jmax} \leq \pi;\ 1 \leq j \leq k;\ i_l \in V_j$$

Definición 3.2.7 El *ángulo mínimo de la partición j*, θ_{jmin}, es el ángulo de menor valor entre las aristas (d,i_l), donde i_l pertenece a la partición V_j.

$$\theta_{jmin} = \text{mín}\{\ \theta(d,i_l)\ \} \tag{3.12}$$
$$-\pi < \theta_{jmin} \leq \pi;\ 1 \leq j \leq k;\ i_l \in V_j$$

Definición 3.2.8 El *ángulo mitad de la partición j*, θ_{jmitad}, es el promedio de los ángulos θ_{jmax} y θ_{jmin} (ecuaciones 3.11 y 3.12 respectivamente).

$$\theta_{jmitad} = \frac{\theta_{jmin} + \theta_{jmax}}{2} \tag{3.13}$$
$$-\pi < \theta_{jmitad} \leq \pi;\ 1 \leq j \leq k$$

Definición 3.2.9 El *ángulo promedio de la partición j*, θ_{jprom}, es el promedio de los ángulos de la aristas (d, i_l), donde i_l pertenece a la partición V_j.

$$\theta_{jprom} = \frac{1}{m_j} \sum_{l=1}^{m_j} \theta(d, i_l) \tag{3.14}$$

$$-\pi < \theta_{jprom} \leq \pi;\ 1 \leq j \leq k;\ i_l \in V_j; m_j = |V_j|$$

En el ejemplo de la figura 3.4, se muestra una partición $V_1 = \{1, 2, 3, 4, 5\}$, donde la arista $(d, 2)$ es la que mayor ángulo tiene, por lo tanto el valor de ese ángulo es asignado a θ_{1max}; la arista $(d, 4)$ es la que menor ángulo tiene, por lo tanto el valor de ese ángulo es asignado a θ_{1min}; θ_{1mitad} es igual al promedio de los ángulos θ_{1max} y θ_{1min}; y, θ_{1prom} es igual al promedio de los ángulos de las aristas (d, i_l), donde $i_l \in V_j$.

Definición 3.2.10 El *ángulo nuevo*, θ_{nuevo}, es el promedio de los ángulos θ_{jmitad} y θ_{jmax} (ecuaciones 3.13 y 3.11 respectivamente), donde la partición V_j es la que más vértices tiene.

Análogamente se ajusta la dirección del vector auxiliar j calculando el promedio de los ángulos θ_{jmin} y θ_{jmitad} (ecuaciones 3.12 y 3.13 respectivamente), debido a que el ángulo nuevo está dentro del área en la cuál los vértices de V_j están ubicados.

$$\theta_{nuevo} = \frac{\theta_{jmitad} + \theta_{jmax}}{2} \tag{3.15}$$

$$\theta_{\vec{a}_j} = \frac{\theta_{jmin} + \theta_{jmitad}}{2} \tag{3.16}$$

$$-\pi < \theta_{nuevo}, \theta_{\vec{a}_j}, \theta_{jmin}, \theta_{jmitad}, \theta_{jmax} \leq \pi;\ 1 \leq j \leq k$$

Definición 3.2.11 El *ajuste de un vector auxiliar*, consiste en cambiar su dirección con base en la ubicación de los vértices que pertenecen a la partición asociada a ese vector.

El nuevo valor asignado a la dirección del vector auxiliar puede ser θ_{prom} (Ecuación 3.14), para centrar al vector entre todos los vértices incluidos dentro de su partición asociada o θ_{nueva} en el caso de reubicar al vector cuando su partición asociada no contiene vértices.

$$\theta_{\vec{a}_j} = \begin{cases} \theta_{jprom} & si \quad V_j \neq \emptyset \\ \theta_{nueva} & si \quad V_j = \emptyset \end{cases} \tag{3.17}$$

$$-\pi < \theta_{\vec{a}_j}, \theta_{jprom}, \theta_{nueva} \leq \pi;\ 1 \leq j \leq k$$

Definición 3.2.12 La *diferencia angular*, $\delta(i, \vec{a}_j)$, es el ángulo que hay entre la arista (d, i) y el véctor auxiliar j.

$$\delta(i,\vec{a}_j) = \begin{cases} |\theta_{\vec{a}_j} - \theta(d,i)| & si \quad 0 \leq |\theta_{\vec{a}_j} - \theta(d,i)| \leq \pi \\ 2\pi - |\theta_{\vec{a}_j} - \theta(d,i)| & si \quad \pi < |\theta_{\vec{a}_j} - \theta(d,i)| \leq 2\pi \end{cases} \tag{3.18}$$

$$0 \leq \delta(i,\vec{a}_j) \leq \pi;\ 1 \leq j \leq k;\ 1 \leq i \leq n;\ d, i \in V$$

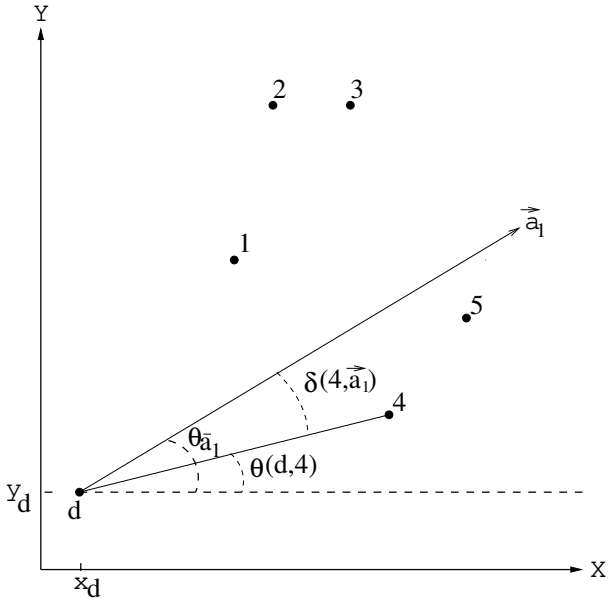

Figura 3.5: Diferencia angular

En el ejemplo de la figura 3.5, se muestra la diferencia angular entre $(d, 4)$ y $\vec{a}_1$, en donde claramente $|\theta_{\vec{a}_1} - \theta(d,4)|$ es menor o igual a π.

Definición 3.2.13 Los *vértices extremos*, v_{et}, v_{el}, v_{eb} y v_{er}, [Vakhania *et al.* 2016], son aquellos que pertenecen a un subconjunto $V_j \cup \{d\}$ tales que

$$T' = \{i_l | y_{i_l} \text{ tiene un valor máximo}; i_l \in V_j \cup \{d\}\}$$
$$v_{et} = i'_l \ | x_{i'_l} \text{ tiene un valor máximo}; i'_l \in T' \tag{3.19}$$
$$L' = \{i_l | x_{i_l} \text{ tiene un valor minimo}; i_l \in V_j \cup \{d\}\}$$
$$v_{el} = i'_l \ | y_{i'_l} \text{ tiene un valor máximo}; i'_l \in L' \tag{3.20}$$
$$B' = \{i_l | y_{i_l} \text{ tiene un valor minimo}; i_l \in V_j \cup \{d\}\}$$
$$v_{eb} = i'_l \ | x_{i'_l} \text{ tiene un valor minimo}; i'_l \in B'\} \tag{3.21}$$
$$R' = \{i_l | x_{i_l} \text{ tiene un valor máximo}; i_l \in V_j \cup \{d\}\}$$
$$v_{er} = i'_l \ | y_{i'_l} \text{ tiene un valor minimo}; i'_l \in R' \tag{3.22}$$

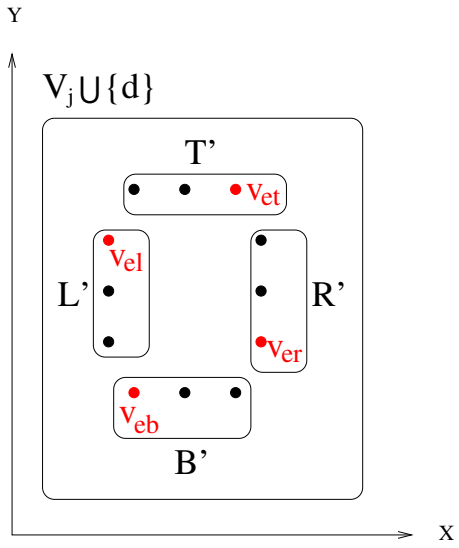

Figura 3.6: Ubicación de los vértices extremos v_{et}, v_{el}, v_{eb} y v_{er} en un subconjunto $V_j \cup \{d\}$

En la figura 3.6, se muestra de manera gráfica un subconjunto $V_j \cup \{d\}$; dentro de él se encuentran los subconjuntos T' con 3 vértices cuya coordenada y es máxima, L' con 3 vértices cuya coordenada x es mínima, B' con 3 vértices cuya coordenada y es mínima; y, R' con 3 vértices cuya coordenada x es máxima. El vértice v_{et} es aquel que pertenece al conjunto T' cuya coordenada x es máxima, el vértice v_{el} es aquel que pertenece al conjunto L' cuya coordenada y es máxima, el vértice v_{eb} es aquel que pertenece al conjunto B' cuya coordenada x es mínima; y, el vértice v_{er} es aquel que pertenece al conjunto R' cuya coordenada y es mínima.

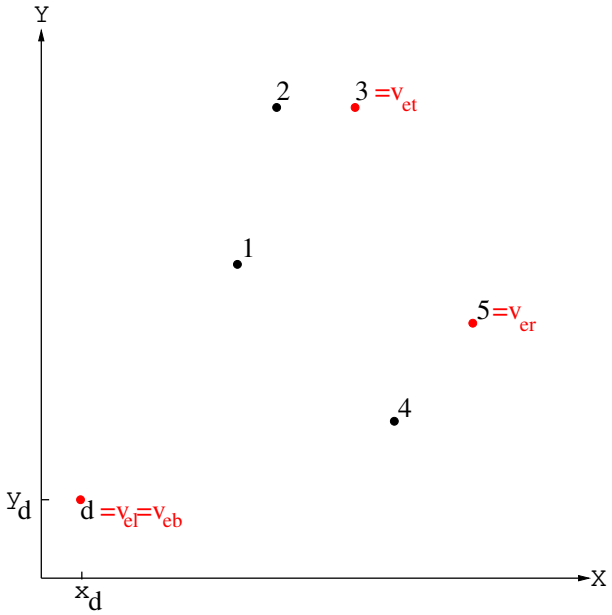

Figura 3.7: Vértices extremos v_{et}, v_{el}, v_{eb} y v_{er}

En el ejemplo de la figura 3.7, se muestra que $T' = \{2, 3\}$, $L' = B' = \{d\}$, $R' = \{5\}$; $v_{et} = 3$, $v_{el} = v_{eb} = d$, $v_{er} = 5$

Definición 3.2.14 Un *polígono envolvente* [Vakhania *et al.* 2016], P_j, es una figura geométrica que encierra a todos los vértices de un subconjunto $V_j \cup \{d\}$, formada por los vértices y aristas que pertenecen a ese subconjunto y cuyos ángulos internos, entre esas aristas tienen un ángulo menor a π radianes; por lo tanto, es un polígono convexo.

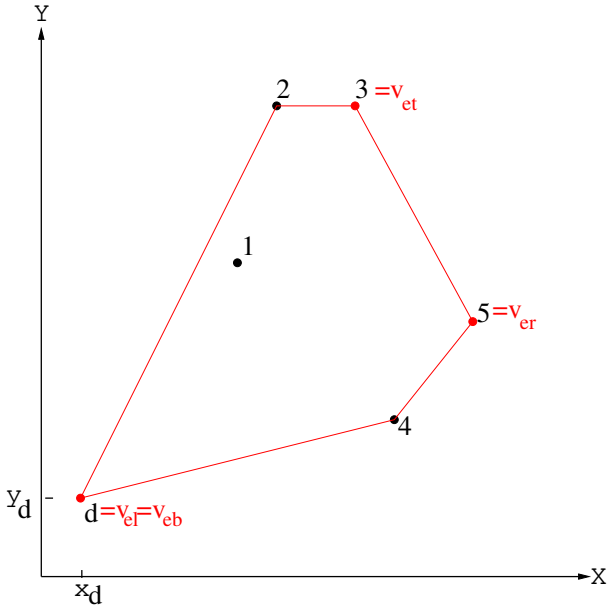

Figura 3.8: Polígono envolvente para un subconjunto

En el ejemplo de la figura 3.8; se muestra el polígono $P_1 = (3, 2, d, 4, 5)$ para un subconjunto $V_1 \cup \{d\}$ donde $V_1 = \{1, 2, 3, 4, 5\}$.

3.3. Procedimientos

Procedimiento 3.3.1 *obtener k vectores auxiliares*

Nuestro diseño, inicialmente requiere de k vectores auxiliares separados por una distancia angular $\frac{2\pi}{k}$, que posteriormente servirán para particionar al subconjunto $V \setminus \{d\}$.

Para establecer la dirección inicial de los vectores auxiliares, primero se asigna al vector auxiliar 1, $\vec{a}_1$, un ángulo aleatorio cuyo valor esté entre 0 y $\frac{2\pi}{k}$; y, posteriormente con base en el ángulo de ese vector, se agregan los $k-1$ vectores auxiliares restantes con la diferencia angular ya mencionada.

Los parámetros de entrada son las variables: θ, que tiene un valor aleatorio entre 0 y $\frac{2\pi}{k}$; x_d y y_d, que son las coordenadas del depósito, el conjunto de los vértices V; y, k, que es el número de recorridos solicitados en la instancia.

El procedimiento inicializa al conjunto A con ningún vector auxiliar, el índice j igual a 1 y un ángulo θ' igual al ángulo θ. De manera iterativa mientras θ' sea menor o igual a π, se agrega al conjunto A el vector auxiliar $\vec{a}_j$, con los valores $x_{\vec{a}_j} := x_d$, $y_{\vec{a}_j} := y_d$, $|\vec{a}_j| := \text{máx}_{i \in V \setminus \{d\}}\{w(d,i)\}$ y $\theta_{\vec{a}_j} := \theta'$; y, se incrementa en 1 el índice j y en $\frac{2\pi}{k}$ el valor de θ'.

Luego se establece para θ' el valor de $\theta - \frac{2\pi}{k}$; y de manera iterativa mientras θ' sea mayor a $-\pi$, se agrega al conjunto A el vector auxiliar $\vec{a}_j$ con los valores $x_{\vec{a}_j} := x_d$, $x_{\vec{a}_j} := y_d$, $|\vec{a}_j| := \text{máx}_{i \in V \setminus \{d\}}\{w(d,i)\}$ y $\theta_{\vec{a}_j} := \theta'$; y se incrementa en 1 el índice j y se decrementa en $\frac{2\pi}{k}$ el valor de θ'.

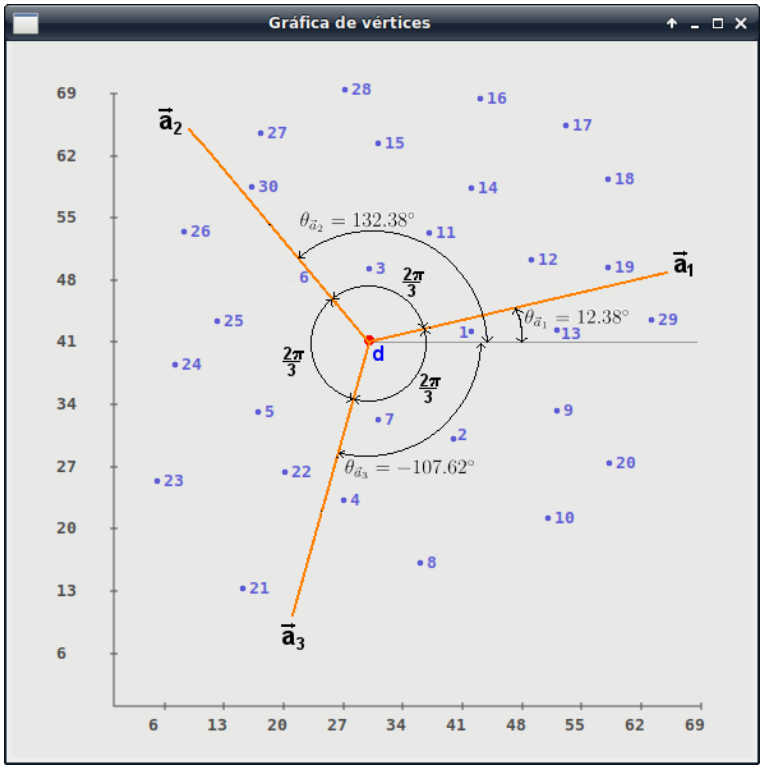

Figura 3.9: Ejemplo de 3 vectores auxiliares iniciales para la instancia del apéndice B, $k = 3$

En el ejemplo que se muestra en la figura 3.9, para la instancia del apéndice B con 3 vendedores; de manera aleatoria la dirección para $\theta_{\vec{a}_1}$ es de 12,38° con respecto al eje de las abscisas. Posteriormente los ángulos para $\theta_{\vec{a}_2}$ y $\theta_{\vec{a}_3}$ tienen un ángulo igual a $12{,}38° + 120° = 132{,}38°$ y $12{,}38° - 120° = -107{,}62°$ respectivamente, con respecto al eje de las abscisas. La magnitud para los 3 vectores es $|\vec{a}_1| = |\vec{a}_2| = |\vec{a}_3| = w(d, 17)$.

PROCEDURE obtenerVectoresAuxiliaresIniciales(θ, x_d, y_d, V, k)

{

$A := \emptyset$ *// Inicialmente no hay vectores auxiliares en el conjunto A*

$j := 1$ *// Lleva el control de los k vectores por crear,* $1 \leq j \leq k$

$\theta' := \theta$ *// θ' es una variable temporal para asignar un ángulo a los nuevos vectores*

WHILE $\theta' \leq \pi$ **DO** *// Ciclo para crear los vectores con ángulo positivo*

$x_{\vec{a}_j} := x_d$ *// Las coordenadas del inicio del vector son las coordenadas de d*

$y_{\vec{a}_j} := y_d$

$|\vec{a}_j| := \max_{i \in V \setminus \{d\}} \{w(d,i)\}$ *// La magnitud del vector auxiliar*

$\theta_{\vec{a}_j} := \theta'$ *//Se asigna el valor actual de θ'*

$A := A \cup \{\vec{a}_j\}$ *// Se agrega el vector $\vec{a}_j$ al conjunto A*

$j := j + 1$ *// Se actualizan los valores de j y θ'*

$\theta' := \theta' + \frac{2\pi}{k}$

$\theta' := \theta - \frac{2\pi}{k}$ *// Se inicializa nuevamente θ' con un ángulo negativo*

WHILE $\theta' > -\pi$ **DO** *// Ciclo para agregar los vectores con ángulo negativo*

$x_{\vec{a}_j} := x_d$ *// Las instrucciones de este ciclo son análogas al ciclo anterior*

$y_{\vec{a}_j} := y_d$

$|\vec{a}_j| := \max_{i \in V \setminus \{d\}} \{w(d,i)\}$

$\theta_{\vec{a}_j} := \theta'$

$A := A \cup \{\vec{a}_j\}$

$j := j + 1$

$\theta' := \theta' - \frac{2\pi}{k}$

RETURN A *// El procedimiento da como resultado: $A = \{\vec{a}_1, \vec{a}_2, \cdots, \vec{a}_k\}$*

}

Procedimiento 3.3.2 *obtener k subconjuntos*

En la *fase 1* de nuestro diseño, se requiere obtener k particiones del conjunto $V \setminus \{d\}$, los cuales se obtienen de manera iterativa mediante la división de ese conjunto en k subconjuntos.

Para obtener k subconjuntos de V, se utiliza la siguiente ecuación:

$$V_j = \{\ i \mid \textit{la distancia angular entre } i \textit{ y } \vec{a}_j \textit{ es minima}; 1 \leq j \leq k, i \in V \setminus \{d\}\} \tag{3.23}$$

Los parámetros de entrada son las variables A, que es el conjunto de los vectores auxiliares; V, que es el conjunto de los vértices dado en la instancia y k, que es el número de recorridos dados en la instancia.

El procedimiento inicializa al conjunto W sin particiones y las k particiones V_j sin vértices. De manera iterativa para $i = 1$ a n, se inicializa la variable temporal $\delta_{min} = 2\pi$, donde 2π es un valor máximo para esta variable; se calcula la distancia angular entre el vértice i con los k vectores auxiliares $\vec{a}_j$. Si $\delta(i, \vec{a}_j) < \delta_{min}$ entonces se guardan los valores $\delta_{min} = \delta(i, \vec{a}_j)$ y $l = j$. Al final de las k comparaciones, se agrega a la partición V_l el vértice i.

Finalmente, se agregan las particiones V_j al conjunto de las particiones W.

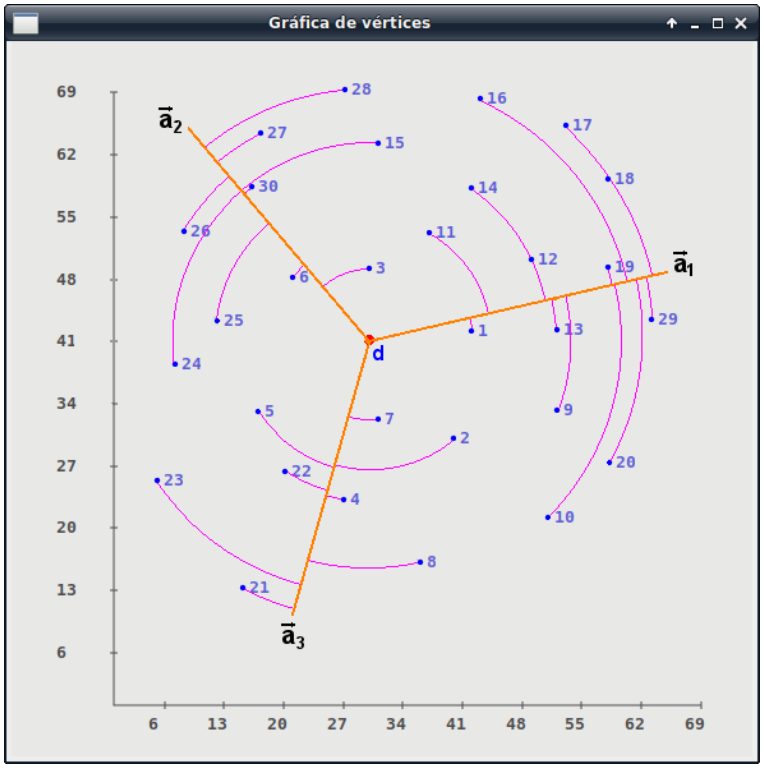

Figura 3.10: Ejemplo de las 3 particiones obtenidas para la instancia del apéndice B, $k = 3$

El ejemplo del resultado de este algoritmo se muestra en la figura 3.10 para la instancia del apéndice B y $k = 3$; $W = \{V_1, V_2, V_3\}$, donde $V_1 = \{1, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 29\}$, $V_2 = \{3, 6, 15, 24, 25, 26, 27, 28, 30\}$ y $V_3 = \{2, 4, 5, 7, 8, 21, 22, 23\}$

PROCEDURE obtener_k_Subconjuntos(A, V, k)

{

$W := \emptyset$ *// Inicialmente no hay particiones en el conjunto W*

FOR $j := 1$ **TO** k **DO**

$V_j := \emptyset$ *// Inicialmente no hay vértices en las particiones* V_j

FOR $i := 1$ **TO** n **DO** *// El índice i se utiliza para recorrer a todos los vértices de* $V \setminus \{d\}$

$\delta_{min} := 2\pi$ *//* δ_{min} *es una variable temporal que se inicializa con un valor máximo igual a* 2π

FOR $j := 1$ **TO** k **DO** *// El índice j se utiliza para recorrer los k vectores auxiliares*

IF $\delta(i, \vec{a}_j) < \delta_{min}$ **THEN** *// usando la ecuación 3.18 para calcular distancia angular*

$l := j$ *// Se guarda el valor del índice j en l*

$\delta_{min} := \delta(i, \vec{a}_j)$ *//Se guarda la distancia angular con menor valor en* δ_{min}

$V_l := V_l \cup \{i\}$ *// Agregar i a la partición* V_l

FOR $j := 1$ **TO** k **DO**

$W := W \cup \{V_j\}$ *// Agregar las k particiones* V_j *a W*

RETURN W *//El procedimiento da como resultado:* $W = \{V_1, \cdots, V_k\}$

}

Procedimiento 3.3.3 *obtener 2 particiones*

Si despúes de obtener las k particiones existe una partición j" que no contiene vértices, $V_{j"} = \emptyset$, entonces se requiere obtener 2 particiones V_j y V_m de un subconjunto $V" \subset V$, donde $V"$ es una partición igual, a la que tenía más vértices justo antes de ejecutar este procedimiento.

Análogamente al procedimiento 3.3.2, para obtener 2 particiones de un subconjunto $V"$ se utilizan las siguientes ecuaciones:

$$V_j = \{\ i_l \mid \textit{la distancia angular entre } i_l \textit{ y } \vec{a}_{j"} \textit{ es minima}; i_l \in V"\} \tag{3.24}$$

$$V_m = \{\ i_l \mid \textit{la distancia angular entre } i_l \textit{ y } \vec{a}_{m"} \textit{ es minima}; i_l \in V"\} \tag{3.25}$$

Se tienen como parámetros de entrada las variables $V"$, que es un subconjunto temporal igual a la partición que tenía la mayor cantidad de vértices; $\vec{a}_{j"}$ y $\vec{a}_{m"}$, que son los vectores auxiliares con los cuales se generan las particiones V_j y V_m.

El procedimiento inicializa al conjunto V_j y V_m sin vértices. De manera iterativa desde $l = 1$ a $|V"|$ se calculan y comparan las distancias angulares $\delta(i_l, \vec{a}_{j"})$ y $\delta(i_l, \vec{a}_{m"})$; si $\delta(i_l, \vec{a}_{j"})$ es menor que $\delta(i_l, \vec{a}_{m"})$ entonces i_l se agrega a V_j, de lo contrario, se agrega a V_m.

PROCEDURE obtener_2_Particiones($V", \vec{a}_{j"}, \vec{a}_{m"}$)

```
{
  // V" = {i_1, ··· , i_|V"|}
  V_j := ∅ // Inicialmente la partición V_j no tiene vértices
  V_m := ∅ // Inicialmente la partición V_m no tiene vértices
  FOR l := 1 TO |V"| DO // El índice l se utiliza para recorrer los vértices de V"
    IF δ(i_l, a_j") < δ(i_l, a_m") THEN // usando la ecuación 3.18 para encontrar distancia angular
      V_j := V_j ∪ {i_l} // Agregar i_l a la partición V_j
    ELSE
      V_m := V_m ∪ {i_l} // Agregar i_l a la partición V_m
  RETURN V_j, V_m
}
```

Procedimiento 3.3.4 *construir polígono envolvente*

Para la *fase 2* de nuestro algoritmo, se requiere inicialmente la construcción de un recorrido equivalente al polígono envolvente, descrito en la definición 3.2.14 de este capítulo, para cada una de las k particiones.

El procedimiento tiene como parámetros de entrada las variables V_j, que es la partición de la que se construirá el polígono envolvente y V, que es el conjunto de los vértices dados en la instancia.

Inicialmente se obtienen los vértices extremos v_{et}, v_{el}, v_{eb} y v_{er} del subconjunto $V_j \cup \{d\}$; y, posteriormente se construye el polígono envolvente en cuatro etapas: 1) v_{et} a v_{el}, 2) v_{el} a v_{eb}, 3) v_{eb} a v_{er} y 4) v_{er} a v_{et}.

Para obtener los vértices extremos, se inicializan las variables v_{et}, v_{el}, v_{eb} y v_{er} igual a d. Luego para cada vértice que pertenece a V_j, se comparan sus coordenadas, con las coordenadas de los vértices extremos. Tal como lo indican las ecuaciones 3.19, 3.20, 3.21 y 3.22.

Para la construcción del polígono envolvente se utilizan las variables t^j, que es una lista en la que se irá guardando la secuencia de vértices que construyen al polígono envolvente, y v_p, que es una variable que guarda al último vértice incluido en t^j, a las que inicialmente se les asigna y se agrega al vértice v_{et} respectivamente.

Luego se verifica si $v_{et} \neq v_{el}$, $v_{el} \neq v_{eb}$, $v_{eb} \neq v_{er}$ ó $v_{er} \neq v_{et}$ para las etapas 1, 2, 3 y 4 respectivamente. Si esta condición no se cumple se considera que esa etapa no existe, de lo contrario, de manera iterativa mientras que $v_{el} \neq v_p$ y $v_{eb} \neq v_p$ y $v_{er} \neq v_p$ y $v_{et} \neq v_p$, se genera un subconjunto $V^* \subset (V_j \cup \{d\})$, que incluya a los elementos que están entre el nodo v_p y el nodo de destino v_{el}, v_{eb}, v_{er} y v_{et} para las etapas 1, 2, 3 y 4 respectivamente. Luego se actualiza al vértice v_p con el vértice $i_{l^*} \in V^*$ cuyo ángulo, con respecto a v_p sea mínimo y se agrega al polígono t^j.

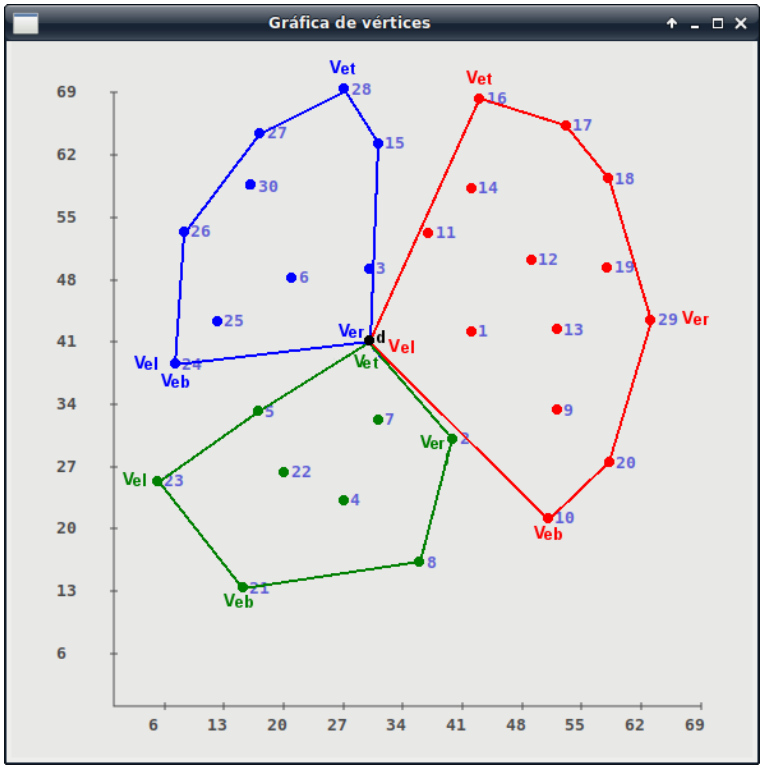

Figura 3.11: Vértices extremos y polígonos envolventes para cada partición de la instancia del apéndice B, $k = 3$

El ejemplo del resultado de este algoritmo se muestra en la figura 3.11 para la instancia del apéndice B y $k = 3$; Para cada partición, se muestran sus respectivos vértices extremos y polígonos envolventes:

- Partición V_1:

  $v_{et} = 16$, $v_{el} = d$, $v_{eb} = 10$ y $v_{er} = 29$

  $P_1 = (16, d, 10, 20, 29, 18, 17)$

- Partición V_2:

  $v_{et} = 28$, $v_{el} = v_{eb} = 24$ y $v_{er} = d$

  $P_2 = (28, 27, 26, 24, d, 3, 15)$

- Partición V_3:

  $v_{et} = d$, $v_{el} = 23$, $v_{eb} = 21$ y $v_{er} = 2$

  $P_3 = (d, 5, 23, 21, 8, 2)$

PROCEDURE construirPoligono(V_j, V)

{
  $v_{et} := d$, $v_{el} := d$, $v_{eb} := d$, $v_{er} := d$
  FOR $l := 1$ **TO** m_j **DO** *// obtener vértices extremos*
    IF $[y_{i_l} > y_{v_{et}}$ **OR** $(y_{i_l} = y_{v_{et}}$ **AND** $x_{i_l} > x_{v_{et}})]$ **THEN** *// Usando ecuación 3.19*
      $v_{et} := i_l$
    IF $[x_{i_l} < x_{v_{el}}$ **OR** $(x_{i_l} = x_{v_{el}}$ **AND** $y_{i_l} > y_{v_{el}})]$ **THEN** *// Usando ecuación 3.20*
      $v_{el} := i_l$
    IF $[y_{i_l} < y_{v_{eb}}$ **OR** $(y_{i_l} = y_{v_{eb}}$ **AND** $x_{i_l} < x_{v_{eb}})]$ **THEN** *// Usando ecuación 3.21*
      $v_{eb} := i_l$
    IF $[x_{i_l} > x_{v_{er}}$ **OR** $(x_{i_l} = x_{v_{er}}$ **AND** $y_{i_l} < y_{v_{er}})]$ **THEN** *// Usando ecuación 3.22*
      $v_{er} := i_l$
  $v_p := v_{et}$ *// Inicia la construcción del recorrido*
  $t^j := \{v_p\}$ *// Inicialmente el recorrido es el vértice superior*
    FOR etapa:=1 **TO** 4 **DO**
      IF $[(v_{et} \neq v_{el}$ **AND** $etapa = 1)$ **OR** $(v_{el} \neq v_{eb}$ **AND** $etapa = 2)$ **OR**
        $(v_{eb} \neq v_{er}$ **AND** $etapa = 3)$ **OR** $(v_{er} \neq v_{et}$ **AND** $etapa = 4)]$ **THEN**
        DO *// Se genera un subconjunto V^* delimitado por v_p y un vértice de destino según la etapa*
          IF etapa=1 **THEN**
            $V^* := \{i_l | x_{i_l} < x_{v_p}$ **AND** $y_{i_l} \geq y_{v_{el}}; i_l \in V_j\}$
          IF etapa=2 **THEN**
            $V^* := \{i_l | x_{i_l} \leq x_{v_{eb}}$ **AND** $y_{i_l} < y_{v_p}; i_l \in V_j\}$
          IF etapa=3 **THEN**
            $V^* := \{i_l | x_{i_l} > x_{v_p}$ **AND** $y_{i_l} \leq y_{v_{er}}; i_l \in V_j\}$
          IF etapa=4 **THEN**
            $V^* := \{i_l | x_{i_l} \geq x_{v_{et}}$ **AND** $y_{i_l} > y_{v_p}; i_l \in V_j\}$
          $\theta_{min} := 2\pi$ *// Se busca al vértice i_{l^*} que tenga un ángulo mínimo con respecto a v_p*
          FOR $l^* := 1$ **TO** $|V^*|$ **DO** *// $i_{l^*} \in V^*$*
            $\theta_{l^*} := obtenerAnguloDeArista(v_p, i_{l^*})$
            IF $\theta_{l^*} < \theta_{min}$ **DO**
              $\theta_{min} := \theta_{l^*}$
              $v_p := i_{l^*}$ *// v_p es el nuevo vértice para construir el polígono*
          IF $v_p \neq v_{et}$ **THEN** *// Se agrega al vértice v_p al polígono t^j*
            $t^j := t^j \cup \{v_p\}$
      WHILE $(v_{el} \neq v_p$ **AND** $v_{eb} \neq v_p$ **AND** $v_{er} \neq v_p$ **AND** $v_{et} \neq v_p)$
  RETURN t^j
}

Procedimiento 3.3.5 *construir recorrido*

Este procedimiento se ejecuta después del procedimiento 3.3.4, en la *fase 2* de nuestro algoritmo e inserta al recorrido j a aquellos vértices que pertenecen a la partición j que no fueron incluidos.

Se tienen como parámetros de entrada las variables t^j, que es un recorrido j equivalente a un polígono envolvente; V_j, que es la partición j asociada al recorrido j y V, que es el conjunto de los vértices.

El algoritmo inicialmente, calcula el *costo* del recorrido equivalente al polígono envolvente. De manera iterativa, se agrega al vértice $i_l \in V_j \setminus t^j$ que agregue un costo mínimo, $costo_{min}$, por haber sido insertado entre los vértices v_p y v_{p+1} dentro del recorrido t^j; y, se actualiza el *costo* del recorrido t^j. Esta operación, se repite hasta que todos los vértices de la partición V_j estén incluidos en el recorrido t^j.

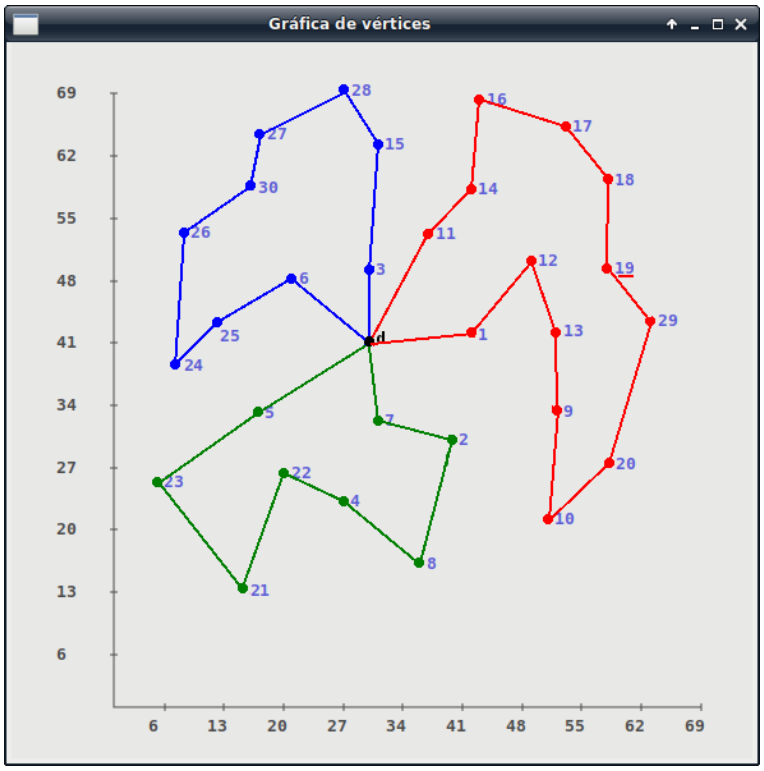

Figura 3.12: Recorridos para cada partición de la instancia del apéndice B, $k = 3$

El ejemplo del resultado de este algoritmo se muestra en la figura 3.12 para la instancia del apéndice B y $k = 3$; Para cada partición, se muestran los recorridos construidos por el algoritmo: $t^1 = (d, 1, 12, 13, 9, 10, 20, 29, 19, 18, 17, 16, 14, 11, d)$, $t^2 = (d, 3, 15, 28, 27, 30, 26, 24, 25, 6, d)$; y, $t^3 = (d, 3, 23, 21, 22, 4, 8, 2, 7)$.

PROCEDURE construirRecorrido(t^j, V_j, V)

{

$costo := \sum_{l=1}^{|t^j|-1} w(i_l, i_{l+1}) + w(i_{|t^j|}, i_1)$ //$t^j = (i_1, \cdots, i_{|t^j|})$, $i_l \in V$

$V" := V_j \setminus t^j$ // *V" es el conjunto de los elementos de la partición V_j que no están en t^j*

WHILE $|V_j| - |t^j| > 0$ **DO** // *mientras existan vértices de V_j no considerados en t^j*

$costo_{min} := \infty$ // *Se inicializa a la variable temporal nuevoCosto con un valor muy grande*

$l := 1$ // *El índice l recorre los vértices del conjunto V"*

WHILE $l \leq |V"|$ **DO** // *Ciclo para recorrer los elementos de V"*

$p := 1$ // *El índice p recorre los vértices del recorrido t^j*

WHILE $p \leq |t^j| - 1$ **DO** // *Ciclo para recorrer los elementos de t^j*

$costo_l := -w(v_{i_p}, v_{i_{p+1}}) + w(v_{i_p}, v_{i_l}) + w(v_{i_l}, v_{i_{p+1}})$

// *Se calcula el costo por insertar el vértice v_{i_l} entre los vértices v_{i_p} y $v_{i_{p+1}}$*

IF $costo_l < costo_{min}$ **THEN** // *Si el $costo_l$ es menor que $costo_{min}$*

$costo_{min} := costo_l$ // *Se actualiza el valor de $costo_{min}$*

$v_{imin} := v_{i_l}$ // *y se guardan los vértices v_{i_l}, v_{i_p} y $v_{i_{p+1}}$*

$v_{pmin} := v_{i_p}$ // *en las variables v_{imin}, v_{pmin} y v_{qmin} respectivamente*

$v_{qmin} := v_{i_{p+1}}$

$p := p + 1$ // *Se incrementa p para hacer cálculos con el siguiente elemento*

$costo_l := -w(v_{i_{|V"|}}, v_{i_1}) + w(v_{i_{|V"|}}, v_{i_l}) + w(v_{i_l}, v_{i_1})$ // *Cálculo análogo al del ciclo anterior*

IF $costo_l < costo_{min}$ **THEN**

$costo_{min} := costo_l$

$v_{imin} := v_{i_l}$

$v_{pmin} := v_{i_{|V"|}}$

$v_{qmin} := v_{i_1}$

$l := l + 1$ // *Se incrementa l para hacer cálculos con el siguiente vértice*

Insertar el vértice v_{imin} entre los vértices v_{pmin} y v_{qmin} dentro del recorrido t^j

$V" := V" \setminus v_{imin}$ // *Quitar v_{imin} del conjunto V"*

$costo := costo + costo_{min}$ // *Actualizar el costo por haber insertado al vértice v_{imin}*

RETURN $t^j, costo$ // *El procedimiento devuelve el recorrido t^j y el costo del recorrido*

}

3.4. Descripción de las fases del algoritmo para el MTSP

Como se ha mencionado en el título de esta tesis y en la introducción de este capítulo, nuestro algoritmo heurístico está compuesto por 2 fases:

- *FASE 1*: Obtener k particiones del conjunto V.
- *FASE 2*: Construir un recorrido t^j para cada partición V_j.

FASE 1: obtener k particiones

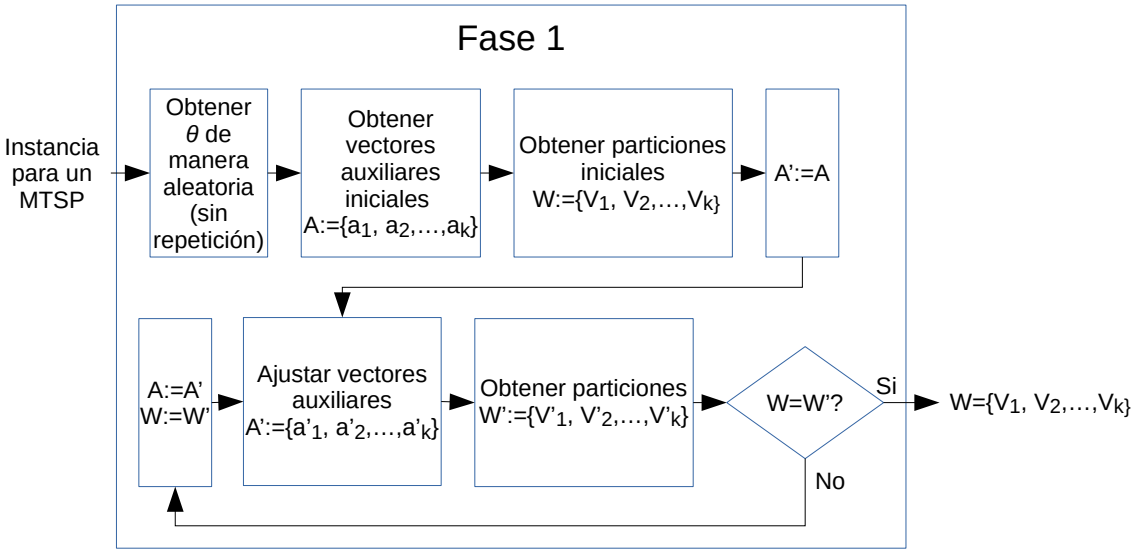

Figura 3.13: Diagrama de bloques de la fase 1 para obtener k particiones.

Se tienen como parámetros de entrada las variables V, que es el conjunto de los vértices y k, que es el número de recorridos, en donde ambos parámetros son dados en la instancia.

Inicialmente, se obtiene un valor θ de manera aleatoria y sin repetición [2] para generar k vectores auxiliares y k particiones asociadas a esos vectores. De manera iterativa hasta que todas las particiones tengan al menos un elemento, se ajustan las direcciones de aquellos vectores cuyas particiones no tengan ningún vértice incluido a una dirección nueva, como se describe en la ecuación 3.17; y, se generan sus particiones.

Posteriormente, se hace una copia de los vectores auxiliares A a A' y se inicializa la variable $iguales$ igual a $FALSE$; por lo que de manera iterativa mientras $iguales = FALSE$, se ajustan los ángulos de los vectores auxiliares de A' a una dirección promedio con respecto a su partición asociada, descrito en la ecuación 3.14; generándose las particiones V' que deben tener incluidos al menos un vértice, ya que de lo contrario, se debe buscar una nueva dirección para los vectores auxiliares de esas particiones sin elementos para generarlas nuevamente, como se describe en la ecuación 3.17. Si las particiones V son iguales a V', se asigna el valor de la variable $iguales := TRUE$, de lo contrario, se copian los vectores de A' a A.

[2]Un nuevo de valor para θ generado de manera aleatoria y sin repetición, es aquel que debe estar fuera del intervalo $[\theta_{ant} - 0{,}00001, \theta_{ant} + 0{,}00001]$ para valores de θ_{ant} generados en iteraciones anteriores. (El valor 0,00001 fue determinado de manera experimental)

PROCEDURE obtener_k_Particiones(V, k)

{

θ :=random$(0, \frac{2\pi}{k})$

A :=obtenerVectoresAuxiliaresIniciales(θ, x_d, y_d, k) *// procedimiento 3.3.1,* $A = \{\vec{a}_1, \cdots, \vec{a}_k\}$

W :=obtener_k_Subconjuntos(A, V, k) *// procedimiento 3.3.2*

FOR $j := 1$ **TO** k **DO** *// Se buscan particiones sin elementos*

IF $|V_j| = 0$ **THEN** *// si existe alguna partición sin elementos, se ajusta su vector asociado*

$m := indice(max[|V_{m'}|])$ *//* $1 \leq m' \leq k$

$\alpha_{\vec{a}_j} := \frac{\alpha_{\vec{a}_{mmitad}} + \alpha_{\vec{a}_{mmax}}}{2}$ *// ecuación 3.15*

$\theta_{\vec{a}_m} := \frac{\alpha_{\vec{a}_{mmin}} + \alpha_{\vec{a}_{mmitad}}}{2}$ *// ecuación 3.16*

V" $:= V_m$ *// Se guarda las partición* V_m *en* V"

$V_m := \emptyset$

V_j, V_m:=obtener_2_Particiones$(V", \vec{a}_m, \vec{a}_j)$ *// procedimiento 3.3.3*

$A' := A$

$iguales := FALSE$ *// Variable para condición de paro*

WHILE $iguales = FALSE$ **DO**

FOR j:=1 **TO** k **DO** *// Para los vectores auxiliares* A'

$\theta_{\vec{a'}_j} := \theta_{\vec{a'}_{jprom}}$ *// Ajustar vectores* $\vec{a'}_j$*, ecuación 3.14*

W' :=obtener_k_Subconjuntos(A', V, k) *// procedimiento 3.3.2*

FOR $j := 1$ **TO** k **DO** *// Se buscan particiones sin elementos*

IF $|V'_j| = 0$ **THEN**

$m := indice(max[|B'_{m'}|])$ *//* $1 \leq m' \leq k$

$\theta_{\vec{a'}_j} := \frac{\theta_{\vec{a'}_{mmitad}} + \theta_{\vec{a'}_{mmax}}}{2}$ *// ecuación 3.15*

$\alpha_{\vec{a'}_m} := \frac{\alpha_{\vec{a'}_{mmin}} + \alpha_{\vec{a'}_{mmitad}}}{2}$ *// ecuación 3.16*

V" $:= V'_m$ *// Se guarda las partición* V'_m *en* V"

$V'_m := \emptyset$

V'_j, V'_m:=obtener_2_Particiones$(B", \vec{a'}_M, \vec{a'}_j)$ *// procedimiento 3.3.3*

IF $W' = W$ **THEN**

$iguales := TRUE$

ELSE

$A := A'$

RETURN W *//* $W = \{V_1, \cdots, V_k\}$

}

FASE 2: obtener k recorridos

Los parámetros para este procedimiento son $W = \{V_1, \cdots, V_k\}$, que es el conjunto de las particiones, V, que es el conjunto de los vértices; y, k, que es el número de recorridos por construir.

Se inicializa al conjunto de los recorridos T con ningún recorrido y el $costo_T$ igual a cero. De manera iterativa hasta que se hayan terminado de construir los k recorridos, se construye al recorrido t^j equivalente a un poligono envolvente de la partición V_j y después se concluye el recorrido t^j con los vértices no considerados en el polígono envolvente. El recorrido resultante se agrega al conjunto T y su costo $costo_{t^j}$ se agrega a $costo_T$.

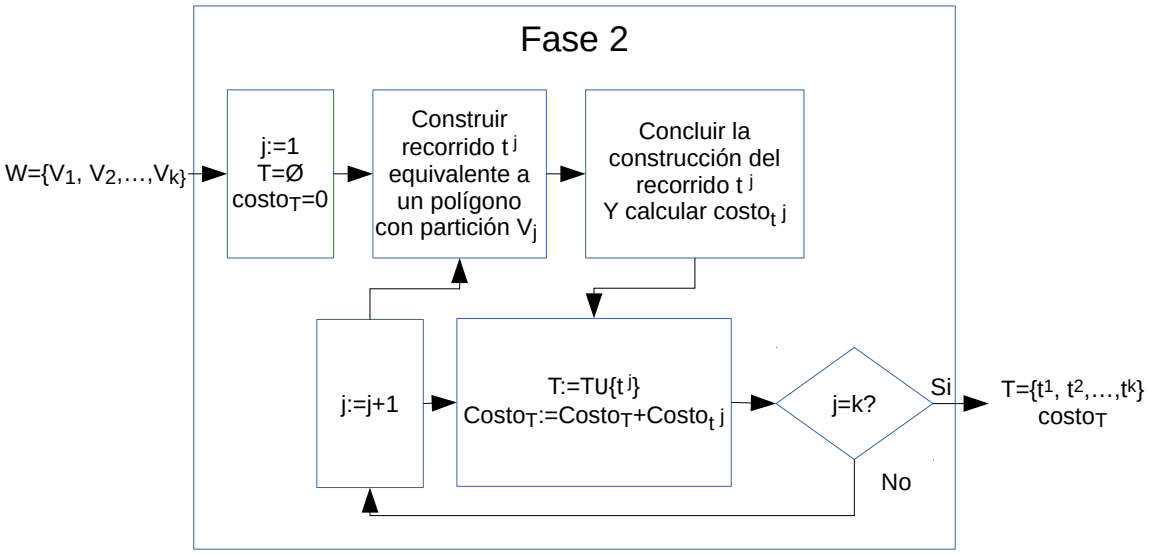

Figura 3.14: Diagrama de bloques de la fase 2 para obtener k recorridos.

PROCEDURE obtener_k_Recorridos(W, V, k)

```
{
  T := ∅
  costo_T := 0
  FOR j := 1 TO k DO
    t^j := construirPoligono(V_j, V) // procedimiento 3.3.4
    t^j, costo_tj := construirRecorridos(t^j, B_j, V) // procedimiento 3.3.5
    T := T ∪ {t^j}
    costo_T := costo_T + costo_tj
  RETURN T, costo_T
}
```

3.5. Descripción del algoritmo para resolver un MTSP

Los parámetros para este procedimiento son V y k, que son el conjunto de los vértices y el número de recorridos dados en la instancia.

El algoritmo inicializa la variable $costo_{min}$ igual a un valor muy grande. De manera iterativa se obtiene el conjunto de las particiones W, el conjunto de recorridos T y el costo de ese recorrido $costo_T$; y si el $costo_T$ es menor al $costo_{min}$, entonces se actualiza el valor de $costo_{min}$ igual a $costo_T$ y se guarda el recorrido T en la variable T_{min}, hasta que se hayan realizado $maxIters$ iteraciones.

PROCEDURE mTsp(V, k)

```
{
  costo_min := ∞
  FOR numIters := 0 TO maxIters DO
    W :=obtener_k_Particiones(k, V) // FASE 1, W = {V_1, ..., V_k}
    T, costo_T :=obtener_k_Recorridos(W, V, k) //FASE 2, T = {t^1, ..., t^k}
    IF costo_T < costo_min THEN //Si se obtuvo un menor costo en esta iteración
      costo_min := costo_T //Se guarda en costo_min y T_min el mejor costo
      T_min := T //y su solución respectivamente
  RETURN  costo_min, T_min
}
```

Capítulo 4

Resultados

El algoritmo en 2 fases desarrollado en este proyecto fue codificado en el lenguaje C++ bajo el paradigma orientado a objetos, ejecutado en una laptop Lenovo thinkpad T540p con las siguientes especificaciones: procesador intel core i7-4710MQ CPU @ 2.5GHz, 8GB en RAM y sistema operativo GNU/Linux Debian 9.6.0 Stretch amd64 (instalación básica).

Instancia					Resultado	Algoritmo 2 FASES			
Nombre	$\|V\|+1$	k	m_{min}	m_{max}	BKS	$c(T)$	m'_{min}	m'_{max}	error
pr76	76	5	1	20	132784	133232	1	52	0.34 %
pr152	152	5	1	40	105205	135160	8	53	28.47 %
pr226	226	5	1	50	148051	118549	4	77	**19.93 %**
pr299	299	5	1	70	72949	65721.6	7	122	**9.91 %**
pr439	439	5	1	100	143785	139067	51	184	**3.28 %**
pr1002	1002	5	1	220	334351	314609	5	485	**5.90 %**
eil51	51	2	23	27	442.32	450.20	17	33	1.78 %
eil51	51	3	15	20	464.11	465.56	10	30	0.31 %
eil51	51	5	7	12	529.70	519.21	2	18	**1.98 %**
eil51	51	7	5	10	605.21	582.63	2	14	**3.73 %**
berlin52	52	2	10	41	7753.89	7965.45	16	59	2.73 %
berlin52	52	3	10	27	8106.85	8426.09	6	53	3.94 %
berlin52	52	5	6	17	9126.33	9648.72	2	39	5.72 %
berlin52	52	7	4	17	9870.02	10698.30	2	25	8.39 %
eil76	76	2	36	39	558.59	591.91	15	36	5.97 %
eil76	76	3	21	30	579.30	591.48	10	30	2.10 %
eil76	76	5	12	17	680.67	627.35	5	24	**7.83 %**
eil76	76	7	7	15	759.90	691.78	2	17	**8.96 %**
rat99	99	2	46	52	1350.73	1352.42	20	78	0.13 %
rat99	99	3	27	36	1519.49	1498.51	17	41	**1.38 %**
rat99	99	5	13	30	1855.83	1822.33	5	41	**1.81 %**
rat99	99	7	9	22	2291.82	2379.46	3	22	3.82 %

Tabla 4.1: Comparación entre los Resultados Mejor Conocidos para el *MTSP Bounded*, con los resultados obtenidos del algoritmo en 2 fases para el MTSP.

Símbolo	Descripción del símbolo
Nombre	Nombre de la instancia
$\|V\|+1$	es el número de vértices, incluido el depósito, que se encuentran en el conjunto de los vértices de la instancia que define al MTSP.
k	número de recorridos que debe tener la solución.
m_{min}	es el número de vértices mínimo que el algoritmo debe incluir en un recorrido sin considerar al depósito.
m_{max}	es el número de vértices máximo que el algoritmo debe incluir en un recorrido sin considerar al depósito.
BKS	Resultados Mejores Conocidos obtenidos de la literatura.
$c(T)$	Costo de la solución que el algoritmo 2 fases obtuvo.
m'_{min}	número de vértices que tiene el recorrido con menor cantidad de vértices.
m'_{max}	número de vértices que tiene el recorrido con mayor cantidad de vértices.
error	porcentaje de error entre BKS y c(T).

Tabla 4.2: La simbología del encabezado de la tabla 4.1.

Como ya se ha mencionado en el capítulo 2.6, no fueron encontrados en la literatura resultados para el problema MTSP con las mismas restriccciones que se fueron planteadas en este proyecto. El más parecido a nuestro diseño es el *MTSP Bounded*; con el cuál hemos hecho nuestras comparaciones, presentadas en la tabla 4.1.

De las 22 instancias, nuestro algoritmo logró obtener un costo menor en 10 instancias: pr226, pr299, pr439, pr1002, eil51-5, eil51-7, eil76-5, eil76-7, rat99-3 y rat99-5; con la observación de que el intervalo que la instancia para el MTSP Bounded establece como cantidad mínima m_{min} y máxima m_{max}, no coinciden con los intervalos m'_{min} y m'_{max} que nuestro algoritmo obtiene.

En la última columna de la tabla 4.1, se presenta el cálculo del error (ecuación 4.1), el cuál indica la diferencia en términos porcentuales que existe entre la Solución Mejor Conocida y la Solución que el algoritmo en 2 fases obtuvo.

$$error = \left| \frac{c(T) - BKS}{BKS} \right| 100\,\% \tag{4.1}$$

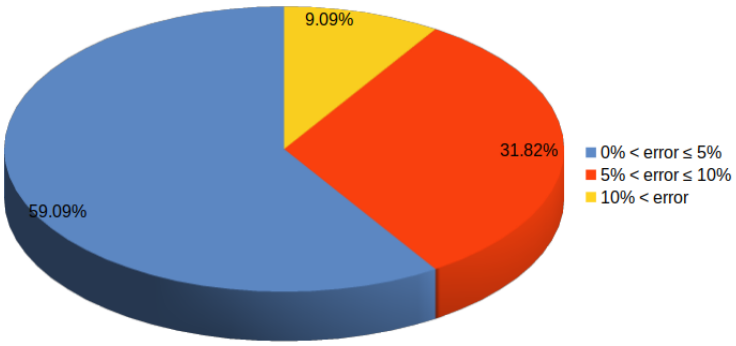

Figura 4.1: Porcentaje de instancias según el error calculado en la tabla 4.1

En la figura 4.1, se muestra en una gráfica el porcentaje de instancias que obtuvieron un error entre 0 % y 5 % (13 instancias), un error entre 5 % y 10 % (7 instancias); y un error mayor al 10 % (2 instancias).

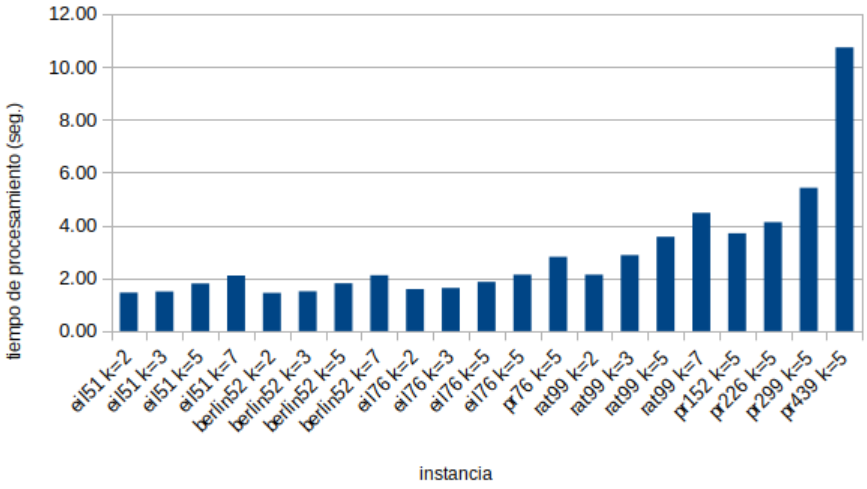

Figura 4.2: Tiempos de ejecución para la solución de las instancias

En la figura 4.2, se muestra la gráfica de los tiempos de procesamiento de 21 instancias, con excepción de la instancia pr1002 que tuvo un tiempo de ejecución de 137.75 seg. En ella se pueden observar cosas:

- El tiempo de ejecución aumenta si el número de vendedores aumenta: Esto podemos apreciarlo para mismas instancias con cantidad de vendedores diferentes, tal como eil51, berlin52, eil76 y rat99.
- El tiempo de ejecución aumenta si el número de vértices aumenta: Análogamente al punto anterior, podemos observar que mientras más vértices tiene una instancia, mayor tiempo de procesamiento se tomó el algoritmo para resolver el problema.

Las soluciones que fueron construidas por el algoritmo 2 fases, pueden clasificarse en 3 formas, con base en la ubicación del depósito:

Depósito ubicado cerca del borde

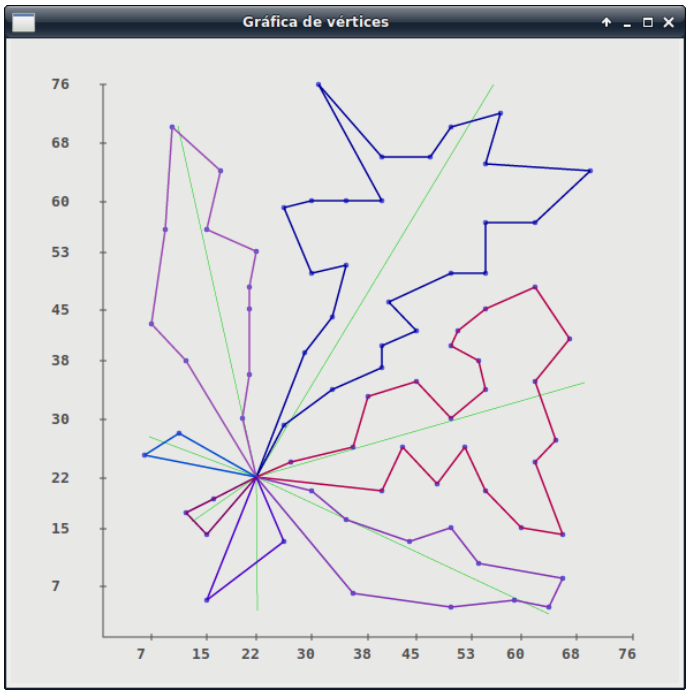

Figura 4.3: Instancia eil76, 7 recorridos, $c(T) = 691{,}78$ y tiempo de ejecución 2.13 seg.

Cuando el depósito se encuentra muy cerca del borde, con respecto a la ubicación de los vértices que pertenecen al conjunto de los vértices, tenemos como resultado que la cantidad de vértices no están distribuidos uniformemente en todos los recorridos. Esto se debe a que el algoritmo genera particiones del conjunto V hacia todas las direcciones y, no existe una restricción que indique que los recorridos deben tener un a cantidad uniforme de vértices.

En el ejemplo de la figura 4.3, puede verse que en la solución de nuestro algoritmo: 3 recorridos incluyen 2 ó 3 vértices, y en los otros 4 recorridos tienen más de 10 vértices.

Depósito ubicado sobre el borde

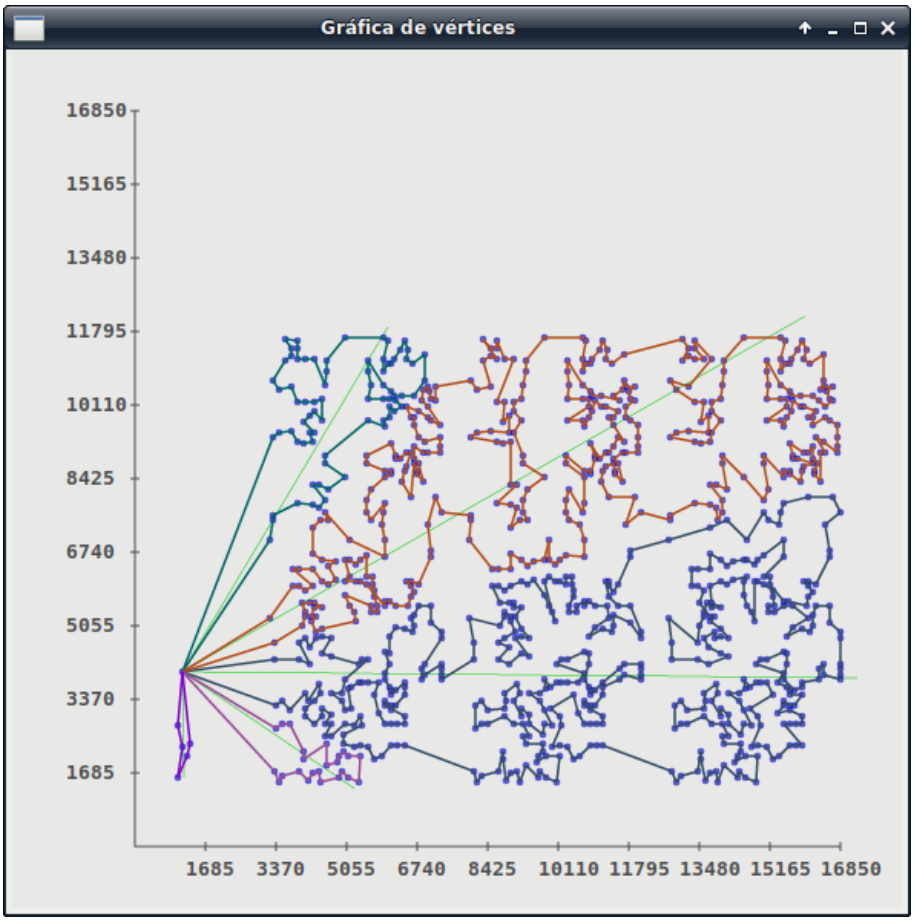

Figura 4.4: Instancia pr1002, 5 recorridos, $c(T) = 315869$ y tiempo de ejecución 133.80 seg.

Cuando el depósito se encuentra sobre el borde, con respecto a la ubicación de los vértices que pertenecen al conjunto de los vértices, la solución tiene recorridos que se asemejan a un abanico. En algunas ocasiones los recorridos que se encuentran a los extremos de ese abanico tienen una cantidad menor de vértices con respecto a los recorridos que se encuentran al centro.

En el ejemplo de la figura 4.4, puede verse que en la solución de nuestro algoritmo: 2 recorridos, unicados al centro del abanico, tienen más del 50 % de los vértices, mientras que los otros 3 recorridos, ubicados en ambos extremos del abanico, tienen un menor porcentaje de vértices.

Depósito ubicado en el centro

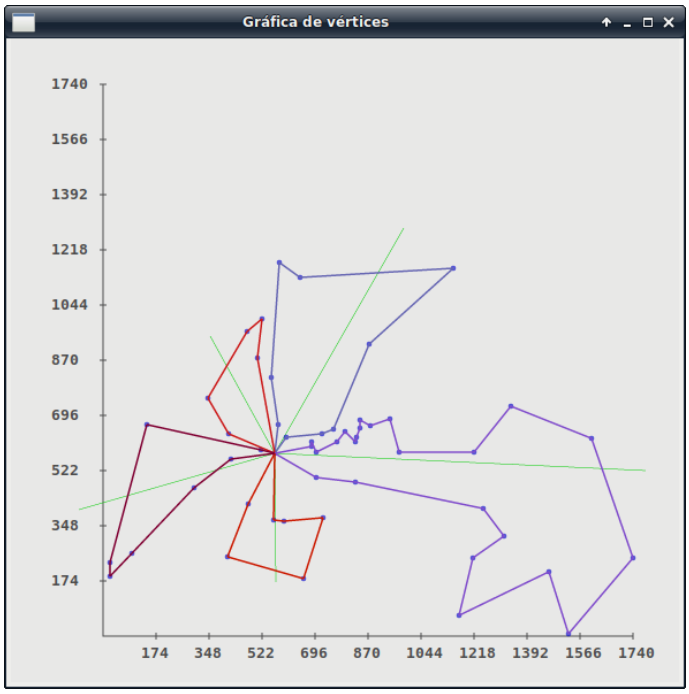

Figura 4.5: Instancia berlin52, 5 recorridos, $c(T) = 9648{,}72$ y tiempo de ejecución 1.79 seg.

Cuando el depósito se encuentra relativamente cerca del centro, con referencia a la ubicación de los vértices que pertenecen al conjunto de los vértices, los resultados obtenidos tienen una mejor distribución de los vértices en los recorridos con respecto a los otros dos casos mencionados anteriormente (4.5).

En el ejemplo de la figura 4.5, puede verse que en la solución de nuestro algoritmo tiene mejor la distribución de los vértices. Debido a que el depósito se encuentra un poco a la izquierda, con respecto al centro de todos los vértices, es que el recorrido que está a hacia la derecha tiene más vértices que los demás.

En el apéndice A, se muestran las gráficas de las soluciones que fueron construidas por nuestro algorimo.

Conclusiones y trabajo futuro

El algoritmo 2 fases diseñado en este proyecto de investigación, genera y evalúa soluciones factibles para el MTSP.

La fase 1, la partición de V, se considera la parte más importante del algoritmo, ya que las soluciones resultantes dependen mucho de la calidad de las particiones que aquí se generan. En ella se observa que la ubicación del depósito influye en las soluciones resultantes, ya que tienen una mejor distribución de los vértices de V en los k recorridos, si el depósito está ubicado cerca del centro con respecto a los demás vértices.

La fase 2, la construcción de recorridos, tiene una complejidad temporal $O(n^3)$, con soluciones que tienen un margen de error entre 6 % y 24 %, sobre los mejores resultados conocidos reportados en [TSPLIB].

El tiempo de procesamiento del algoritmo aumenta si el número de vértices aumenta y/o, el número de vendedores aumenta.

Los resultados para 22 instancias disponibles en la biblioteca [TSPLIB], las cuales se utilizan para comparar soluciones de algoritmos que resuelven problemas del MTSP Bounded, son los siguientes:

- El 45 % de los costos obtenidos, son menores que los costos reportados en la literatura para el MTSP Bounded.
- El 59.09 % de los costos obtenidos, tiene una diferencia menor al 5 % con respecto a los costos reportados en la literatura para el MTSP Bounded.

Las propuestas sugeridas para continuar con esta investigación son:

- Agregar la restricción que establece una cantidad mínima,m_{min}, y máxima, m_{max}, de vértices que cada recorrido debe incluir, en el diseño del algoritmo.
- Generar más resultados para instancias del [TSPLIB] para futuras pruebas de este algoritmo.
- Mejorar el algoritmo que particiona el conjunto de los vértices en la fase 1, para que los subconjuntos resultantes sean de mejor calidad.
- Mejorar el algoritmo que construye los recorridos en la fase 2, con la finalidad de reducir su complejidad temporal de $O(n^3)$ a $O(n^2)$.
- Agregar un algoritmo de búsqueda local que mejore las soluciones que el algoritmo obtiene.

Apéndice A

Gráficas de los resultados obtenidos por el algoritmo 2 fases

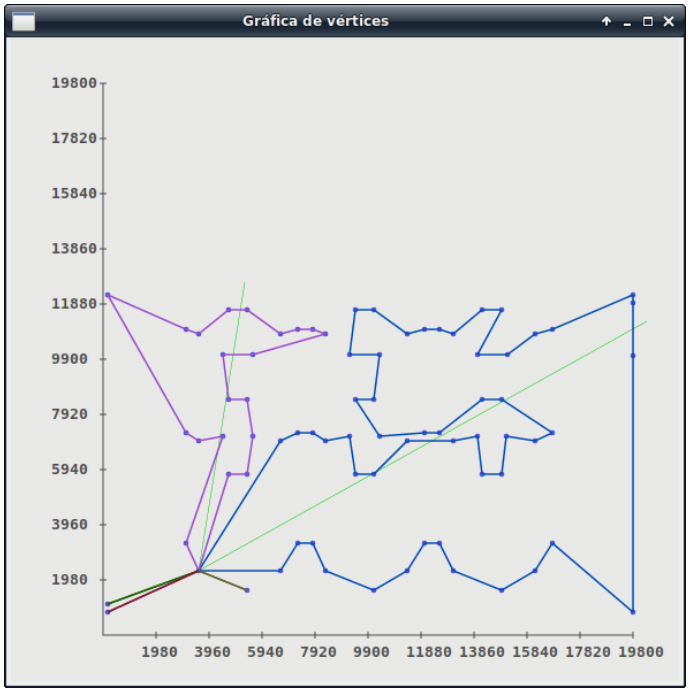

Figura A.1: Instancia pr76, 5 recorridos, $c(T) = 133232$ y tiempo de ejecución 2.91 seg.

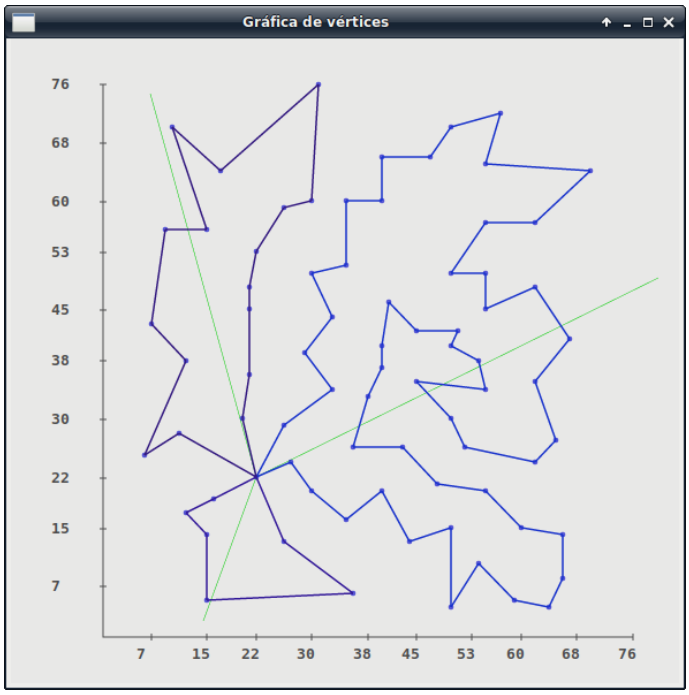

Figura A.2: Instancia eil76, 3 recorridos, $c(T) = 591{,}48$ y tiempo de ejecución 1.62 seg.

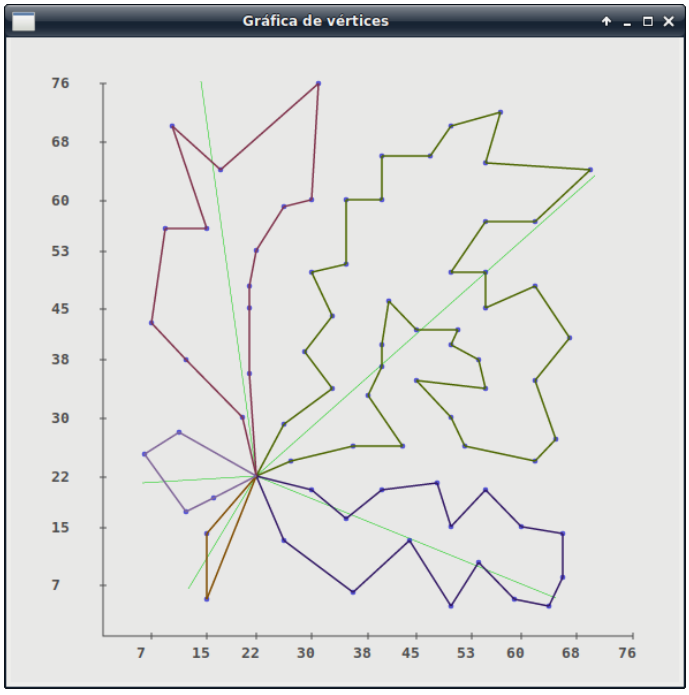

Figura A.3: Instancia eil76, 5 recorridos, $c(T) = 634$ y tiempo de ejecución 1.84 seg.

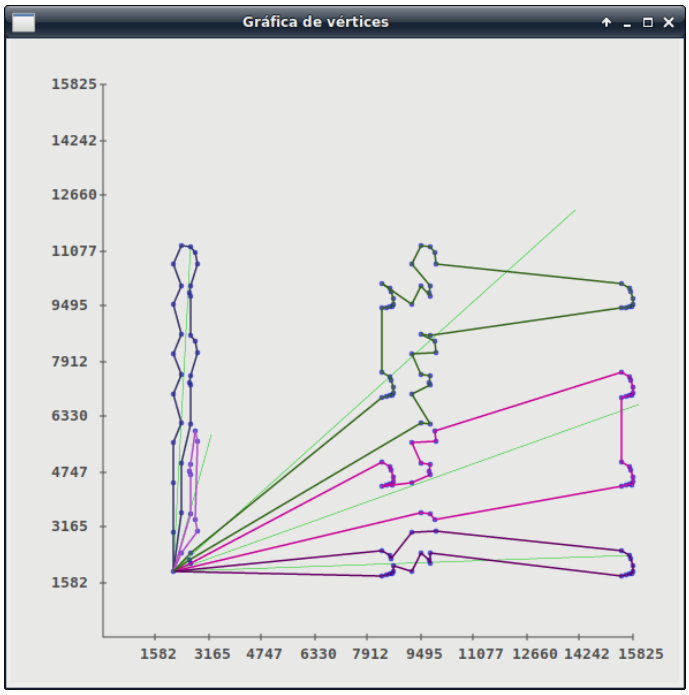

Figura A.4: Instancia pr152, 5 recorridos, $c(T) = 135160$ y tiempo de ejecución 3.70 seg.

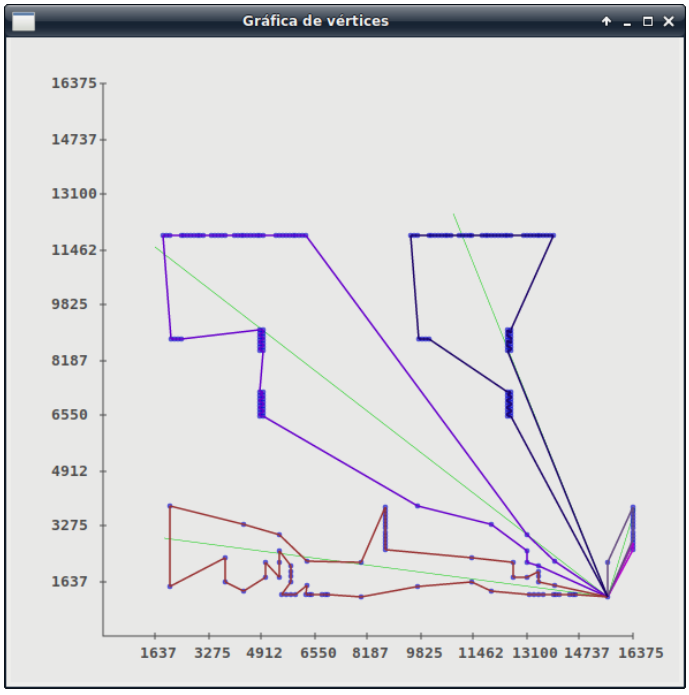

Figura A.5: Instancia pr226, 5 recorridos, $c(T) = 118366$ y tiempo de ejecución 4.12 seg.

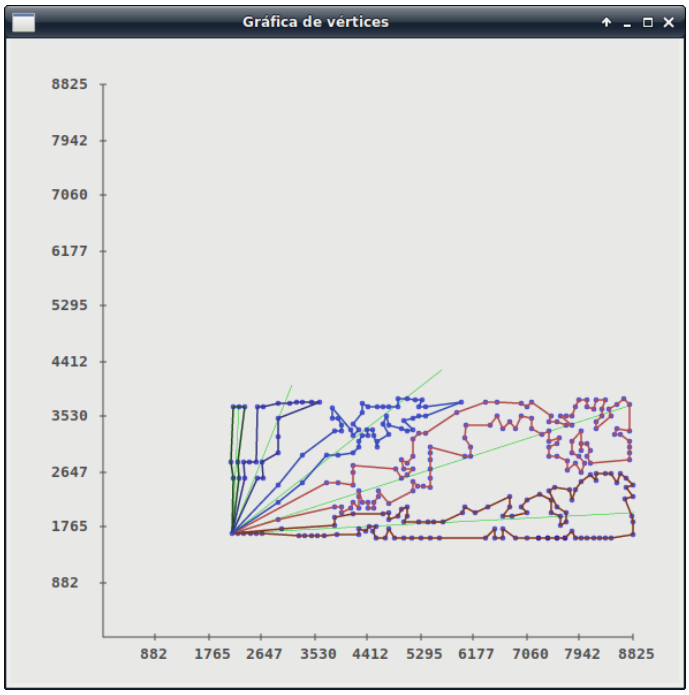

Figura A.6: Instancia pr299, 5 recorridos, $c(T) = 65695{,}9$ y tiempo de ejecución 5.31 seg.

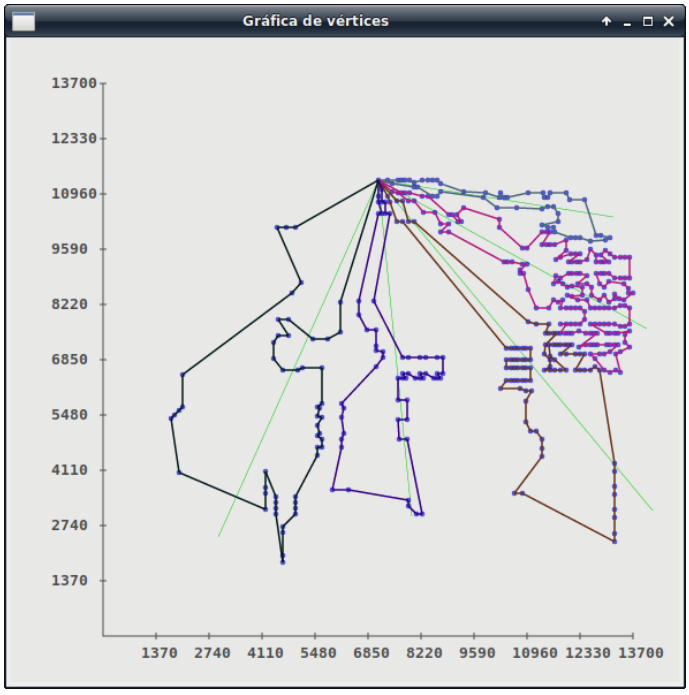

Figura A.7: Instancia pr439, 5 recorridos, $c(T) = 138463$ y tiempo de ejecución 10.15 seg.

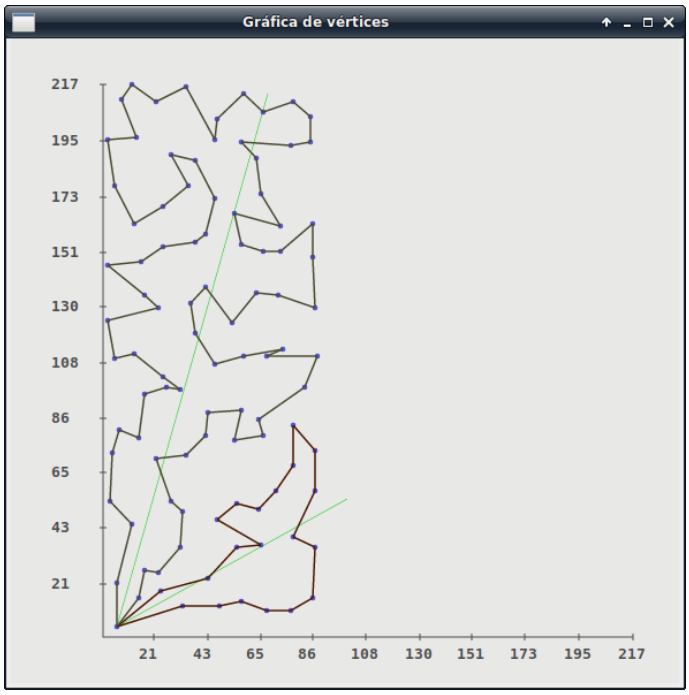

Figura A.8: Instancia rat99, 2 recorridos, $c(T) = 1352{,}42$ y tiempo de ejecución 2.15 seg.

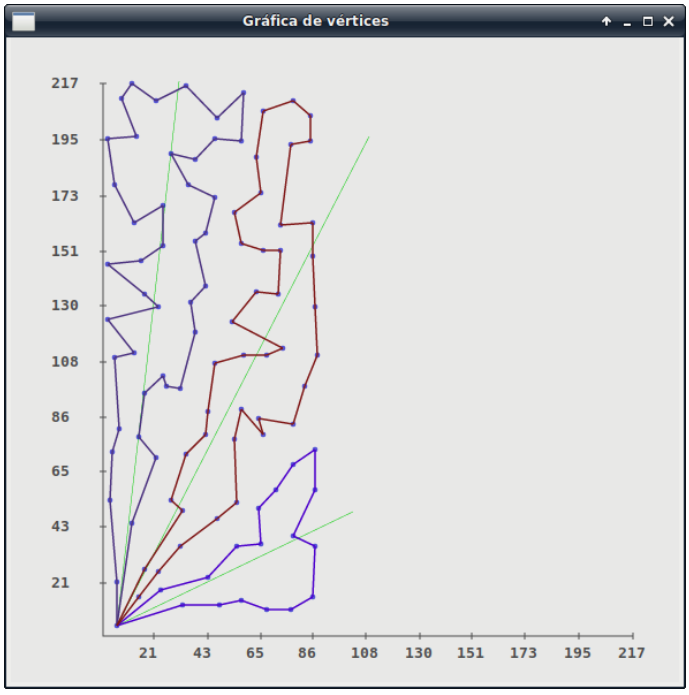

Figura A.9: Instancia rat99, 3 recorridos, $c(T) = 1498{,}51$ y tiempo de ejecución 2.85 seg.

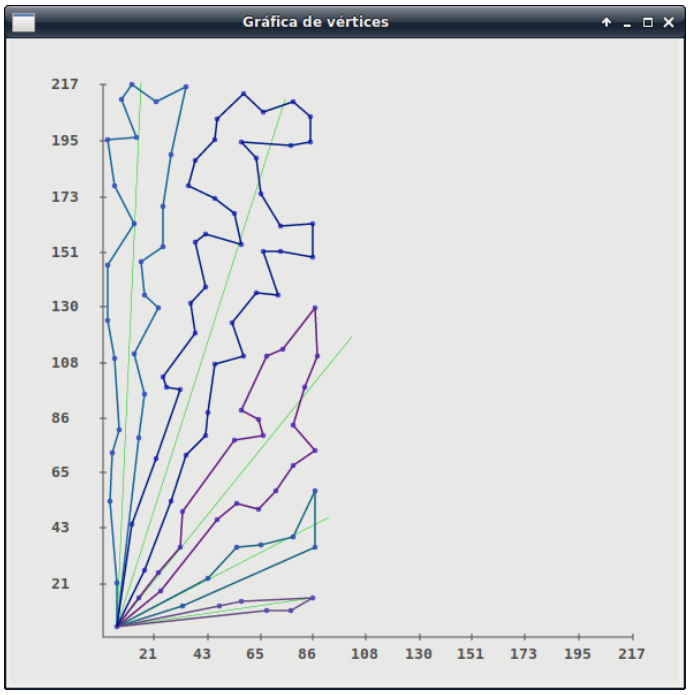

Figura A.10: Instancia rat99, 5 recorridos, $c(T)$1827,27 y tiempo de ejecución 3.56 seg.

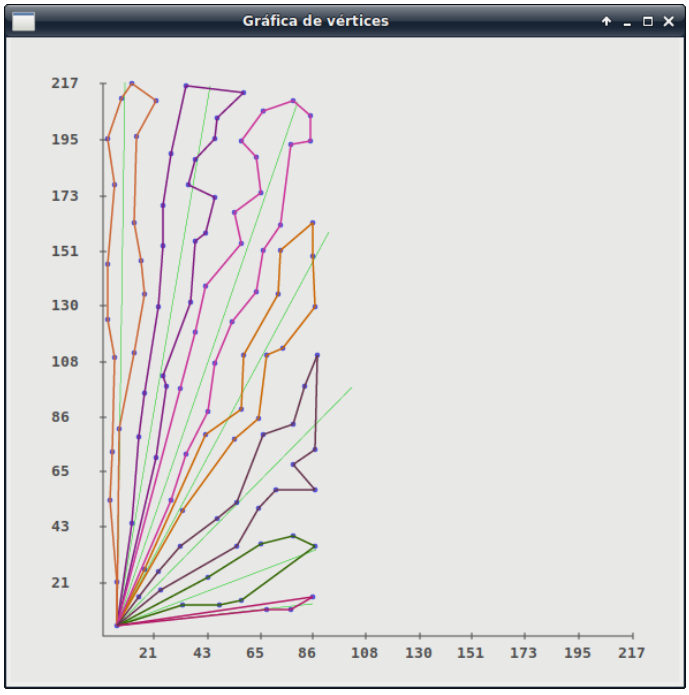

Figura A.11: Instancia rat99, 7 recorridos, $c(T) = 2379{,}46$ y tiempo de ejecución 4.42 seg.

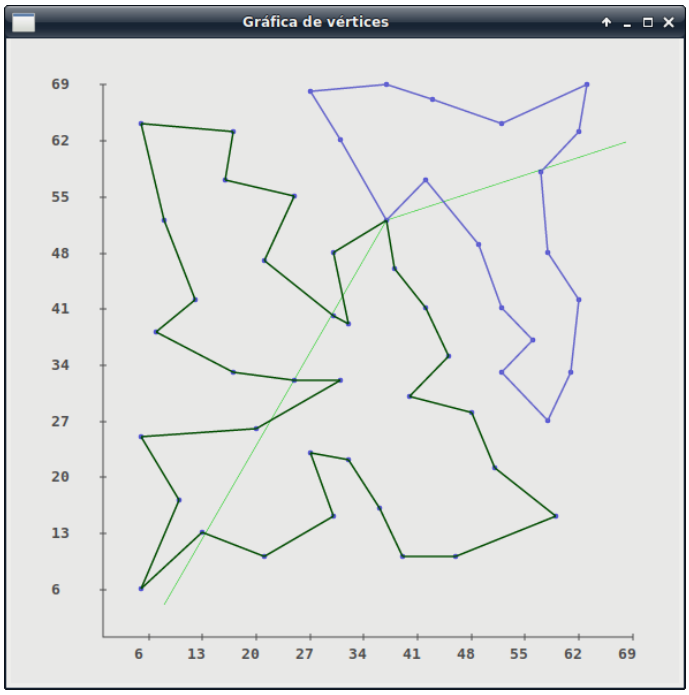

Figura A.12: Instancia eil51, 2 recorridos, $c(T) = 450{,}20$ y tiempo de ejecución 1.34 seg.

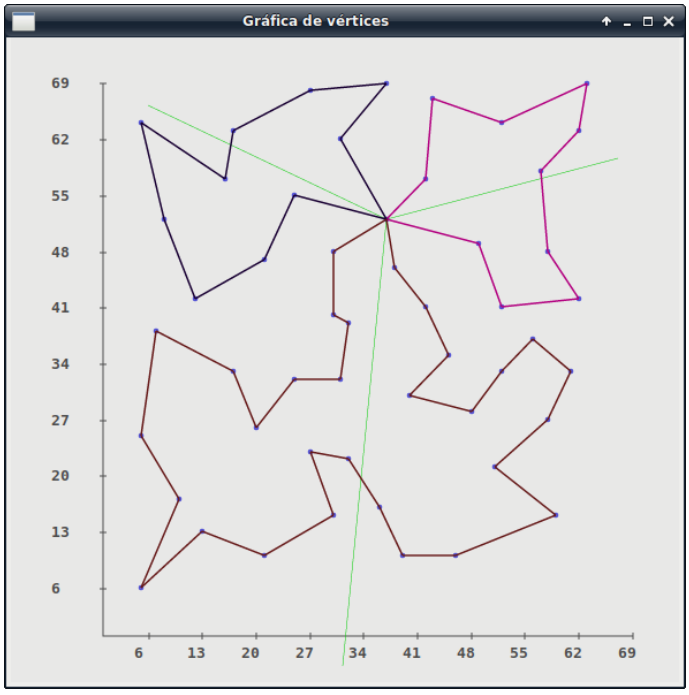

Figura A.13: Instancia eil51, 3 recorridos, $c(T) = 465{,}56$ y tiempo de ejecución 1.49 seg.

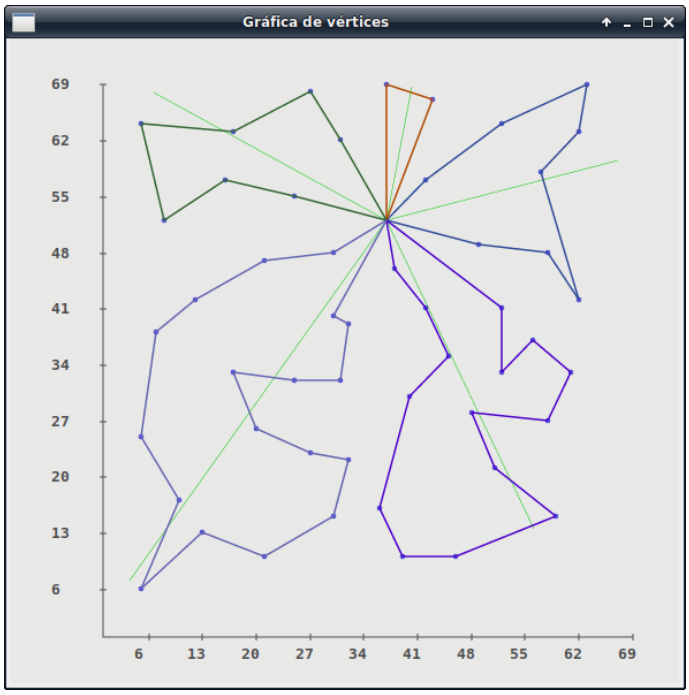

Figura A.14: Instancia eil51, 5 recorridos, $c(T) = 519{,}21$ y tiempo de ejecución 1.79 seg.

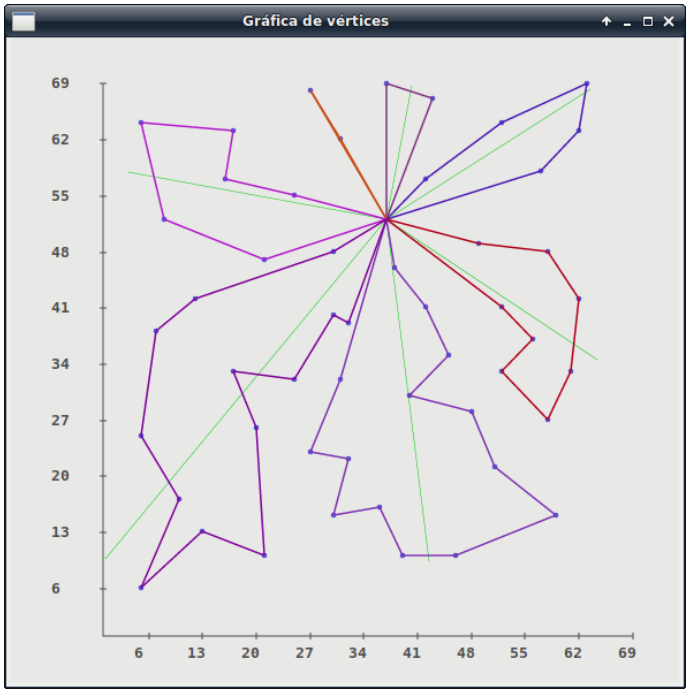

Figura A.15: Instancia eil51, 7 recorridos, $c(T) = 582{,}63$ y tiempo de ejecución 2.09 seg.

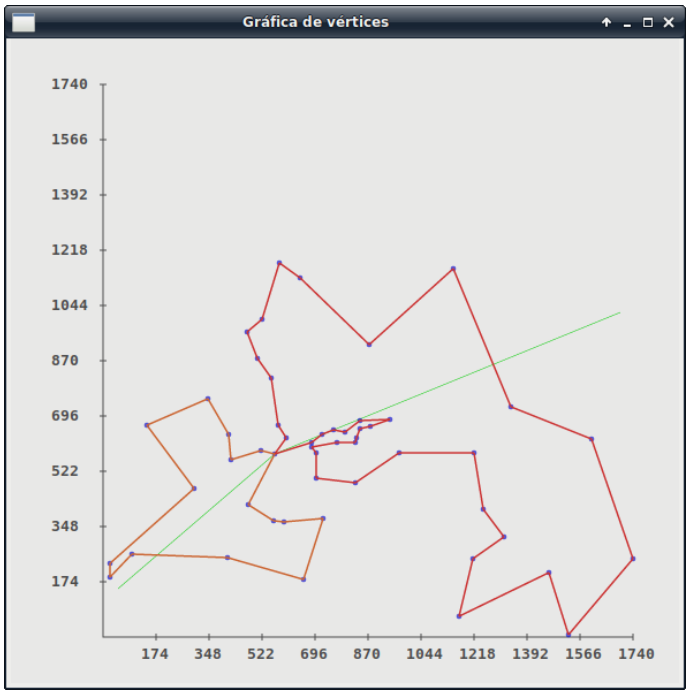

Figura A.16: Instancia berlin52, 2 recorridos, $c(T) = 7965{,}45$ y tiempo de ejecución 1.36 seg.

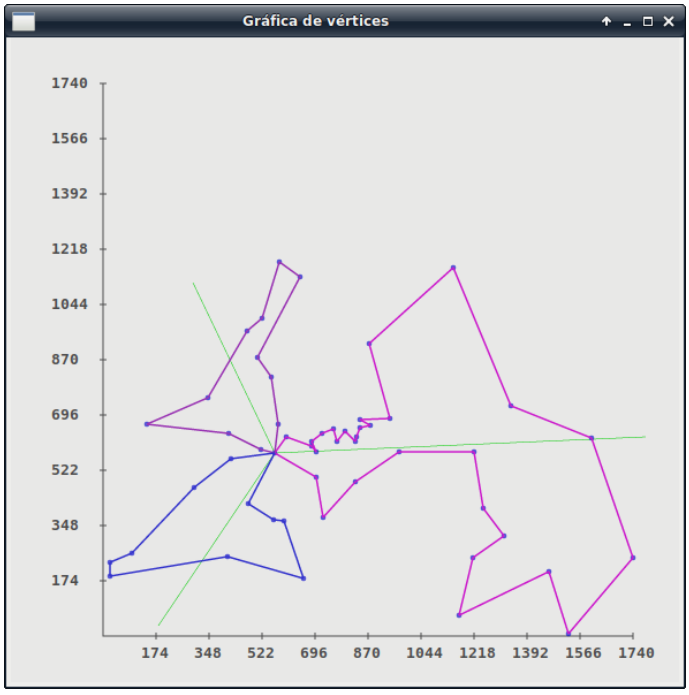

Figura A.17: Instancia berlin52, 3 recorridos, $c(T) = 8426{,}09$ y tiempo de ejecución 1.48 seg.

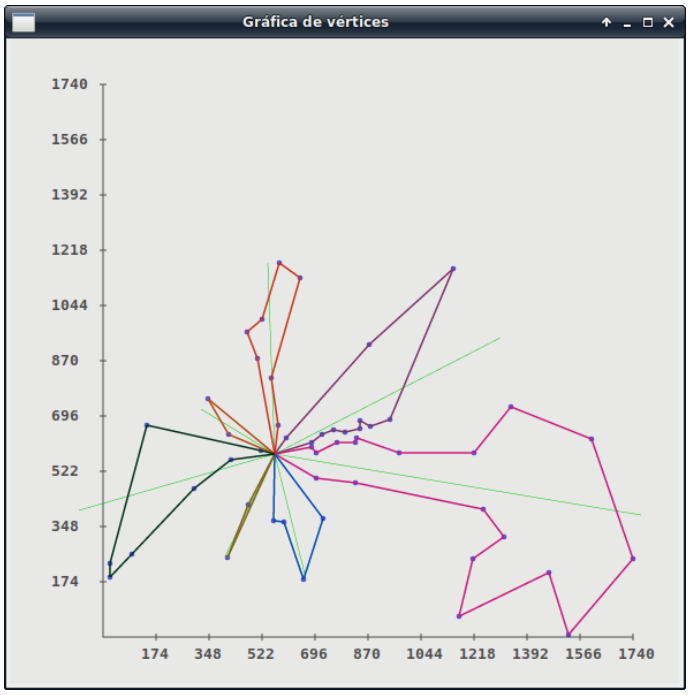

Figura A.18: Instancia berlin52, 7 recorridos, $c(T) = 10698{,}3$ y tiempo de ejecución 2.09 seg.

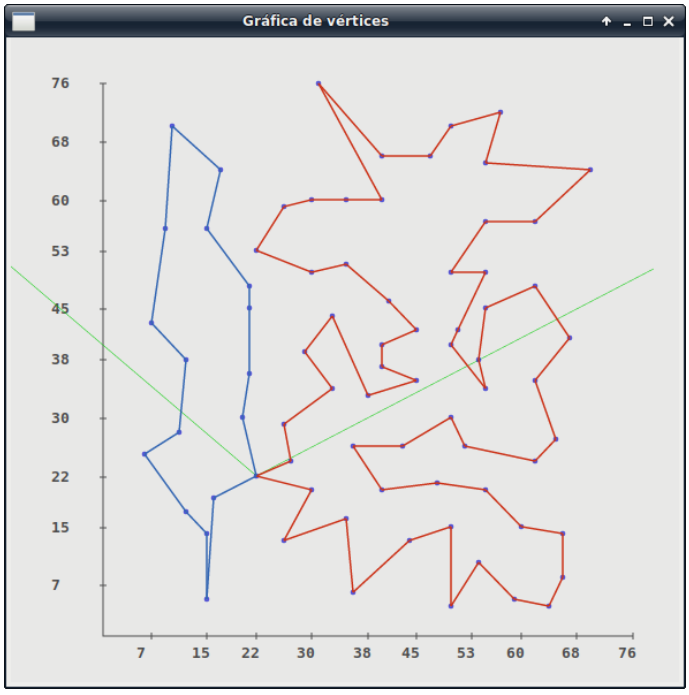

Figura A.19: Instancia eil76, 2 recorridos, $c(T) = 591{,}91$ y tiempo de ejecución 1.58 seg.

Apéndice B

Instancia usada en capítulo 2.5

$$|V| = 31$$

$$n = |V - \{v_d\}| = 30$$

$$x_{v_d} = 30 \ , \ y_{v_d} = 41$$

Tabla B.1: Instancia usada en el capítulo 2.5

v_i	x_{v_i}	y_{v_i}		v_i	x_{v_i}	y_{v_i}		v_i	x_{v_i}	y_{v_i}
1	42	42		11	37	53		21	15	13
2	40	30		12	49	50		22	20	26
3	30	49		13	52	42		23	5	25
4	27	23		14	42	58		24	7	38
5	17	33		15	31	63		25	12	43
6	21	48		16	43	68		26	8	53
7	31	32		17	53	65		27	17	64
8	36	16		18	58	59		28	27	69
9	52	33		19	58	49		29	63	43
10	51	21		20	58	27		30	16	58

Bibliografía

[Bektas 2006] Bektas Tolga *The multiple traveling salesman problem: an overview of formulations and solution procedures.* Omega, 34(3) (2006), p. 209–219.

[Benavent y Martínez 2013] , Benavent Enrique y Martínez Antonio, *Multi-depot Multiple TSP: a polyhedral study and computational results.* Annals of Operations Research, 2013, vol. 207, no 1, p. 7-25.

[Brumitt y Stentz 1996] Brumitt Barry L. y Stentz Anthony *Dynamic mission planning for multiple mobile robots.* Proceedings of IEEE International Conference on Robotics and Automation. IEEE, 1996. p. 2396-2401.

[Cordeau *et al.* 2005] Cordeau Jean-François, Gendreau Michel, Hertz Alain, Laporte Gilbert y Sormany Jean-Sylvain, *New heuristics for the vehicle routing problem.* In Logistics systems: design and optimization. Springer, Boston, MA, 2005. p. 279-297.

[Cuevas *et al.*, 2016] Cuevas Jiménez Erick V., Osuna Enciso José V., Oliva Navarro José A. y Díaz Cortés Margarita A., *Optimización: algoritmos programados con MATLAB.* editorial Alfaomega, primera edición, impreso en México, 2016.

[Dantzig y Ramser 1929] Dantzig George B. y Ramser John H., *The truck dispatching problem.* Management science, 1959, vol. 6, no 1, p. 80-91.

[Desrochers *et al.* 1992] Desrochers Martin, Desrosiers Jacques y Solomon Marius *A new optimization algorithm for the vehicle routing problem with time windows.* Operations research, 1992, vol. 40, no 2, p. 342-354.

[Gendreau *et al.* 2008] Gendreau Michel, Potvin Jean-Yves, Bräysy Olli, Geir Hasle y Lokketangen Arne *Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography.* The vehicle routing problem: latest advances and new challenges. Springer, Boston, MA, 2008. p. 143-169.

[Gilbert y Hofstra 1992] Gilbert, Kenneth C. y Hoftra, Ruth B., *A new multiperiod multiple traveling salesman problem with heuristic and application to a scheduling problem.* Decision Sciences, 1992, vol. 23, no 1, p. 250-259.

[Gorenstein 1970] Gorenstein, Samuel, *Printing press scheduling for multi-edition periodicals.* Management Science, 1970, vol. 16, no 6, p. B-373–B-383.

[Harrath *et al.* 2019] Harrath Youssef, Salman Abdul Fattah, Alqaddoumi Abdulla, Hesham Hasan y Radhi Ahmed, *A novel hybrid approach for solving the multiple traveling salesmen problem.* Arab Journal of Basic and Applied Sciences, 2019, vol. 26, no 1, p. 103-112.

[Horowitz y Sahni 1978] Horowitz Ellis y Sahni Sartaj, *Computer Algorithms.* editorial Computer Science Press, impreso en los Estados Unidos, 1978.

[Hosseinabadi *et al.* 2014] Hosseinabadi Ali A., Kardgar Maryam, Shamshirband Shojafar y Abraham Ajith, *GELS-GA: hybrid metaheuristic algorithm for solving multiple travelling salesman problem.* 14th International Conference on Intelligent Systems Design and Applications. IEEE, 2014. p. 76-81.

[IBM 2019] IBM *IBM Knowledge Center* consultado en: `https://www.ibm.com/support/knowledgecenter/es/SSSA5P_12.5.0/ilog.odms.cplex.help/CPLEX/UsrMan/topics/preface/preface_title_synopsis.html`, 16 de septiembre de 2019.

[Junjie and Dingwei 2006] Junjie Pan y Dingwei Wang, *An ant colony optimization algorithm for multiple travelling salesman problem.* First International Conference on Innovative Computing, Information and Control-Volume I (ICICIC'06). IEEE, 2006. p. 210-213.

[Jünger *et al.* 1995] Jünger Michael, Reinelt Gerhard y Rinaldi Giovanni, *The traveling salesman problem.* Handbooks in operations research and management science, 1995, vol. 7, p. 225-330.

[Kitjacharoenchai *et al.* 2019] Kitjacharoenchai Patchara, Ventresca M, Moshref-Javadi M., Lee S., Tanchoco J. y Brunese P.A., *Multiple traveling salesman problem with drones: Mathematical model and heuristic approach.* Computers & Industrial Engineering, 2019, vol. 129, p. 14-30

[Laporte *et al.* 1992] Laporte Gilbert, Nobert Yves y Taillefer Serge, *A branch-and-bound algorithm for the asymmetrical distance-constrained vehicle routing problem.* Mathematical Modelling, 1987, vol. 9, no 12, p. 857-868.

[Laporte 1995] Laporte Gilbert, *The vehicle routing problem: An overview of exact and approximate algorithms.* European journal of operational research, 1992, vol. 59, no 3, p. 345-358.

[Lenstra y Kan 1975] Lenstra Jan Karel, Kan AHG Rinnooy, *Some simple applications of the travelling salesman problem.* Journal of the operational Research Society, 1975, vol. 26, no 4, p. 717-733.

[Lin y Kernighan 1973] Lin Shen, Kernighan Brian W., *An effective heuristic algorithm for the traveling-salesman problem.* Operations research, 1973, vol. 21, no 2, p. 498-516.

[Lo *et al.* 2018] , Lo Kin-Ming, Yi Wei-Ying, Wong Pak-Kan, Leung Kwong-Sak, Leung Yee y Mak Sui-Tung, *A genetic algorithm with new local operators for multiple traveling salesman problems.* International Journal of Computational Intelligence Systems, 2018, vol. 11, no 1, p. 692-705.

[Necula *et al.* 2015] , Necula Racula, Breaban Mihaela y Raschip Madalina, *Performance evaluation of ant colony systems for the single-depot multiple traveling salesman problem.* International Conference on Hybrid Artificial Intelligence Systems. Springer, Cham, 2015. p. 257-268.

[Pecin *et al.* 2017] , Pecin Diego, Pessoa Artur, Poggi Marcus y Uchoa Eduardo, *Improved branch-cut-and-price for capacitated vehicle routing.* Mathematical Programming Computation , 2017, vol. 9, no 1, p. 61-100.

[Rosen 2004] Rosen Kenneth H., *Matemática Discreta y sus aplicaciones.* editorial McGrawHill, quinta edición, impreso en Colombia, 2004.

[Rostami *et al.* 2015] Rostami Ali Shokouhi, Mohanna Farahnaz, Keshavarz y Hosseinabadi Ali Asghat Rahmani, *Solving multiple traveling salesman problem using the gravitational emulation local search algorithm.* Applied Mathematics & Information Sciences, 2015, vol. 9, no 2, p. 1-11.

[Saleh y Chelouah 2004] , Saleh Hussain y Chelouah Rachid, *The design of the global navigation satellite system surveying networks using genetic algorithms.* Engineering Applications of Artificial Intelligence, 2004, vol. 17, no 1, p. 111-122.

[Simon y Newell 1958] , Simon Herbert y Newell Allen, *Heuristic problem solving: The next advance in operations research.* Operations research, 1958, vol. 6, no 1, p. 1-10.

[Singh 2016] , Singh Amarbir, *A Review on Algorithms Used to Solve Multiple Travelling Salesman Problem.* International Research Journal of Engineering and Technology, 2016. p.598-603

[Sörensen 2013] , Sörensen Kenneth, *Metaheuristics—the metaphor exposed.* International Transactions in Operational Research, 2015, vol. 22, no 1, p. 3-18.

[Sundar y Rathinam 2016] , Sundar Kaarthik y Rathinam Sivakumar, *Generalized multiple depot traveling salesmen problem—Polyhedral study and exact algorithm.* Computers & Operations Research, 2016, vol. 70, p. 39-55.

[Svestka y Huckfeldt 1973] , Svestka, Joseph y Huckfeldt, Vaughn E., *Computational experience with an m-salesman traveling salesman algorithm.* Management Science, 1973, vol. 19, no 7, p.790-799.

[Tang *et al.* 2000] Tang Lixin, Liu Jiyin, Rong Aiying y Zihou Yang, *A multiple traveling salesman problem model for hot rolling scheduling in Shanghai Baoshan Iron & Steel Complex.* European Journal of Operational Research, 2000, vol. 124, no 2, p. 267-282.

[Thenepalle y Singamsetty 2019] Thenepalle Jayanth y Singamsetty Purusotham, *An open close multiple travelling salesman problem with single depot.* Decision Science Letters, 2019, vol. 8, no 2, p. 121-136.

[TSPLIB] Universität Heidelberg, Institut für Informatik, Gerhard Reinelt, *Symmetric traveling salesman problem (TSP).* consultado en: `https://wwwproxy.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`, 8 de junio de 2019.

[UAIC 2019] UAIC, Alexandru Ioan Cuza University of Iaşi, *Benchmark data for the Single-Depot Multiple Traveling Salesman Problem (multiple-TSP).* consultado en: `https://profs.info.uaic.ro/`, 30 de marzo de 2019.

[Vakhania *et al.* 2016] Vakhania Nodari, Hernández José Alberto, Alonso Pecina Federico y Zavala Crispín, *A Simple Heuristic for Basic Vehicle Routing Problem.* Journal of Computer Science Technology Updates, México 2016, p.38-44.

[Yousefikhoshbakht y Sedighpour 2012] Yousefikhoshbakht Majid y Sedighpour Mohammad, *A combination of sweep algorithm and elite ant colony optimization for solving the multiple traveling salesman problem.* Proceedings of the Romanian academy A, 2012, vol. 13, no 4, p. 295-302.

[Yousefikhoshbakht *et al.* 2013] Yousefikhoshbakht Majid, Didehvar F. y Rahmati, F., *Farhad. Modification of the ant colony optimization for solving the multiple traveling salesman problem.* Romanian Journal of Information Science and Technology, 2013, vol. 16, no 1, p. 65-80.

[Wang *et al.* 2013] Wang, Xiaobin, Liu, Daibo y Hou, Mengshu, *A novel method for multiple depot and open paths, multiple traveling salesmen problem.* In 2013 IEEE 11th international symposium on applied machine intelligence and informatics (SAMI). IEEE, 2013. p. 187-192.

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

FCAeI

FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA

Cuernavaca, Morelos a 02 de septiembre del 2019.

DRA. LAURA PATRICIA CEBALLOS GILES
DIRECTORA DE LA FCAeI
PRESENTE

En mi carácter de revisor de Tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado Maestría en Optimización y Cómputo Aplicado, del estudiante Víctor Hugo Pacheco Valencia, con matricula 10010401, con el título **DISEÑO DE UN ALGORITMO EN 2 FASES PARA UN MTSP** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, concluyo que el trabajo se caracteriza por el establecimiento de objetivos académicos pertinentes y una metodología adecuada para su logro. Además construye una estructura coherente y bien documentada, por lo cual considero que los resultados obtenidos contribuyen al conocimiento del tema tratado.

Con base en los argumentos precedentes me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que la estudiante continúe con los trámites que esta Secretaría de Investigación y Posgrado tenga establecidos para obtener el grado mencionado.

Atentamente
Por una humanidad culta
Una universidad de excelencia

Dr. Nodari Vakhania Maisuradze
Profesor- investigador
Centro de Investigación en Ciencias

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA

Cuernavaca, Morelos a 02 de septiembre del 2019.

DRA. LAURA PATRICIA CEBALLOS GILES
DIRECTORA DE LA FCAeI
PRESENTE

En mi carácter de revisor de Tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado Maestría en Optimización y Cómputo Aplicado, del estudiante Víctor Hugo Pacheco Valencia, con matricula 10010401, con el título **DISEÑO DE UN ALGORITMO EN 2 FASES PARA UN MTSP** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, concluyo que el trabajo se caracteriza por el establecimiento de objetivos académicos pertinentes y una metodología adecuada para su logro. Además construye una estructura coherente y bien documentada, por lo cual considero que los resultados obtenidos contribuyen al conocimiento del tema tratado.

Con base en los argumentos precedentes me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que la estudiante continúe con los trámites que esta Secretaría de Investigación y Posgrado tenga establecidos para obtener el grado mencionado.

Atentamente
Por una humanidad culta
Una universidad de excelencia

Dr. José Alberto Hernández Aguilar
Profesor- investigador
Facultad de Contaduría, Administración e Informática

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

FCAeI

FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA

Cuernavaca, Morelos a 02 de septiembre del 2019.

DRA. LAURA PATRICIA CEBALLOS GILES
DIRECTORA DE LA FCAeI
PRESENTE

En mi carácter de revisor de Tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado Maestría en Optimización y Cómputo Aplicado, del estudiante Víctor Hugo Pacheco Valencia, con matricula 10010401, con el título **DISEÑO DE UN ALGORITMO EN 2 FASES PARA UN MTSP** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, concluyo que el trabajo se caracteriza por el establecimiento de objetivos académicos pertinentes y una metodología adecuada para su logro. Además construye una estructura coherente y bien documentada, por lo cual considero que los resultados obtenidos contribuyen al conocimiento del tema tratado.

Con base en los argumentos precedentes me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que la estudiante continúe con los trámites que esta Secretaría de Investigación y Posgrado tenga establecidos para obtener el grado mencionado.

Atentamente
Por una humanidad culta
Una universidad de excelencia

Dr. Federico Alonso Pecina
Profesor- investigador
Facultad de Contaduría, Administración e Informática

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA

Cuernavaca, Morelos a 02 de septiembre del 2019.

DRA. LAURA PATRICIA CEBALLOS GILES
DIRECTORA DE LA FCAeI
PRESENTE

En mi carácter de revisor de Tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado Maestría en Optimización y Cómputo Aplicado, del estudiante Víctor Hugo Pacheco Valencia, con matricula 10010401, con el título **DISEÑO DE UN ALGORITMO EN 2 FASES PARA UN MTSP** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, concluyo que el trabajo se caracteriza por el establecimiento de objetivos académicos pertinentes y una metodología adecuada para su logro. Además construye una estructura coherente y bien documentada, por lo cual considero que los resultados obtenidos contribuyen al conocimiento del tema tratado.

Con base en los argumentos precedentes me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que la estudiante continúe con los trámites que esta Secretaría de Investigación y Posgrado tenga establecidos para obtener el grado mencionado.

Atentamente
Por una humanidad culta
Una universidad de excelencia

Dr. José Crispín Zavala Díaz
Profesor- investigador
Facultad de Contaduría, Administración e Informática

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

FCAeI

FACULTAD DE CONTADURÍA, ADMINISTRACIÓN E INFORMÁTICA

Cuernavaca, Morelos a 02 de septiembre del 2019.

DRA. LAURA PATRICIA CEBALLOS GILES
DIRECTORA DE LA FCAeI
PRESENTE

En mi carácter de revisor de Tesis, hago de su conocimiento que he leído con interés la tesis para obtener el grado Maestría en Optimización y Cómputo Aplicado, del estudiante Víctor Hugo Pacheco Valencia, con matricula 10010401, con el título **DISEÑO DE UN ALGORITMO EN 2 FASES PARA UN MTSP** por lo cual, me permito informarle que después de una revisión cuidadosa de dicha tesis, concluyo que el trabajo se caracteriza por el establecimiento de objetivos académicos pertinentes y una metodología adecuada para su logro. Además construye una estructura coherente y bien documentada, por lo cual considero que los resultados obtenidos contribuyen al conocimiento del tema tratado.

Con base en los argumentos precedentes me permito expresar mi **VOTO APROBATORIO** por lo que de mi parte no existe inconveniente para que la estudiante continúe con los trámites que esta Secretaría de Investigación y Posgrado tenga establecidos para obtener el grado mencionado.

Atentamente
Por una humanidad culta
Una universidad de excelencia

Dra. Lorena Díaz González
Profesor- investigador
Centro de Investigación en Ciencias

Cuernavaca, Morelos, a 25 de septiembre de 2019

CERTIFICACIÓN DE AUTORÍA

Yo, Víctor Hugo Pacheco Valencia, certifico que la Disertación titulada, DISEÑO DE UN ALGORITMO EN 2 FASES PARA UN MTSP, la cual presento como requisito para optar por el grado de Maestro en Administración de Organizaciones en la Facultad de Contaduría, Administración e Informática de la Universidad Autónoma del Estado de Morelos en el país México, es el producto de mi labor investigativa.

Asimismo, doy fe de que este trabajo ha observado las normas establecidas en el Reglamento de la Universidad y del posgrado de la Universidad Autónoma del Estado de Morelos para su realización.

Con lo anterior, deslindo a la institución de toda acción que genere plagios y asumo toda responsabilidad.

Víctor Hugo Pacheco Valencia
Matricula: 10010401