UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

INSTITUTO DE INVESTIGACIÓN EN CIENCIAS BÁSICAS Y APLICADAS

CENTRO DE INVESTIGACIÓN EN INGENIERÍA Y CIENCIAS APLICADAS

HIPERHEURÍSTICA PARA EL PROBLEMA DEL TRANSPORTE MULTI-DEPÓSITO CON VENTANAS DE TIEMPO.

TESIS PROFESIONAL PARA OBTENER EL GRADO DE:

DOCTORADO EN INGENIERÍA Y CIENCIAS APLICADAS CON OPCIÓN TERMINAL EN TECNOLOGÍA ELÉCTRICA

P R E S E N T A:

M.I.C.A. ALFONSO D´GRANDA TREJO

DIRECTORES: DR. MARCO ANTONO CRUZ CHÁVEZ

SINODALES: DR. ÁLVARO ZAMUDIO LARA, DRA. MARGARITA TECPOYOTL TORRES, DR. FEDERICO ALONSO PECINA, DR. MARTÍN HERIBERTO CRUZ ROSALES, DRA.JESÚS DEL CARMEN PERALTA ABARCA, DRA. BEATRIZ MARTÍNEZ BAHENA.

CUERNAVACA, MOR. NOVIEMBRE, 2019

Cuernavaca, Morelos, a 19 de noviembre de 2019.

DR. ROSENBERG JAVIER ROMERO DOMÍNGUEZ
COORDINADOR DEL POSGRADO EN
INGENIERÍA Y CIENCIAS APLICADAS
P R E S E N T E

Atendiendo a la solicitud para emitir DICTAMEN sobre la revisión de la TESIS titulada **"HIPERHEURÍSTICA PARA EL PROBLEMA DEL TRANSPORTE MULTI-DEPÓSITO CON VENTANAS DE TIEMPO."** que presenta el alumno **ALFONSO D'GRANDA TREJO,** para obtener el título de **DOCTORADO EN INGENIERÍA Y CIENCIAS APLICADAS** con opción terminal en **TECNOLOGÍA ELÉCTRICA**.

Nos permitimos informarle que nuestro voto es:

NOMBRE	DICTAMEN	FIRMA
DR. ÁLVARO ZAMUDIO LARA	Aprobado	
DRA. MARGARITA TECPOYOTL TORRES	Aprobado	
DR. FEDERICO ALONSO PECINA (FCAeI)	Aprobado	
DR. MARTÍN HERIBERTO CRUZ ROSALES (FCAeI)	Aprobado	
DRA. JESÚS DEL CARMEN PERALTA ABARCA (FCQeI)	Aprobado	
DRA. BEATRIZ MARTÍNEZ BAHENA (FCQeI)	Aprobado	
DR. MARCO ANTONIO CRUZ CHÁVEZ	Aprobado	

PLAZO PARA LA REVISIÓN 20 DÍAS HÁBILES (A PARTIR DE LA FECHA DE RECEPCIÓN DEL DOCUMENTO)

NOTA. POR CUESTION DE REGLAMENTACIÓN LE SOLICITAMOS NO EXCEDER EL PLAZO SEÑALADO, DE LO CONTRARIO LE AGRADECEMOS SU ATENCIÓN Y NUESTRA INVITACIÓN SERÁ CANCELADA.

RESUMEN

Hoy en día, una gran parte de los problemas de distribución de bienes consiste básicamente, en asignar una ruta a cada vehículo de una flota para repartir o recoger mercancías, lo cual constituye un conjunto de problemas habituales que no se resuelven de manera óptima y causan una merma significativa en los ingresos de las empresas. Hacer una buena planeación de ruteo vehicular requiere de herramientas tecnológicas precisas que logren un mejor aprovechamiento de la flota en términos de su capacidad de transporte y, por consecuencia, un mejor nivel de servicio. El diseño de rutas eficientes permite definir el número de vehículos que serán utilizados para visitar un número importante de clientes (destinos), es un factor crítico que afecta de manera considerable la logística y la competitividad de muchas compañías.

El problema tratado en esta tesis doctoral se le conoce como el *problema de ruteo vehicular con capacidades homogéneas* (CVRP por sus siglas en inglés Capacitated Vehicle Routing Problem) en el cual, cada cliente tiene una demanda que es conocida y no se puede dividir y es atendida por un único vehículo. Cada uno de los clientes es un nodo de una red, para conectar a los clientes, geográficamente dispersos, se tienen arcos (arista) los cuales son no dirigidos. Cada arco tiene asociado un costo (positivo). Se cuenta con un número K de vehículos disponibles los cuales tienen una capacidad limitada Q. El objetivo es determinar la ruta para cada uno de los vehículos, los cuales parten del depósito y deben regresar a él de tal forma que se minimicen los costos y se satisfagan las demandas de todos los clientes. Se considera una serie de restricciones entre las que destacan, que los vehículos no pueden exceder su capacidad ni visitar a un cliente dos veces.

Para tratar este problema se proponen dos algoritmos, un algoritmo de recocido simulado secuencial apoyado en una estructura híbrida de vecindad y con reinicio como mecanismo de exploración y explotación del espacio de soluciones. El otro algoritmo es el de recocido simulado distribuido también con reinicio y una estructura híbrida de vecindad, el cual fue desarrollado con la librería de "Paso de Mensajes (MPI)".

Los algoritmos desarrollados se ejecutan de manera secuencial y distribuida en CPU y en el clúster Cuexcomate respectivamente. Fueron evaluados utilizando un conjunto de instancias obtenidas de la literatura para el CVRP. Los resultados obtenidos fueron muy cercanos a los óptimos reportados en el estado del arte y en algunos casos se igualó a éstos. En la metodología experimental, se utilizaron instancias pequeñas (de 32 a 150 clientes) para evaluar el desempeño del algoritmo secuencial. Para el algoritmo distribuido se usaron instancias medianas (de 199 a 261 clientes).

ABSTRACT

Currently, there are several models that are breaking the paradigms in the design of transport routes, allowing solve problems that have to do with the uncertainty of customer demand. The benefit is the better use of the fleet in terms of transport capacity and, consequently, a better level of the service. The design of efficient routes to define the number of vehicles that will be used to visit a large number of clients (destinations), is a critical factor affecting significantly the logistics and competitiveness of many companies. This fact allows the creation of specific areas, in several companies, dedicated to design their vehicle routing, in order to obtain strategies to reduce the transportation costs, instead of the generation of functions of operational nature. Make a good vehicle routing planning requires precise technological tools.

This thesis is focused on the problem known as Capacitated Vehicle Routing Problem (CVRP), which deals with the design a routing from the deposit, where begin and end their tour routes. Vehicles go to visit a set of geographically dispersed customers, whose demand is known. Its goal is to travel the route with minimal cost. Among restrictions, it is important to point out that vehicles cannot exceed their capacity or visit a client twice.

To solve this problem, two algorithms are proposed: a sequential simulated annealing algorithm supported by a hybrid neighborhood structure and with restart as a mechanism for exploring and exploiting the solution space.

The other algorithm is simulated annealing also distributed with restart and a hybrid neighborhood structure, which was developed with the "Message Passage (MPI)" library.

The developed algorithms are executed sequentially and distributed in CPU and in the cluster Cuexcomate respectively. They were evaluated using a set of instances obtained from the literature for the CVRP. Results were very close to the optimal reported. In the experimental methodology, small instances (from 32 up to 150 customers) were used to evaluate the performance of sequential algorithm and medium instances (from 199 up to 261 customers) for the distributed algorithm.

AGRADECIMIENTOS

A CONACYT por brindarme el apoyo económico para la realización de mis estudios de posgrado.

Al Centro de Investigación en Ingeniería y Ciencias Aplicadas por .

Al Dr. Marco Antonio Cruz Chávez, director de este trabajo de investigación, por la orientación y consejos para la realización de este trabajo.

A los integrantes del comité tutoral y revisores de tesis: Dra. Margarita Tecpoyotl Torres, Dr. Martín Heriberto Cruz Rosales, Dr. Federico Alonso Pecina, Dr. David Juárez Romero, Dr. Álvaro Zamudio Lara, Dra Carmen y Dra. Beatriz por sus comentarios, orientación y consejos para la realización de este trabajo de investigación.

DEDICATORIAS

Primeramente, al Gran Arquitecto Del Universo.

A mis hijos que tanto amo.

A mis compañeras del grupo CAOS por su amistad, apoyo, paciencia y

De manera muy especial a quien me devolvió el interés en seguir construyendo mi vida y me ha alentado a ser mejor persona, que me hace vencer una de las dos fuerzas que mueven todo por medio de la otra,.

NOMENCLATURA

Problema de Ruteo Vehicular con Ventanas de Tiempo (MDVRPTW)

A	Aristas que conectan dos clientes en el grafo (Distancia entre dos clientes)
ai	Apertura de la ventana de tiempo del cliente i
bi	Cierre de la ventana de tiempo del cliente i
C	Capacidad máxima del vehículo
cij	Costo de ir de un cliente *i* a un clente *j*
di	Demanda del cliente i
E	Apertura de la ventana de tiempo del depósito
K	Tamaño total de la flota de vehículos
L	Cierre de la ventana de tiempo del depósito
m	Ruta perteneciente a una solución
N	Total de clientes a calendarizar (tamaño de la instancia)
Si	Tiempo de servicio requerido para un cliente i
V	Cada uno de los vértices del grafo, correspondiente a un cliente
Wik	Tiempo de inicio del servicio del vehículo *k* en el cliente *i*

ÍNDICE DE TABLAS

Tabla 4. 1 Datos que almacenan los vectores de la estructura de solución.**¡Error! Marcador no definido.**

Tabla 5. 1 Número de rutas generado..... **¡Error! Marcador no definido.**

Tabla 5. 2 Instancias de Solomon R..... **¡Error! Marcador no definido.**

Tabla 5. 3 Instancias de Solomon C..... **¡Error! Marcador no definido.**

Tabla 5. 4 Instancias de Solomon RC. ..... **¡Error! Marcador no definido.**

Tabla 5. 5 Gehring & Homberger's c1_10. ..... **¡Error! Marcador no definido.**

Tabla 5. 6 Gehring & Homberger's c2_10. ..... **¡Error! Marcador no definido.**

Tabla 5. 7 Gehring & Homberger's r1_10..... **¡Error! Marcador no definido.**

Tabla 5. 8 Gehring & Homberger's r2_10..... **¡Error! Marcador no definido.**

Tabla 5. 9 Gehring & Homberger's rc1_10. ..... **¡Error! Marcador no definido.**

Tabla 5. 10 Gehring & Homberger's rc2_10 ... **¡Error! Marcador no definido.**

ÍNDICE DE FIGURAS

Figura 2. 1. Ejemplo de solución al VRP. ..... **¡Error! Marcador no definido.**
Figura 2. 2 Modelo matemático del VRP. ..... **¡Error! Marcador no definido.**
Figura 2. 3 Métodos de solución para el VRP. **¡Error! Marcador no definido.**
Figura 2. 4 Ejemplo de datos agrupados. ..... **¡Error! Marcador no definido.**
Figura 2. 5 Métodos de dos fases. ..... **¡Error! Marcador no definido.**
Figura 2. 6 Pseudocódigo CA-CVRP ..... **¡Error! Marcador no definido.**

Figura 3. 1 Modelo matemático del problema de agrupamiento.**¡Error! Marcador no definido.**
Figura 3.2 Números de Stirling 100 clientes. ... **¡Error! Marcador no definido.**
Figura 3.3 Números de Stirling 900 clientes. ... **¡Error! Marcador no definido.**

Figura 4. 1 Pseudocódigo del algoritmo propuesto.**¡Error! Marcador no definido.**
Figura 4. 2 Ejemplo del archivo de entrada. .... **¡Error! Marcador no definido.**
Figura 4. 3 Estructura de los datos de entrada.**¡Error! Marcador no definido.**
Figura 4. 4 Estructura de la solución. ..... **¡Error! Marcador no definido.**
Figura 4. 5 Ejemplo de formación de agrupamientos.**¡Error! Marcador no definido.**
Figura 4. 6 Representación gráfica del ángulo izquierdo**¡Error! Marcador no definido.**

Contenido

RESUMEN

ABSTRACT

INDICE DE FIGURAS…….…… XII

INDICE DE TABLAS……..……. XVII

Contenido

CAPÍTULO 1 INTRODUCCIÓN 3

Antecedentes..... 3

Modelos del Problema de Ruteo Vehicular..... 5

Complejidad del problema MDVRPTW..... 11

Planteamiento del Problema..... 14

Hipótesis..... 14

Objetivo General..... 14

Objetivos Específicos 15

Justificación 15

Alcance de la investigación 16

Contribución 16

Organización de la Tesis 16

Capítulo 2: MARCO TEORICO..... 18

Técnicas heurísticas de Optimización 20

Clasificación de las Técnicas Metaheurísticas 20

Heurísticas..... 21

Metaheurísticas 23

Clasificación de las Técnicas Metaheurísticas 23

2.1 Estado del arte..... 29

Metaheurísticas aplicadas al MDVRP Y VRPTW 29
Capítulo 3: DESCRIPCIÓN DEL PROBLEMA 34
3.1 CRITERIOS DE AGRUPAMIENTO 34
3.2 MODELO MATEMATICO 35
3.3 ESPACIO DE SOLUCIONES 36
3.4 COMPLEJIDAD DEL PROLEMA 36
Capítulo 4: DESCRIPCIÓN DEL ALGORITMO DE AGRUPAMIENTO PROPUESTO 40
4.1 ALCANCE 40
4.2 DATOS DE ENTRADA 42
4.2.1 INSTANCIAS 43
4.3 CRITERIOS PARA LA FORMACION DE LOS GRUPOS 45
4.3.1 FORMACIÓN DE LOS AGRUPAMIENTOS 46
4.4 DATOS DE SALIDA 49
Capítulo 5: RESULTADOS 51
5.1 TABLAS COMPARATIVAS 51
Capítulo 6: Conclusiones y trabajos futuros 60
6.1 CONCLUSIONES 60
6.2 TRABAJOS FUTUROS 61
REFERENCIAS 62
Apéndice A CÓDIGO FUENTE 68
Apéndice B CÓDIGO FUENTE VERIFICADOR DE SOLUCIUONES 111

CAPÍTULO 1 INTRODUCCIÓN

Antecedentes

La logística es un conjunto de medios, métodos, infraestructuras que al combinarlos pretenden garantizar el almacenamiento, transporte y entrega de los bienes y servicios que requieren los clientes. Tiene su origen en la milicia, conocida como logística militar donde era preciso la coordinación de suministros hacia las tropas en el campo de batalla de una manera rápida y eficiente.

La logística está relacionada con los medios de transporte, es el medio para enviar productos y mercancías de un lugar geográfico a otro (lugares que se encuentran geográficamente dispersos) en años recientes esta situación se ha incrementado de manera importante a causa de la globalización y la aplicación de nuevas técnicas logísticas (Elizondo & Aceves, 2007). Un sistema logístico ágil y flexible permite a las empresas aumentar sus niveles de servicio, reducir costos y competir.

El manejo de la logística involucra principalmente, entre otras, las funciones siguientes: Transporte, gestión de flotillas, manipulación de materiales, entrega de bienes, diseño de rutas, manejo de inventarios. Es importante porque se enfoca en la entrega de productos a los consumidores, si el producto no llega, la empresa falla y su rentabilidad y viabilidad a largo plazo no se garantiza, entre más eficiente el transporte el negocio será rentable.

Así el problema principal de la logística consiste en encontrar un plan de entrega de bienes a un conjunto de clientes que están geográficamente dispersos, minimizando el costo de recorrido de los vehículos considerando las restricciones generadas por:

1. La red de transporte
2. Número de vehículos
3. Capacidad de los vehículos
4. Demanda de los clientes
5. La ubicación de los clientes
6. La ubicación del depósito o depósitos

Un problema frecuente en las decisiones logísticas, es un balance entre la reducción de los costos de transporte y la mejora del servicio al cliente. Se buscan las mejores rutas que deben seguir los vehículos para minimizar el tiempo o la distancia de los recorridos. Un buen sistema de redes logísticas (ruteo) logra de un 5% a un 20% en ahorro de los costos logísticos (Vigo & Toth, 2014).

Se han detectado tres problemáticas principales para establecer rutas: (Rodríguez, 2007):

1) A través de una red donde el punto de origen es diferente al punto de destino.
2) Cuando existen múltiples puntos de origen y destino.
3) Donde el punto de origen y destino son los mismos.

A este tipo de problemas se les ha denominado problema de ruteo vehicular (VRP).

El problema de Ruteo Vehicular (Vehicle Routing Problem, VRP por sus siglas en inglés), de ahora en adelante VRP, es uno de los problemas de optimización combinatoria que ha sido intensamente estudiado debido a muchas aplicaciones prácticas en el campo de la logística (Pop, Pop, Zelina, Lupse, & Chira, 2011). consiste en un diseño óptimo de redes logísticas, denominados *rutas*, para la entrega o recolección de bienes o personas desde un depósito central hacia un conjunto de puntos geográficamente dispersos, éste diseño

está sujeto a varias limitaciones, como capacidad del vehículo, longitud de la ruta, horarios de carga, descarga, entrega o recolección (conocidos como ventanas de tiempo), relaciones de precedencia entre los clientes, entre otras limitaciones (Laporte G., 1992) (Hasle & Kloster, 2007).

El objetivo del VRP es minimizar el costo de las rutas que inician y terminan en un depósito, para un conjunto de clientes con demandas conocidas.

El modelo de VRP es ampliamente utilizado en: la entrega de alimentos y bebidas a tiendas de abarrotes o CeDis (Centros de Distribución), transporte escolar y de personal, recolección de basura, empresas de mensajería entre otros usos.

Modelos del Problema de Ruteo Vehicular

El problema del VRP, ha involucrado cada vez más variantes que lo van haciendo computacionalmente más complicado de resolver: Las diferentes características de los clientes, el depósito o depósitos y vehículos, así como las diferentes restricciones operativas de las rutas, dan lugar a gran número de variantes del problema generando una amplia variedad de extensiones del VRP (Bermeo & Calderón, 2009) (González & González, 2006)

Los más conocidos y aplicados en investigaciones son:

1) Agente Viajero (Travelling Salesman Problem TSP)
2) Rutas de Vehículos con capacidades idénticas (CVRP)
3) Rutas de Vehículos con capacidades y flota heterogénea HFCVRP)
4) Rutas con entrega y recolección VRPB)
5) Con ventanas de tiempo VRPTW)
6) Con múltiples almacenes (VRPMD)
7) VRP Multidepósito (MDVRP)
8) VRP Periódico
9) VRP Estocástico (SVRP)

El problema de VRP es uno de los más comunes en la investigación de operaciones y de redes logísticas además de los más estudiados (Hiller & Lieberman, 2012) (Salazar González, 2001); plantea la búsqueda de la solución óptima con diferentes restricciones tales como: número de vehículos, su capacidad, lugares de destinos (clientes) y su demanda. Una formulación de éste tipo puede incluir un amplio número de variables y diversos parámetros. Este tema presenta un interés práctico y académico por constituirse en un problema de optimización combinatoria y pertenecen en su mayoría a la clase NP-Hard, pues no es posible resolverlos en tiempo polinomial (Vigo & Toth, 2014) (Archetti, Feillet, Gendreau, & Grazia, 2011) (Garey & Johnson, 1979).

Cuando a un cliente se le asocia una demanda y al vehículo que le otorga el servicio una capacidad, es cuando se afirma que el problema del agente viajero da origen al problema de ruteo. (Vigo & Toth, 2014).

En 1960, Miller, Tucker y Zemlin trabajaron con el agente viajero múltiple conocido como **m-TSP**, considerado como una generalización del TSP en donde se tiene un depósito y **m** vehículos, (**m** agentes viajeros). El objetivo del modelo es construir exactamente ***m*** *rutas*, una para cada vehículo, de modo que cada cliente (con una demanda o requerimiento del servicio) sea visitado una vez por uno de los vehículos (a los cuales se le ha asignado un número determinado de clientes) sin rebasar su capacidad e iniciando y finalizando en el depósito.

El modelo del CVRP determina una cantidad exacta de rutas con mínimo costo, este costo, se define como la suma de los costos de los arcos pertenecientes a los recorridos, de tal manera que cada recorrido realizado por un vehículo visita el centro de distribución (CeDis), la suma de las demandas de los clientes definidos para cada ruta no excede la capacidad del vehículo.

Del modelo generalizado del CVRP, se generan dos categorías más, el VRP

Homogéneo en el cual se aplican las mismas variables a cada cliente (pero cada una con las características determinadas por éste) como lo son distancias, ventanas de tiempo, entregas, etc. En el VRR Heterogéneo cada cliente utiliza diferentes variables como entregas fraccionadas, distancias asimétricas, más de un depósito, distancias, ventanas de tiempo, entre otras.

En la Figura 1 se resumen los tipos de VRP:

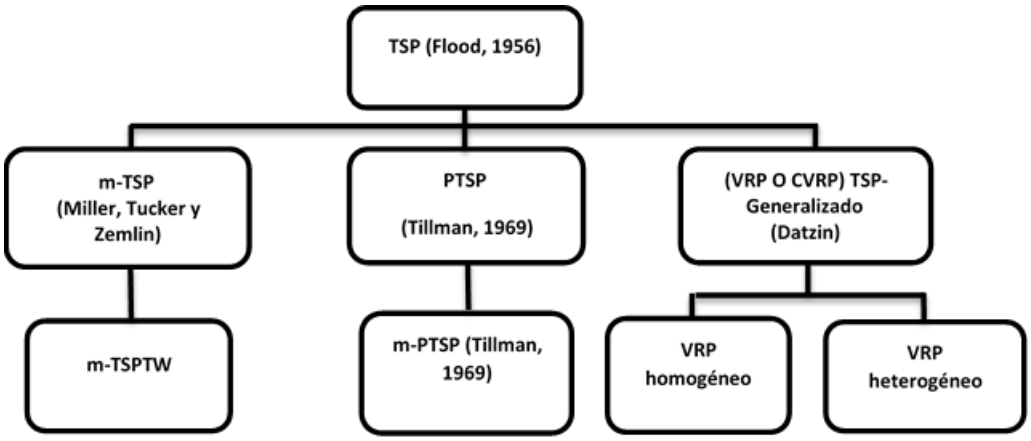


Figura 1. Modelos del VRP.

Dentro del VRP Homogéneo se desprenden cuatro categorías:

DVRP, la restricción de capacidad es reemplazada por la de distancia máxima, la cual no puede ser rebasada. (Vigo & Toth, 2014), (Birattari, Chiarandini, Manfrin, & Mastrolilli, 2004) y (Dorronsoro Díaz, 2012). Tiene una variante DCVRP que tiene la restricción en la capacidad y en la distancia máxima.

VRPTW que surge en 1967, se le ha agregado la restricción adicional de una ventana de tiempo asociada a cada cliente, esta ventana es el intervalo de tiempo donde el cliente debe ser atendido, si llega antes tendrá que esperarse para hacer el servicio requerido, si la llegada es tardía, se pierde la atención al cliente.

(Dorronsoro Díaz, 2012), (Vigo & Toth, 2014) y (Restrepo, Medina, & Cruz, 2008)

Existen tres variaciones más:

VRPTD (VRP with time deadlines), surge en 1986 y es denominado VRP con ventanas rígidas de tiempo.

VRPMTW (VRP con ventanas de tiempo múltiples), formulado en 1988.

VRPSTW (VRP with soft time windows) con ventanas de tiempo blandas en al año 1992.

VRPB (VRP with backhauls) con retornos se modela en al año 1985, considera los casos en que los clientes devuelvan una parte de la mercancía por situaciones de cambio por obsolescencia, daños o especificaciones erróneas en las entregas. Se deben hacer las entregas a todos los clientes para posteriormente hacer las recolecciones, debe de considerarse que el vehículo tenga la capacidad para almacenar el tamaño de las devoluciones. Las mercancías entregadas y devueltas son conocidas con anticipación. (Dorronsoro Díaz, 2012) (Vigo & Toth, 2014).

Deriva en VRPBTW y en el año 1994 al hibridarse los modelos VRPB Y VRPTW.

SDVRP (Split delivery VRP) con entregas parciales, permite que el cliente pueda ser atendido por diferentes vehículos porque considera el tamaño del pedido que por lo general son grandes cantidades. (Vigo & Toth, 2014), (Dorronsoro Díaz, 2012), (Olivera, 2004)

También tiene una hibridación y surge en el año 1995 el SDVRPTW (entregas fraccionadas y ventanas de tiempo), se crea a partir de los modelos SDVRP Y VRPTW.

La Figura 2 resume los modelos arriba comentados.

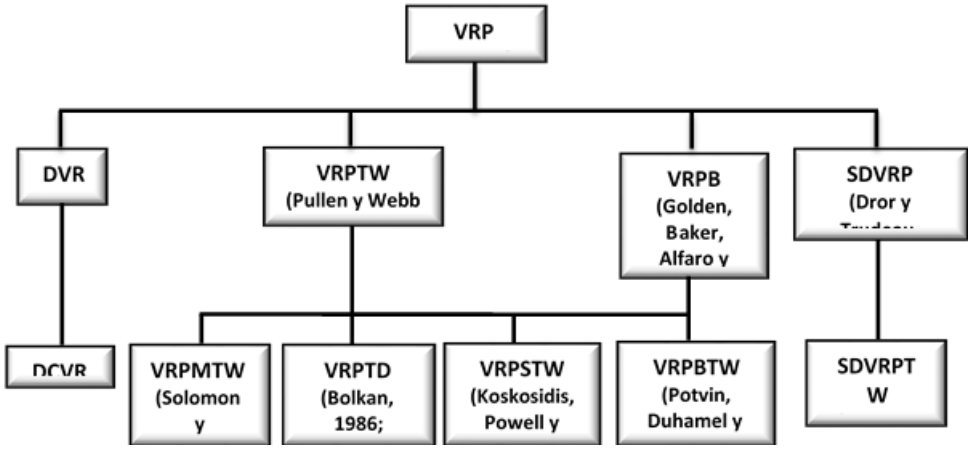


Figura 2. Modelos del VRP Homogéneo.

En el VRP Heterogéneo se hace referencia a características desiguales, se han aplicado 7 modelos, éstos se describen brevemente:

1) **VRPHF** (Problema de Ruteo de Vehículos Flota Heterogénea), modelo con flota vehicular diversa, hay variación en los costos y capacidades de los vehículos, se considera ilimitada la cantidad de vehículos a emplear.
2) **PVRP** (Problema de Ruteo de Vehículos Periódico) el modelo aplica un horizonte de planeación de x días, en los otros modelos es de un día. Considera servicios a clientes cuya ubicación geográfica no permite sean atendidos en un solo día de trabajo.
3) **MULTI-TRIP-VRP** (Problema de Ruteo de Vehículos Múltiples Viajes), asigna varias rutas a un solo vehículo para atender a los clientes.
4) **MULTI-DEPOT-VRP** (Problema de Ruteo de Vehículos Múltiples Depósitos), también la empresa puede contar con varios depósitos lo que permite atender a los clientes, la diferencia con el VRP es que los clientes necesitan de varios artículos o mercancías que no están en solo depósito.
5) **MCVRP** (Problema de Ruteo de Vehículos Múltiples Capacidades), este modelo permite la transportación de más de una cantidad de objetos a la

vez.

6) **MOVRP** (Problema de Ruteo de Vehículos Múltiples Objetivos), es un modelo que necesita cumplir varios objetivos como lo son: costos, ventanas de tiempo, distancias, etc.
7) **SVRP** (Problema de Ruteo de Vehículos Estocástico), aplica componentes aleatorios de tipo estocástico, esto es que su solución depende de la probabilidad.

Los modelos anteriores se resumen en la Figura 3.

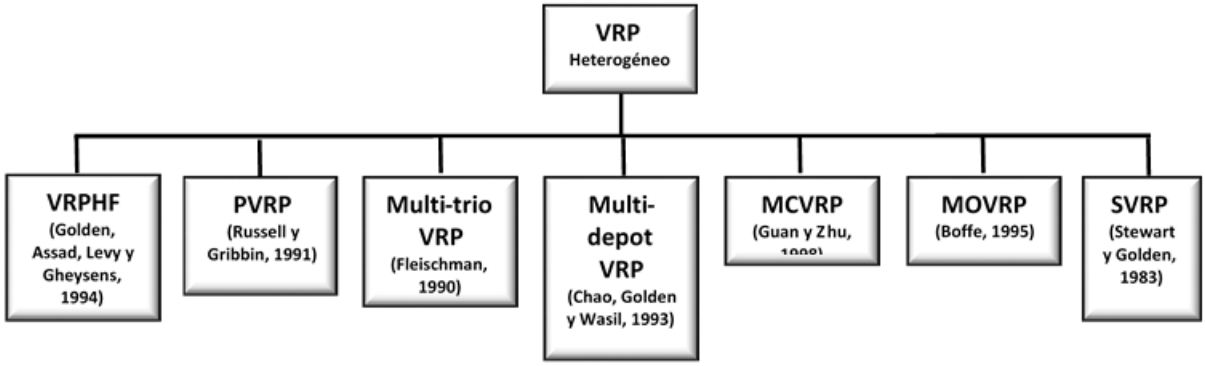


Figura 3. Modelos del VRP Heterogéneo

En esta investigación se trabajará con el modelo de Problema de Ruteo Vehicular con Multi-Depósitos y Ventanas de Tiempo conocido como MDVRPTW por sus siglas en inglés, el cual se describe como un conjunto de *n* clientes, dos o más depósitos, las distancias entre los clientes y los depósitos son conocidas; los vehículos que se utilizan para el reparto son de capacidad idéntica. El problema a resolver será el encontrar los recorridos que deben realizar los vehículos para minimizar la distancia total recorrida.

Complejidad del problema MDVRPTW

De manera general, los problemas se pueden clasificar en dos clases: los tratables y los intratables.

Para los intratables no existe ningún algoritmo que permita resolverlos en todos los casos, incluyen aquellos problemas para los que sí se conoce un algoritmo que podría resolverlos pero que por la cantidad de tiempo computacional necesario para hacerlo los convierte en inaccesibles independientemente de la capacidad computacional de que se disponga. Para estos problemas, no existe ningún algoritmo que permita resolverlos en un número de pasos que sea una función polinomial del tamaño de entrada del problema, no se pueden resolver en tiempo polinomial.

Pero la tarea de encontrar un algoritmo que resuelva con pocos recursos un problema, no es lo que realmente importa, el inconveniente es saber si existe una solución o no a esta problemática presentada y que tan compleja resultará su solución.

Para saber el grado de complejidad que puede tener un problema, se hace uso del modelo computacional de la Máquina de Turing (Sanjeev & Boaz, 2009) (Garey & Johnson, 1979) (Cruz-Chávez, Peralta-Abarca, & Moreno-Bernal, 2014), con el cual se obtiene una clasificación de los problemas con base al grado de complejidad inherente para resolver éstos.

La Máquina de Turing, inventada por el matemático inglés Alan M. Turing en 1937, es una máquina autómata artesanal que se utiliza para clasificar a los problemas de acuerdo al tipo de Máquina de Turing que puede existir para resolver un problema, siendo el principal modelo computacional que soporta la “Teoría de la complejidad de los problemas”.

El modelo computacional de Turing clasifica los problemas por el grado de complejidad para resolverlo, así a través de este modelo, se han detectado

problemas intratables, clasificados como NP (Nondeterministic Polynomial Time), que se piensa son imposibles de resolver en un tiempo razonable cuando el número de variables que componen el problema es una cantidad extremadamente grande. Dentro de este grupo de problemas NP, se encuentran dos subconjuntos de problemas:

1. Problemas **P** para los cuales existe una máquina de Turing determinista que los puede resolver en tiempo polinómico[1], esto indica que existe un algoritmo determinista[2] con complejidad polinomial que lo puede resolver. Estos problemas se consideran como la clase de problemas de reconocimiento relativamente sencillos, aquellos para los que existen algoritmos eficientes o exactos (Papadmitriou & Steiglitz, 1998).
2. Los NP-completos, no existe una máquina de Turing determinista que pueda resolverlos en tiempo polinómico, en su lugar, se puede encontrar un valor próximo a la solución del problema mediante una máquina de Turing no determinística (Alfonseca, 2000) acotando polinomialmente el tiempo. Estos problemas NP son los problemas cuya solución hasta la fecha no han podido ser resueltos de manera exacta por medio de algoritmos deterministas en tiempo polinomial, en su lugar se tratan de resolver por algoritmos no-deterministas[3] acotados en tiempo polinomial y cuya solución deseable sea de complejidad polinomial, esta clase de algoritmos se conocen como heurísticas[4] computacionales.

[1] En computación, cuando el tiempo de ejecución de un algoritmo (mediante el cual se obtiene una solución al problema) es menor que un cierto valor calculado a partir del número de variables implicadas (variables de entrada) usando una fórmula polinómica, se dice que dicho problema se puede resolver en un tiempo polinómico.

[2] En ciencias de la computación, un algoritmo determinista es un algoritmo del cual se conocen sus variables de entrada y por consiguiente siempre generará el mismo resultado, como el algoritmo simplex por ejemplo.

[3] Algoritmo no determinista es un algoritmo que con las mismas variables de entrada ofrece diferentes resultados. No se puede saber de antemano cuál será el resultado de la ejecución de un algoritmo no determinista.

[4] Heurística es un método con reglas empíricas para encontrar soluciones a problemas basados en la experiencia, el cual no tiene una manera de comprobar que se alcanzó la mejor solución y sirve para encontrar soluciones aproximadas a la mejor solución en diversos problemas complejos.

El Problema de satisfacibilidad booleana (también llamado SAT) fue el primer problema identificado como perteneciente a la clase de complejidad NP-completo. Una instancia del problema SAT es una expresión booleana que combina variables booleanas con operadores booleanos. El teorema de Cook fue la primera prueba de existencia de problemas NP-Completos.

El problema del MDVRPTW es considerado como un problema para el cual no se ha encontrado un algoritmo que pueda resolverlo en un tiempo polinomial, es probable que en el caso peor el tiempo de ejecución para cualquier algoritmo que resuelva el TSP aumente de forma exponencial con respecto al número de ciudades, por eso, se le ha catalogado en teoría de la complejidad computacional como NP-completo (Daza, Montoya, & Narducci, 2009) (Dasgupta, Papadimitriou, & Vazirani, 2006).

En la Tabla 1 se muestra la forma en que se han clasificado algunos problemas, a la derecha están los problemas que pueden ser resueltos de forma eficiente, a la izquierda los que no tienen una solución en tiempo polinomial. (Dasgupta, Papadimitriou, & Vazirani, 2006).

Tabla 1. Clasificación de los problemas P y NP

Hard problems (NP-complete)	Easy problems (in P)
3SAT	2SAT, HORN SAT
TRAVELING SALESMAN PROBLEM	MINUMUM SAPNNING TREE
LONGEST PATH	SHORTEST PATH
3D MATCHING	BIPARTITE MATCHING
KNAPSACK	UNARY KNAPSACK
INDEPENDENT SET	INDEPENDENT SET on tres
INTEGER LINEAR PROGRAMMING	LINEAR PROGRAMMING
RUDRATA PATH	EULER PATH
BALANCED CUT	MINIMUM CUT

Las metodologías actuales de solución para el VRP, requieren de mucho tiempo de procesamiento para encontrar la solución óptima por la cantidad de operaciones a resolver.

Planteamiento del Problema

Hipótesis

El usar una hiperheurística y aplicando estructuras de vecindades híbridas dentro de su búsqueda local, permitirá obtener estrategias de solución, eficientes y eficaces Problema de Ruteo Vehicular con Multi-Depósitos y Ventanas de Tiempo conocido como (MDVRPTW).

Objetivo General

Desarrollar una hiperheurística para la solución de instancias para el Problema de Ruteo Vehicular con Multi-Depósitos y Ventanas de Tiempo conocido como MDVRPTW, que se ejecute en un clúster computacional haciendo uso de

estructuras híbridas.

Objetivos Específicos

1. Diseñar y programar la hiperheurística usando estructuras híbridas de vecindad que sean eficientes, que permita obtener soluciones eficaces para el problema del MDVRPTW.
2. Realizar el análisis estadístico para la validación de los resultados del algoritmo propuesto con otras metaheurísticas aplicadas.
3. Programar la hiperheurística, de forma distribuida con MPI, para evaluar su desempeño en la solución del modelo de transporte vehicular con capacidades homogéneas en instancias de benchmarks mayores a 1000 clientes.
4. Medir la eficacia y speep-up del algoritmo distribuido para valorar su ejecución al trabajar con diferente número de procesadores.

Justificación

Los algoritmos capaces de calcular modelos matemáticos de optimización de rutas se han vuelto clave en la investigación sobre las actividades del transporte, para proporcionar bienes y / o servicios a partir de uno o más puntos de suministro a varios puntos geográficamente dispersos; en optimización combinatoria. El problema de ruteo vehicular con Multi-Depósitos y Ventanas de Tiempo (conocido como MDVRPTW por sus siglas en inglés) es una variante del problema de ruteo de vehículos (VRP) con la adición de las restricciones: ventanas de tiempo y suministro desde varios depósitos: que agrega condiciones al modelo inherentes y útiles que lo aproximan a problemas en la vida real.

Alcance de la investigación

La investigación que se realiza, trabaja con el modelo del Transporte Vehicular Multidepósitos con Ventanas de Tiempo aplicando la metaheurística del algoritmo hiperheurístco con una estructura híbrida de vecindad. El algoritmo programado, fue paralelizado aplicando programación distribuida con MPI. Las pruebas experimentales fueron realizadas utilizando un clúster computacional para mejora de la eficiencia del algoritmo.

Contribución

Las aportaciones de la presente investigación doctoral son:

a) Una alternativa de solución al problema de ruteo de vehículos con multi-depósito con ventanas de tiempo mediante una estrategia Hiperheurística.
b) El diseño y programación de nuevos movimientos de perturbación con búsqueda guiada que la generen soluciones vecinas para ser aplicados como heurísticas de bajo nivel.
c) Un algoritmo de verificación de soluciones del problema de ruteo de vehículos con multi- depósito con ventanas de tiempo.

Organización de la Tesis

El trabajo ha sido organizado de la manera siguiente;

- Capítulo 1. Se presentan los antecedentes del problema MDVRPTW, en donde se define el problema, objetivos, alcances y contribuciones.
- Capítulo 2. Realiza una presentación de la teoría de la complejidad, técnicas de optimización existentes y una revisión del estado del arte.
- Capítulo 3. Se hace una presentación del problema de transporte vehicular, su modelo matemático, los benchmark utilizados y la hiperheurística de solución aplicada a la investigación.
- Capítulo 4. Se presenta la metodología de solución, la solución inicial, la forma de representarla, la generación de las soluciones vecinas, las

heurísticas de los movimientos de mejora.

- Capítulo 5. Se muestran los resultados experimentales obtenidos de la ejecución del algoritmo en cuanto a la eficacia y eficiencia.
- Finalmente se muestran las conclusiones y trabajos futuros del tema en el Capítulo 6, así como los anexos y bibliografía.

Capítulo 2: MARCO TEORICO

La optimización combinatoria es una rama de la optimización en ciencias computacionales y matemáticas aplicadas, relacionada con teoría de algoritmos y complejidad computacional (Cook et al., 1997). La optimización combinatoria consiste en la solución algorítmica de problemas donde se busca minimizar o maximizar el valor de la función objetivo. Los algoritmos de optimización combinatoria resuelven instancias de problemas que son difíciles en general, explorando el espacio de soluciones. De acuerdo a la teoría de la complejidad se presentan diferentes tipos de problemas: P, NP y NP-hard.

Problema P. Este tipo de problema puede ser resulto por un algoritmo determinístico en tiempo polinomial. Problema NP. Este tipo de problema puede ser resulto por un algoritmo no determinístico en tiempo polinomial. Problema NP-hard. Para este tipo de problema no existe solución en tiempo polinomial. Por su complejidad se hace uso de metaheurísticas e hiperheurísticas. (Papadimitriou y Steiglitz, 1998, Talbi, 2009). Estos problemas pueden ser resueltos mediante métodos de optimización.

Los métodos de optimización se dividen en métodos exactos y métodos aproximados (o heurísticos). Los métodos exactos son aquellos que proporcionan una solución óptima de casi cualquier problema, en tiempo polinomial. Algunos de los problemas de optimización que pueden ser resueltos mediante estos métodos son: el problema de la ruta más corta, el problema de árbol de expansión mínima, y la red de flujo máximo entre otros.

Los métodos heurísticos son aquellos que permiten encuentran resultados aproximados en tiempo razonables. Una *heurística* es un procedimiento bien estructurado, desarrollado con base a la experiencia que permite obtener

soluciones aproximadas de un problema sin garantizar la optimalidad (Wetzel, 1983). Dentro de los heurísticos se encuentran las *metaheurísticas* que son algoritmos de propósito general que pueden ser aplicadas para resolver cualquier problema de optimización. Estos algoritmos pueden ser vistos como metodologías generales de alto nivel que se pueden utilizar como una estrategia de guía en el diseño de heurísticas para resolver problemas de optimización específicos (Talbi, 2009). Algunos de los problemas de optimización que pueden ser resueltos mediante estos heurísticos son: el problema del agente viajero (TSP), el problema de redes de distribución de agua (WDN), el problema del ruteo vehicular (VRP) entre otros.

Las metaheurísticas se dividen en dos clases: *basadas en trayectoria*, son aquellas que requieren una solución inicial y en cada paso de la búsqueda la solución actual es reemplazada por una mejor solución (solución vecina) encontrada en su espacio de vecindad. Estos métodos permiten encontrar rápidamente una solución local óptima. Los métodos basados en una solución son: recocido simulado (SA), búsqueda tabú (TS), búsqueda local iterada (ILS), búsqueda local variable (VNS). Las metaheurísticas *basadas en población,* son aquellas que requieren de una población de soluciones, la cual mediante un proceso iterativo se va mejorando. En cada generación del proceso, la población se sustituye por nuevos individuos que fueron creados mediante el uso de ciertos operadores. Estos métodos son: algoritmos evolutivos (EAs), optimización de colonia de hormigas (ACO), optimización por enjambre de partículas (PSO), búsqueda dispersa (SS), evolución diferencial (DE), estrategias evolutivas (ES) (Gendreau & Potvin, 2005;Talbi, 2009;Alba et al., 2013).

Técnicas heurísticas de Optimización

En este apartado se introducen los conceptos más importantes relacionados con las técnicas de optimización aplicadas para solucionar el problema del MDVRPTW.

Las técnicas de optimización son herramientas matemáticas que tienen como objetivo la maximización de beneficios o la minimización de esfuerzos o pérdidas. Puede expresarse como una función objetivo (FO) de varias variables, un proceso de optimización de la FO, se puede definir como el proceso de búsqueda de aquellas variables que minimizan o maximizan el valor de la función.

Clasificación de las Técnicas Metaheurísticas

Cuando se desea resolver un problema de optimización se puede hacer de dos maneras, utilizando un método exacto o mediante la aplicación de métodos heurísticos.

Se puede decir que no existe un solo método para resolver todos los problemas de optimización de forma eficiente, así se han desarrollado una gran cantidad de métodos de optimización para resolver diferentes tipos de problemas.

Una gran variedad de los problemas que se presentan en la práctica no pueden ser resueltos mediante métodos exactos; principlamente por su complejidad, y el tiempo requerido para resoverlo es extremadamente alto, aumentando exponencialmente con el tamaño del problema (Tansini, Urquhart, & Viera, 2001).

Teniendo en cuenta la complejidad de un determinado problema de optimización, este puede ser abordado mediante un método exacto o uno

aproximado. Los métodos exactos consiguen soluciones óptimas y garantizan su optimalidad. Los métodos aproximados (heurísticas y metaheurísticas) generan soluciones prácticas de alta calidad en un tiempo razonable, pero sin garantizar la optimalidad de las soluciones encontradas. Para este proyecto sólo se consideran los métodos aproximados (Heurísticas y Metaheurísticas) para el problema del MDVRPTW.

Heurísticas

Una heurística, se define como "*Un procedimiento simple, a menudo basado en el sentido común, que se supone ofrecerá una buena solución (no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido*" (Zanakins & Evans, 1981)

Las heurísticas se utilizan cuando;

a) No existe un método exacto de resolución;
b) Aunque existe un método exacto, su uso es computacionalmente muy costoso (consume mucho tiempo para ofrecer la solución óptima);
c) Existen limitaciones de tiempo;
d) Se usa como paso intermedio para obtener una solución para la aplicación de otra técnica.

Las heurísticas son una excelente alternativa para resolver problemas de difícil solución pero tienen ciertas características que impiden su uso de forma generalizada:

a) Su diseño depende de la estructura del problema para el cual será diseñado.
b) Si hay modificaciones al problema inicial, hay que adaptar la heurística a estos cambios.

c) Su diseño es específico, aunque pueden ser “trasladados a otro problema”, deben de particularizarse para cada caso.
d) Tienen el inconveniente de que pueden quedarse atrapados en “óptimos locales”.

Las heurísticas encuentran soluciones en un tiempo razonable, son procedimientos que proporcionan soluciones de aceptable calidad mediante una exploración del espacio de búsqueda en instancias de gran tamaño (Rocha, González, & Orjuela, 2011).

Permiten obtener un buen rendimiento con un costo aceptable en una gran cantidad de problemas. Las heurísticas específicas son aquellas diseñadas a medida para resolver un problema concreto y/o instancia y se apoyan en las metaheurísticas para realizar una búsqueda guida que mejora su desempeño; las técnicas metaheurísticas son procedimientos de búsqueda que tampoco garantizan la obtención del óptimo del problema y que también se basan en la aplicación de reglas.

Los métodos heurísticos se justifican por la existencia de problemas de optimización pertenecientes a la categoría denominada NP. Es decir, problemas para los que no existe un algoritmo de resolución que sea polinomial con el tamaño del problema.
Si se demuestra que un problema de optimización pertenece a esta categoría de problemas NP, es práctica habitual, el abordarlo por medios heurísticos de optimización.

Metaheurísticas

Clasificación de las Técnicas Metaheurísticas

Cuando se desea resolver un problema de optimización se puede hacer de dos maneras, utilizando un método exacto o mediante la aplicación de métodos heurísticos.

Se puede decir que no existe un solo método para resolver todos los problemas de optimización de forma eficiente, así se han desarrollado una gran cantidad de métodos de optimización para resolver diferentes tipos de problemas.

Una gran variedad de los problemas que se presentan en la práctica no pueden ser resueltos mediante métodos exactos; principlamente por su complejidad, y el tiempo requerido para resoverlo es extremadamente alto, aumentando exponencialmente con el tamaño del problema (Tansini, Urquhart, & Viera, 2001).

Teniendo en cuenta la complejidad de un determinado problema de optimización, este puede ser abordado mediante un método exacto o uno aproximado. Los métodos exactos consiguen soluciones óptimas y garantizan su optimalidad. Los métodos aproximados (heurísticas y metaheurísticas) generan soluciones prácticas de alta calidad en un tiempo razonable, pero sin garantizar la optimalidad de las soluciones encontradas. Para este proyecto sólo se consideran los métodos aproximados (Heurísticas y Metaheurísticas) para el problema del MDVRPTW.

Las metaheurísticas son estrategias inteligentes para diseñar o mejorar algoritmos heurísticos. El término metaheurística lo introdujo Glover en el año de 1986, al definir una clase de algoritmos de aproximación que combinan

heurísticos tradicionales con estrategias eficientes de exploración del espacio de búsqueda (Blum & Roli, 2003)

Se ha definido a la metaheurística de la forma siguiente*: "Las metaheurísticas son métodos aproximados, diseñados para resolver problemas de optimización combinatoria en los que las heurísticas clásicas no son efectivas. Proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos"* (Osman & Kelly , 1996).

Otra definición es la que proporciona VoB S. (VoB, 2001), conjunta los aspectos más importantes de las metaheurísticas que han señalado otros autores:

"Una metaheurística es un proceso iterativo que dirige y modifica las operaciones de otras heurísticas subordinadas para producir soluciones de alta calidad. Puede manipular una solución única (completa o incompleta) o un conjunto de ellas en cada iteración. El heurístico subordinado puede ser un procedimiento de alto (bajo) nivel, una simple búsqueda local o un método de construcción".

Las metaheurísticas son algoritmos de propósito general que pueden ser aplicados para resolver casi cualquier problema de optimización. En cierto modo, pueden verse como metodologías de alto nivel que pueden ser utilizadas, a su vez, como estrategias guía en el diseño de heurísticas que resuelvan problemas de optimización más específicos.

De acuerdo a esta definición, las metaheurísticas se sitúan conceptualmente por encima de las heurísticas porque guían el diseño de éstas.

Lo que diferencia las heurísticas de las metaheurísticas es que éstas tratan de escapar de óptimos locales orientando la búsqueda en cada momento dependiendo de la evolución del proceso de búsqueda.

El uso de las técnicas metaheurísticas es importante para los problemas de optimización combinatoria: problemas en las que las variables de decisión son enteras (o discretas) en las cuales, el espacio de soluciones está constituido por ordenaciones de valores de dichas variables, aunque pueden ser aplicadas a problemas con variables continuas.

La forma de trabajar de las metaheurísticas es la siguiente: un punto de partida que es una solución (o conjunto de soluciones) que no son óptimas y pueden ser o no factibles. A partir de ésta solución(es), se obtienen otras parecidas de entre las cuales se elige una que satisface alguna restricción (o grupo de ellas) a partir de la cual inicia de nuevo el proceso, el cual se detiene cuando se cumple alguna condición establecida previamente.

Existen muchos criterios de clasificación para las metaheurísticas, algunos ejemplos son:

Basados en naturaleza o no basados en naturaleza: Muchas metaheurísticas están inspiradas en procesos naturales. Algunos ejemplos son los algoritmos evolutivos y los sistemas inmunológicos artificiales (artificial immune systems) en la biología; hormigas (ants), colonias de abejas (bees colonies) y la optimización de enjambre de partículas (particle swarm optimization) de la inteligencia colectiva (swarm intelligence) de diferentes especies; y el recocido simulado (simulated annealing) de la física.

Las que utilizan memoria y las que no: Algunas metaheurísticas no tienen memoria, no utilizan información extraída dinámicamente durante el proceso de búsqueda. Dos técnicas representativas son Greedy Randomized Adaptive Search Procedure (GRASP) y Simulated Annealing (SA). Por otro lado, otras metaheurísticas utilizan memoria que contiene información extraída durante el proceso de búsqueda Tabu Search (TS).

Deterministas versus estocásticas: Una metaheurística determinista resuelve un problema de optimización tomando decisiones deterministas (por ejemplo, TS). Las metaheurísticas estocásticas aplican reglas aleatorias durante la búsqueda (por ejemplo, SA y los algoritmos evolutivos). Si considerásemos la misma solución inicial en los algoritmos deterministas siempre obtendríamos la misma solución final, por contra, los algoritmos estocásticos podrían obtener diferentes soluciones finales con la misma solución inicial. Esta característica debe tenerse en cuenta en la evaluación del rendimiento de los algoritmos metaheurísticos.

Basadas en población o en una única solución **(trayectoria):** Los algoritmos basados en población (por ejemplo, Particle Swarm Optimization (PSO) o los algoritmos evolutivos), manipulan una población entera de soluciones.

Autores como (Michalewicz & Fogel, 2004) han clasificado algunas heurísticas de la forma siguiente como métodos de solución:

- **Métodos constructivos** que se caracterizan por construir una solución definiendo diferentes partes de ella en pasos sucesivos. Constan de un procedimiento que incorpora iterativamente elementos a una estructura, inicialmente vacía, que representa la solución. Algunos ejemplos de éstas son la estrategia voraz (greedy) y los métodos GRASP en su primera fase.
- **Métodos de descomposición**, establecen pautas para resolver el problema dividiéndolo en subproblemas más pequeños, de los que se construye una solución que se obtiene a partir de la solución de cada uno de éstos.
- **Métodos de búsqueda por vecindades (entornos)** establecen estrategias para recorrer el espacio de soluciones del problema transformando de forma iterativa una solución de partida a la que le

realizan modificaciones con iteraciones sucesivas para obtener una solución final. En cada iteración se genera un conjunto de soluciones vecinas que pueden ser la nueva solución en el proceso de optimización. Es una metaheurística importante porque abarca una gran variedad de métodos de **búsqueda local** y **búsqueda global**. En los métodos de **búsqueda local** se parte de una solución inicial e iterativamente se trata de mejorar la solución hasta que no sea posible obtener mejoras. El recocido simulado (Simulated Annealing) la más representativa de este apartado y la búsqueda tabú (Tabu Search)

- **Métodos de reducción**, tratan de identificar alguna característica de la solución que permita simplificar el tratamiento del problema.
- **Métodos Evolutivos,** son procedimientos basados en conjuntos de soluciones, llamados poblaciones, que evolucionan en el espacio de búsqueda con el fin de acercarse a una solución óptima utilizando aquellos elementos que sobreviven en las continuas generaciones de poblaciones. Están los algoritmos genéticos, los algoritmos meméticos y los algoritmos de estimación de distribuciones (EDA).
- Desde hace más de tres décadas se han implementado diferentes métodos de optimización para resolver el problema MDVRPTW. Pero en este trabajo de investigación se implementó una metodología hiperheurística.
- El término “hiperheurístico” se utilizó primera vez en 1996 (Gratch & Chien, 1996) para describir un protocolo que combina varios métodos de inteligencia artificial para la demostración automatizada de teoremas.
- Según (Drake et al., 2019), el término "hiperheurístico" puede definirse como una metodología de búsqueda automatizada de alto nivel que explora un espacio de búsqueda de heurística de bajo nivel (operadores de vecindad o movimiento, o metaheurística) o componentes heurísticos para resolver problemas difíciles.

- Las hiperheurísticas son estrategias de alto nivel que permiten seleccionar o generar heurísticas para resolver problemas de optimización combinatoria (Burke et al., 2019).
- Las hiperheurísticas se clasifican en: *con aprendizaje* y *sin aprendizaje*. *Las hiperheurísticas sin aprendizaje* contienen enfoques que utilizan varias heurísticas (estructuras de vecindario), pero estas las seleccionan de acuerdo a una secuencia predeterminada. Las *hiperheurísticas con aprendizaje* incluyen métodos que cambian dinámicamente la preferencia de cada heurística en función de su rendimiento histórico, guiados por algún mecanismo de aprendizaje (Burke et al., 2019).

- Las hiperheurísticas se clasifican *en constructivas* y *en métodos de búsqueda local*. Las hiperheurísticas constructivas construyen una solución incrementalmente seleccionando heurísticas adaptativas, de un grupo de heurísticas constructivas, en diferentes etapas del proceso de construcción. La hiperheurística de búsqueda local, comienza con una solución inicial completa y selecciona iterativamente, de un conjunto de estructuras de vecindario, las heurísticas apropiadas para dirigir la búsqueda en una dirección prometedora (Swiercz, 2017, Burke et al., 2019).

2.1 Estado del arte

Metaheurísticas aplicadas al MDVRP Y VRPTW

El VRPWT Y MDVRP son variantes del VRP, el primero tiene la característica de tener ventanas de tiempo en las cuales da o recibe atención de los clientes que atiende.

El MDVRP es un problema NP-hard y los métodos exactos no son adecuados para obtener soluciones óptimas. El modelo se caracteriza por tener por tener más de un depósito para atender a sus clientes, éstos y los depósitos están mezclados y se resueve como un sistema de ruteo de vehículos con múltiples depósitos. En ambos, cada vehículo debe salir y regresar al mismo depósito. En el MDVRP existe un gran número de depósitos y el problema radica en determinar qué clientes son atendidos por cuál depósito sin exceder las limitaciones de capacidad, la agrupación se realiza considerando las distancias entre los clientes y los depósitos. Cada depósito requeiere la asignación de clientes y de una flota de vehículos, el objetivo del problema es encontrar rutas para los vehículos que presten el servicio a todos los clientes a un costo mínimo en términos de número de rutas y distancia total del viaje sin violar la capacidad y las limitaciones de tiempo de viaje de los vehículos. MDVRP puede ser visto como un problema de clusterización debido a que arroja un conjunto de programas de vehículos agrupados por depósito. Existen muchas aplicaciones en tiempo real que motivaron la investigación en el campo de los MDVRP, tales como los servicios de mensajería, servicios de emergencia, servicios de taxi y distribución de periódicos. (Geetha, Vanathi, & Poonthalir, 2012).

Para resolver el MDVRP se utilizan dos fases: (Geetha, Vanathi, & Poonthalir, 2012) 1. Fase de agrupamiento, en la cual los clientes son asignados a un depósito.

2. Fase de enrutamiento, en el que la ruta de costo mínimo debe ser identificada dentro de los grupos formados.

Tabla 2. Metaheurísticas aplicadas al problema de MDVRP

Variante	**Investigador(es)**
heurística de ahorros de Clark and Wright	*(Pillac, Gendreau & Laporte, 2007)*
clúster first–route second y route first–cluster second	(Geetha, Vanathi, & Poonthalir, 2012)
búsqueda tabú con restricciones de ruta y capacidad	(Renaud, Laporte, & Boctor, 1996)
Genetic Clustering and Tabu Routing,	(González V. & González A., 2007)
metodología de tres fases basada en la memoria adaptativa y búsqueda tabú	(Crevier, Cordeau, & Laporte, 2007)
algoritmo híbrido con regla de aceptación de Metropolis del recocido simulado en el algoritmo genético	(Chen & Xu, 2008)
algoritmo genético híbrido	(Ho, Ho, Ji, & Lau, 2008)
estrategia de protección de genes en su conjunto; introduce dos operadores de mutación de cambio	(Wang, Sun, & Ren, 2009)
algoritmo genético de objetivos múltiples	(Tang & Yin, 2010)
Heurística descentralizada para el VRP Multi-depósito con entregas divididas.	(Soeanu, Ray, & Debbabi, 2011)

algoritmo genético híbrido para VRP Multi-depósito y Periódico	(Vidal, Crainic, Gendreau, Lahrichi, & Rei, 2011)
enfoque metaheurístico para el VRP Multi-depósito	(Geetha, Vanathi, & Poonthalir, 2012)
algoritmo híbrido de varias fases basado en clusterización	(Luo, Li, & Chen, 2013)
algoritmo de búsqueda tabú con agrupación de clúster variable	(He, Miao, Xie, & Shi, 2014)

Con lo que respecta al *cómputo paralelo*, al revisar la literatura se presentan varios artículos, la mayoría de congresos (Alba, Luque, & Nesmachnow, 2013), aplicándose a diversas áreas del conocimiento como eléctrica, electrónica, asignación de tareas, entre otros.

Entre los trabajos de investigación para el VRP los artículos presentados en esta sección, inician desde el año 1989. En esta revisión solo se consideraron tres artículos, dos actuales y uno anterior al año 2000, que se consideraron como inicio del cómputo paralelo:

- (Groër, Golden, & Wasil, 2011) presenta un algoritmo paralelo con búsqueda local y programación entera; dedica algunos procesadores para resolver instancias de un conjunto de problemas (llamados set covering solvers) a través del método de búsqueda local, mientras que los restantes tratan de combinar rutas de diferentes soluciones para generar nuevas y mejores soluciones (llamados heuristic solvers). Utiliza primero el modelo de ***Movimientos Paralelos***, en el cual saca copia de una solución inicial y la reparte a los nodos esclavos (heuristic solvers) que harán la mejora. Los solvers generan varias soluciones iniciales que son enviadas a los otros nodos restantes (covering solvers) que las combinarán para obtener mejores soluciones, éstas son

envíadas al maestro y de ahí selecciona la mejor y final. En esta parte aplicó ***búsquedas simultáneas con interacciones periódicas***. Trabajó con 3 instancias para el CVRP, cada una de diferente benchmark: G15 de Golden, T385 de Taillard y L25 de Li. Reporta valores de tiempo de ejecución de 100 s para 64 procesadores, 200 s para 32, 400 s y 800 s. No hace mención del speed up.

- (Czech & Czarnas, 2002) trabajó con el algoritmo de recocido simulado para el problema de ruteo vehicular con ventanas de tiempo. Trabaja con el modelo de ***Movimientos paralelos***, el nodo maestro envía copia de la solución inicial a los nodos esclavos. Hay comunicación entre los procesadores esclavos cada cierto número de ciclos para intercambiar su mejor solución encontrada. Los resultados obtenidos fueron aceptables, pero no lograron el óptimo reportado en la literatura. No hace mención del tiempo de ejecución ni del speed up.
- (Malek, Guruswamy, Pandya, & Owens, 1989), aplica la paralelización al algoritmo de recocido simulado para resolver el problema del agente viajero (TSP). Trabajaron con instancias de 25 a 100 ciudades. Utiliza el ***Búsquedas simultáneas con interacciones periódicas*** el cual consiste el nodo maestro asigna una copia de la instancia a cada nodo esclavo, cada nodo esclavo tiene un esquema de enfriamiento, diferente de los otros, para dar solución del problema a través del recocido simulado. Después de un tiempo específico el nodo maestro recibe resultados, los compara y envía a los nodos esclavos la mejor solución encontrada y siguen el proceso, al final de los ciclos asignados, se envían de nuevo los resultados al nodo maestro y se selecciona la mejor solución obtenida. Los resultados obtenidos en la ejecución del algoritmo de recocido simulado en paralelo fueron muy buenos, con respecto al secuencial lo mejoró significativamente en tiempo y costo. Lograron un super spedd up que atribuyen al uso de diferentes esquemas de enfriamiento en cada nodo esclavo.

- En el trabajo de (Kendall & Li, 2013) propusieron una metodología hiperherística para resolver el problema CTSP. Esta hiperheurística consiste en un conjunto de heurísticas de bajo nivel, donde cada agente adopta una heurística (o conjunto de heurísticas) para crear su recorrido y cada agente sabe los recorridos de todos los demás, esto se hace en conjunto con una heurística de alto nivel que se utiliza para seleccionar entre las heurísticas de bajo nivel en cada punto de decisión.
- En el trabajo de (Maashi, Özcan, & Kendall, 2014) presentan una función de elección de selección de aprendizaje basada en hiperheurística para resolver problemas de optimización de objetivos múltiples. El enfoque hiperheurístico controla y combina las ventajas de tres algoritmos evolutivos de múltiples objetivos (NSGAII, SPEA2 y MOGA), utilizándolos como la heurística de bajo nivel. El rendimiento de la hiperheurística es aplicado a un conjunto de pruebas Walking Fish Group y al problema del diseño de la resistencia al choque del vehículo como un problema de objetivos múltiples del mundo real. De acuerdo a los resultados experimentales el enfoque hiperheurístico demuestra efectividad en comparación con el rendimiento de cada ejecución heurística de bajo nivel por sí solo.
- En el trabajo de (Cutillas-Lozano & Giménez, 2014) usaron una hiperheurística para mejorar la determinación de las constantes cinéticas de una reacción química en fase heterogénea. El enfoque hiperheurístico proporciona valores satisfactorios para los parámetros metaheurísticos para el problema de determinar las constantes cinéticas. Los valores de los parámetros cinéticos de la reacción química se pueden determinar integrando la ecuación de la velocidad de reacción. Este es un problema clásico de optimización que puede abordarse con métodos metaheurísticos.

Capítulo 3: DESCRIPCIÓN DEL PROBLEMA

En el desarrollo del algoritmo de agrupamiento como parte de la estrategia de dos fases a la solución del CVRP, se debe considerar lo siguiente;

- El problema de generar K rutas en una población de N clientes tiene un gran cantidad de posibles soluciones, debido a que el número de particiones factibles crece considerablemente al aumentar el número de clientes (N) y el número de rutas a formar (K).
- La restricción de capacidad debe ser considerada al momento de formar los grupos (rutas).
- El conjunto de clientes que atiende un vehículo representa una ruta.
- Los clientes deben ser visitados únicamente por un vehículo.
- Cada vehículo visita a los clientes una sola vez.

3.1 CRITERIOS DE AGRUPAMIENTO

De forma matemática, expresado en conjuntos, el algoritmo de agrupamiento debe resolver la siguiente cuestión: dado un conjunto de datos X, de la forma: $X = \{X1, X2, \bullet \bullet \bullet Xn\}$, hay que encontrar un entero k, tal que $2 < k < n$, y una k-partición de X tal que los conjuntos sean homogéneos, dado un criterio determinado.

El problema surge cuando hay que definir el criterio de agrupamiento. Según (Bezdek, 1981) el método de agrupamiento debe ser ajustado a los datos ya que:

1. No hay ningún criterio de medida de similitud que pueda ser aplicable universalmente.

2. La selección de un criterio determinado es, al menos, parcialmente subjetiva y siempre abierta a debate.

3.2 MODELO MATEMATICO

El modelo que se presenta a continuación es utilizado por el algoritmo de agrupamiento propuesto para resolver el problema de agrupar, satisfaciendo las restricciones para obtener soluciones de calidad y número de rutas reducidos.

$$\sum_{i \in S} d_i x_{ik} \leq C y_k \qquad \forall\, k \in K \qquad (3.1)$$

$$\sum_{k \in K} x_{ik} = 1 \qquad \forall\, i \in S \qquad (3.2)$$

$$x_{ik} \in \{0,1\} \qquad \forall\, i \in S, k \in K \qquad (3.3)$$

$$y_k \in \{0,1\} \qquad \forall\, k \in K \qquad (3.4)$$

Figura 3. 1 Modelo matemático del problema de agrupamiento.

Donde ***K*** es el conjunto de vehículos requeridos para satisfacer la demanda. Este problema, según (Martello & Toth, 1990) es conocido como el problema del empaquetamiento (en inglés Bin Packing Problem BPP) del cual ha sido tomado únicamente el conjunto de restricciones sin función objetivo porque este problema no busca optimizar, busca soluciones iniciales que cumplan el conjunto de restricciones. La desigualdad (3.1) indica que la suma de las demandas de los clientes debe ser menor o igual a la capacidad del vehículo, estableciendo la restricción de capacidad y la restricción (3.2) asegura que un cliente es atendido sólo una vez por un vehículo. Se necesita que los vehículos

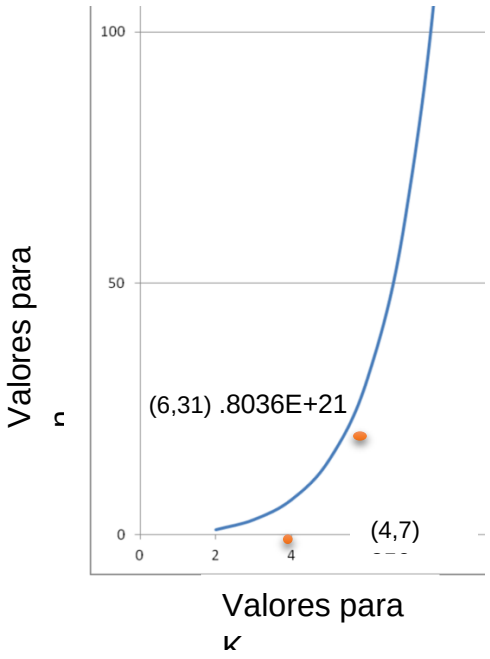
tengan un alto porcentaje de su capacidad para reducir la cantidad de vehículos utilizados.

3.3 ESPACIO DE SOLUCIONES

El espacio de soluciones de un agrupamiento está determinado por todas las permutaciones posibles entre las cuales puede haber soluciones factibles y no factibles; para que una solución sea factible debe satisfacer las restricciones del problema.

3.4 COMPLEJIDAD DEL PROLEMA

Cuando las dimensiones y la complejidad matemática del problema son grandes, surge lo que denomina explosión combinatoria, que son la gran cantidad de soluciones factibles y no factibles que aparecen. En la práctica, las técnicas exactas no pueden ser aplicadas para su solución, debido al gran tiempo de cómputo necesario. (Gallego R., Escobarr Z., & Romero L., 2006).



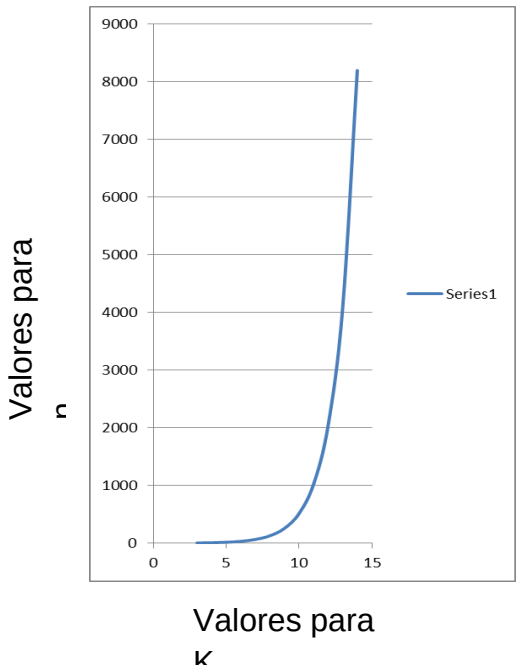


Figura 3.2 Números de Stirling 100 clientes. **Figura 3.3 Números de Stirling 900 clientes.**

En la Figura 3.2 se muestra gráficamente el crecimiento de la cantidad posible de permutaciones con n (clientes) hasta 100, K (grupos) de 8 y en la Figura 3.3 n (clientes) hasta 900, K (grupos) de 15. Se puede observar que el crecimiento es exponencial. En la gráfica que muestra hasta 100 clientes, puede notarse en la línea el rótulo que indica las coordenadas y el valor en ese punto, mientras que, en la gráfica que muestra hasta 900 clientes no están rotulados los valores en la línea por ser valores muy grandes, para ver la tabla que genera estos valores consultar (página **¡Error! Marcador no definido.**).

La Tabla 0.1 muestra algunos valores de la fórmula (3.6) para diferentes combinaciones de *n* y *K*. Como puede observarse en la tabla, el número de particiones factibles crece considerablemente al aumentar el número de elementos (n) y el número de grupos a formar (K).El número de formas en las que se pueden clasificar m observaciones en k grupos es un número de Stirling de segunda especie (Abramowitz & Stegun, 1964).

$$\mathbb{S}_m^{(k)} = \frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\binom{k}{i}i^m \qquad (3.6)$$

Para un conjunto de elementos de tamaño n = 30 y un número de grupos K = 6 la implementación de un método exacto como el método enumerativo puede resultar muy ineficiente para obtener la mejor partición. Con una instancia de tamaño n = 1000 y un número de segmentos K = 20 resultaría muy complicado enumerar todas las posibles combinaciones para resolverla de manera exacta.

CLIENTES	RUTAS	CANTIDAD DE RUTAS FACTIBLE
10 20 30	2 2 2	511 524287 536870911
10 20 30	4 4 4	34105 45232115901 48004081105038304
10 20 30	6 6 6	22827 4306078895384 29931010274694863257 6

Tabla 0.1 Cantidad de posibles permutaciones con n clientes y K Rutas.

El problema se complica aún más por el hecho de que usualmente el número de grupos es desconocido, por lo que el número de posibilidades es la suma de números de Stirling; así, por ejemplo, en el caso de m observaciones tendríamos que el número total de posibles clasificaciones seria:

$$\sum_{j=1}^{m} \mathbb{S}_m^{(j)} \tag{3.7}$$

Que es un número excesivamente grande, por lo que el número de posibles clasificaciones puede ser enorme por ejemplo, en el caso de 25 observaciones, se tiene que:

$$\sum_{j=1}^{25} \mathbb{S}_{25}^{(j)} > 4 \times 10^{18} \tag{3.8}$$

Así, es necesario encontrar una solución aceptable considerando sólo un pequeño número de alternativas; por este motivo el uso de heurísticas es una herramienta que permite la búsqueda de soluciones aceptables del problema (Polya, 1971).

Capítulo 4: DESCRIPCIÓN DEL ALGORITMO DE AGRUPAMIENTO PROPUESTO

El algoritmo de agrupamiento desarrollado hace una combinación de estrategias de los algoritmos CA-VRP (Cruz-Chavez & Martinez-Oropeza, 2011) y el algoritmo de barrido (Sweep) que es una variante del algoritmo de pétalos. Incorporando del algoritmo CA-VRP el uso del primer cliente de cada grupo, seleccionado de forma aleatoria como centroide inicial, la incorporación sucesiva de clientes más cercanos al centroide (mientras se satisfaga la restricción de capacidad); y del segundo algoritmo, agregar al final de la formación de cada grupo, a clientes que se encuentren entre el ángulo menor y el ángulo mayor formados por la localización de los clientes de un grupo, teniendo como vértice de dicho ángulo la localización del depósito. Se muestra el pseudocódigo del algoritmo de agrupamiento propuesto, del que se van describiendo en este capítulo cada uno de los puntos más importantes.

4.1 ALCANCE

El algoritmo de agrupamiento propuesto está diseñado para resolver instancias del CVRP de 1000 o hasta 4000 clientes con un bajo costo de tiempo.

1. **Leer** Datos de entrada

2. **Calcular** distancias[i][j]=$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

3. **Hacer**

4. **Hacer**

centroide= *Elemento de la pila de aleatorios*

Mientras lista_tabu[centroide]==1

lista_tabu[centroide]=1

Vector_solucion[i]=centroide

i++

5 Cliente_mas_cercano_al _centroide

Mientras $(d+\sum d_i) < C$

lista_tabu[centroide]=1

Vector_solucion[i]=centroide

i++

Cliente_mas_cercano_al _centroide

Repetir

6 Cliente_cercano_dentro_angulo

```
        Mientras encuentre_cliente
            Si (d+∑ di ) < C
                lista_tabu[centroide]=1
                Vector_solucion[i]=centroide
                i++
                Cliente_cercano_dentro_angulo
            Fin-si
        Repetir
        Index++
    Mientras index <= N
7   Escribir Archivos de Rutas generadas
```

Figura 4. 1 Pseudocódigo del algoritmo propuesto.

4.2 DATOS DE ENTRADA

En la lectura de los archivos de texto se obtienen los datos del problema, como son: la capacidad de los vehículos, las coordenadas cartesianas x y (de los clientes y el depósito) y la demanda de los clientes; mientras que la cantidad de clientes se calcula al ir leyendo cada archivo.

4.2.1 INSTANCIAS

Una instancia en el problema del CVRP, es una representación concreta y especifica de su clase, comprende todos los elementos incorporados en el problema y generalmente están ligadas a los vehículos y a los clientes.

Las instancias son aquellas características o datos del problema, que se usan en un planteamiento concreto. Las variables involucradas en el planteamiento de una instancia del problema CVRP son:

- El número de clientes.
- La localización de los clientes.
- Los centros de distribución o depósitos.
- La capacidad de los vehículos.
- La demanda de los clientes.

Los archivos de entrada tienen el formato de la primer línea es la capacidad de los vehículos, las siguientes líneas contienen en sus dos primeras columnas las coordenadas rectangulares que determinan la posición del depósito y los clientes en plano cartesiano, la tercer columna es la demanda de cada cliente; la segunda línea contiene los datos del depósito, por eso, la demanda es 0 (cero). La cantidad de clientes es calculada cuando se lee el archivo de entrada, contando el número de líneas y restado las dos primeras; el obtener el número de clientes al leer el archivo de entrada evita que ese dato forme parte de una configuración inicial.

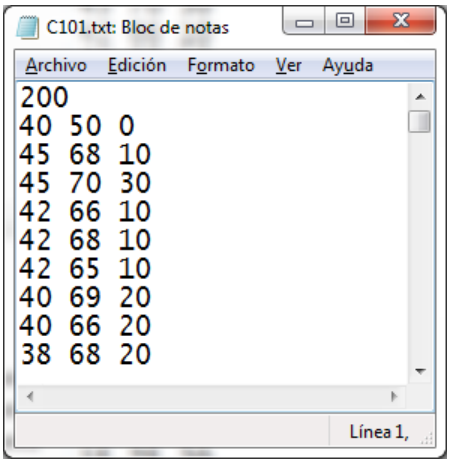
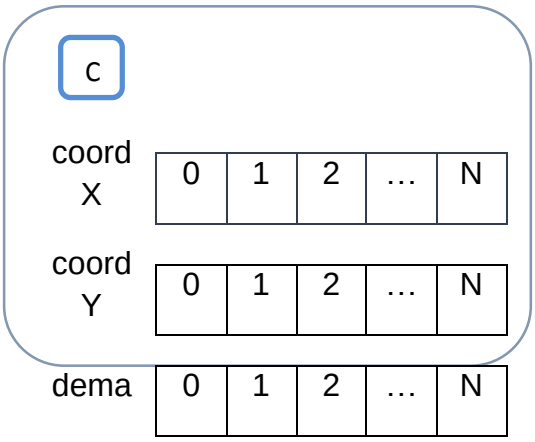


Figura 4. 2 Ejemplo del archivo de entrada.

En la Figura 4. 3 se muestra la estructura que almacena una instancia en donde:



c es la capacidad de los vehículos.

CoordX y cordY son los vectores que contienen las coordenadas x y, de los clientes y el depósito.

Dema es el vector que contiene la demanda de cada cliente.

Figura 4. 3 Estructura de los datos de entrada.

4.3 CRITERIOS PARA LA FORMACION DE LOS GRUPOS

La estructura que almacena los datos de la solución se muestra en la Figura 4. 4, es una estructura de arreglos formada por cuatro vectores y los datos que almacenan se muestran en la Tabla 4. 1.

vectorSolucion	Guarda la serie de índices de los clientes conforme se incorporaron a la solución.
limiteInferior	Guarda la serie de índices de los centroides, es decir los primeros clientes de cada agrupamiento, determinando el inicio de un segmento el vectorSolucion(el inicio de una ruta).
limiteSuperior	Guarda la serie de índices de los últimos clientes de cada agrupamiento, determinando el fin de un segmento el vectorSolucion(el fin de una ruta).
demadaRutas	Guarda la suma de la demanda de los clientes de cada agrupamiento(demanda por ruta).

Tabla 4. 1 Datos que almacenan los vectores de la estructura de solución.

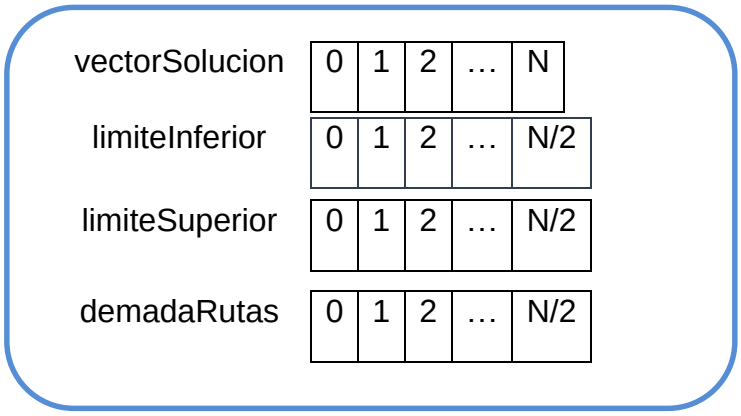


Figura 4. 4 Estructura de la solución.

4.3.1 FORMACIÓN DE LOS AGRUPAMIENTOS

La estrategia para formar los agrupamientos está dividida en tres etapas. La elección de los centroides, la integración de los clientes cercanos al centroide que cumplan con la restricción de capacidad y por último la integración de los clientes dentro del ángulo; enseguida se explica cada una de las etapas mencionadas.

- Primera etapa.

Elección de los centreoides: Es el primer cliente de cada agrupamiento y se determina de forma aleatoria se toma del tope de la pila aleatorio, verificando que la bandera del cliente no esté en 1 en el vectorSolucion; esto determina que se trata de un cliente que no ha sido elegido como parte de la solución y por tanto, no es elemento de ningún agrupamiento.

- Segunda etapa.

Clientes cercanos al centroide: La incorporación de los clientes a un agrupamiento ya iniciado por un centroide. Se hace buscando en la matriz de distancias en que tenga la mínima con respecto a su centroide y es asignado al agrupamiento después de verificar que cumpla con la restricción de capacidad. Una representación gráfica de la formación de los agrupamientos se ve en la Figura 4. 5, en donde los centroides son los centros de cada estrella, que es la forma de un conjunto clientes.

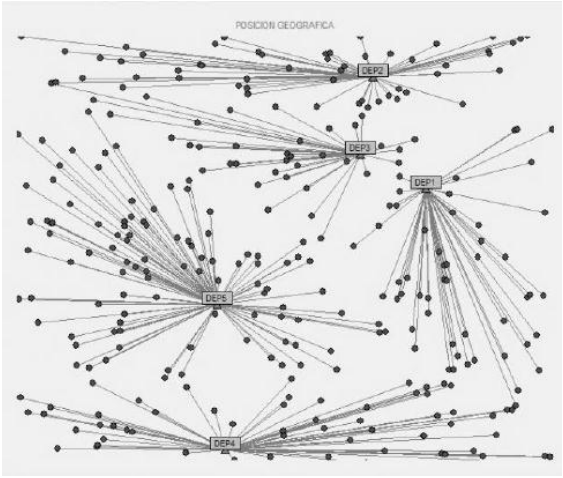


Figura 4. 5 Ejemplo de formación de agrupamientos.

- Tercera etapa.

Clientes dentro del ángulo: Es posible que un agrupamiento se cierre como ya formado porque el siguiente cliente cercano al centroide no cumple con la restricción de capacidad, pero la capacidad sobrante (la capacidad del vehículo menos la suma de las demandas de los clientes) puede permitir la incorporación de un cliente que no sea cercano al, pero se localice en dirección al depósito; esta condición hace que no afecte al costo del recorrido en la segunda fase (ruteo). Para asegurar que un cliente se encuentra en dirección al depósito con respecto al agrupamiento, cuando se incorpora cada cliente al agrupamiento en la primera etapa se van calculando tres valores:

a) AngIzq. Es el ángulo mayor de los clientes de un agrupamiento.
b) AngDer. Es el ángulo menor de los clientes de un agrupamiento.
c) MaximoDist. Es el índice del cliente del agrupamiento que tiene la mayor distancia con respecto al depósito.

Esto tres valores permiten identificar el área que forma un triángulo entre el depósito y el agrupamiento, en la que puede existir un cliente que tenga una demanda menor a la capacidad sobrante de la formación del agrupamiento y como ya se mencionó al estar en el área en dirección al depósito no afecta al costo de recorrido. Los ángulos y la formación del triángulo se muestran marcados con líneas rojas desde el depósito hasta los clientes con el ángulo menor y el ángulo mayor del agrupamiento como también el arco que marca la distancia del cliente más lejano con respecto al depósito.

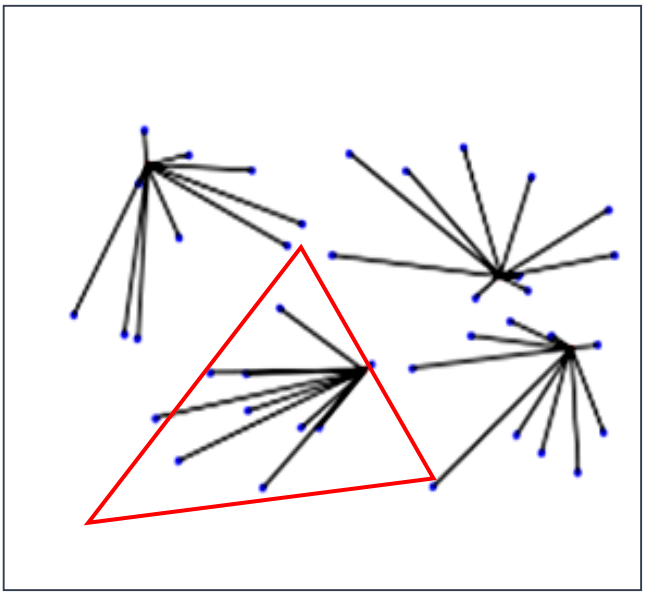


Figura 4. 6 Representación gráfica del ángulo izquierdo y derecho del agrupamiento.

Las tres etapas se repiten de forma iterativa hasta que no existan clientes por asignar a un agrupamiento.

Es importante mencionar que en esta estrategia de formación de los agrupamientos, los vehículos alcanzan la satisfacción de demanda de los clientes atendidos cercana a la capacidad total del vehículo, lo que ayuda a reducir el número de rutas necesarias para atender al conjunto de clientes.

La estrategia de tomar los ángulos menor y mayor de los agrupamientos es una abstracción de la técnica de barrido (sweep) que incorpora a los clientes el ángulo que recorre una recta, para este caso el barrido está delimitado por los ángulos mencionados.

4.4 DATOS DE SALIDA

El resultado de la formación de los agrupamientos. Se imprime en un archivo de salida para su uso en la segunda fase (ruteo), con el número de agrupamientos en la primera línea y el índice de cada cliente en las líneas posteriores, comenzando y terminando la serie de números con 0 (cero) por ser el índice del depósito. Como se muestra en la Figura 4.1

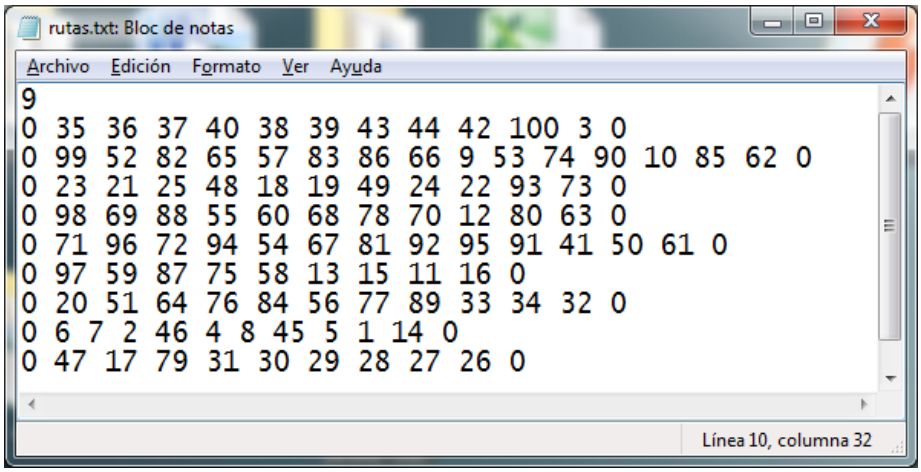


Figura 4.1 Ejemplo de archivo de salida.

Una solución inicial dada por el algoritmo propuesto en su representación gráfica (Figura 4.2), los clientes de cada agrupamiento forman un conjunto de líneas, que inician del centriode al primer cliente incorporado, de ahí al siguiente cliente incorporado y así sucesivamente hasta el último cliente del grupo. En el archivo de salida se agrega al inicio y al final de cada agrupamiento el índice 0 (cero), que es el depósito, porque en la segunda fase del método de solución del VRP usado en esta investigación, los recorridos inician y terminan en el depósito como lo muestra las desigualdades 2.2 y 2.3.

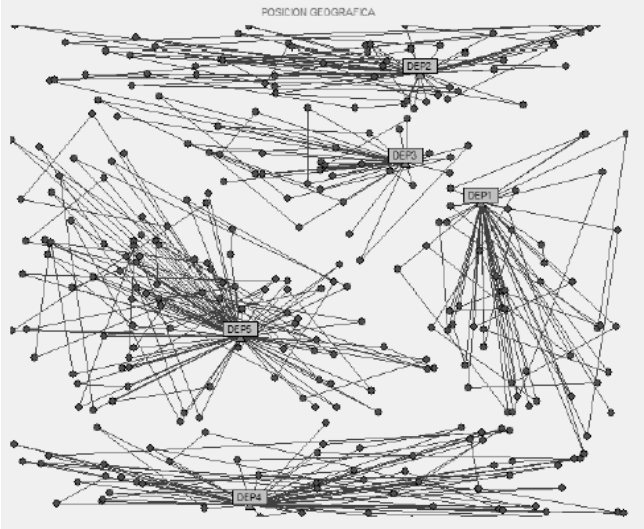


Figura 4.2 Ejemplo de solución inicial.

Capítulo 5: RESULTADOS

Se ejecutó el programa en una computadora personal con procesador AMD Athlon(tm)II Dual-Core M300 a 2.00GHz. y 2.75GB. De memoria RAM.

Se utilizaron instancias de Solomon de 100 clientes y Gehring & Homberger's de 1000 clientes para probarlo en diferentes rangos, obteniendo los resultados que se muestran en las tablas comparativas.

5.1 TABLAS COMPARATIVAS

Al comparar el número de vehículos NV (rutas) que genera algoritmo propuesto con el CA-CVRP (Tabla 5. 1), se pude observar en los tres primeras grupos de instancias C101, R101 y RC101; que se consigue igualar el resultado, siendo estas instancias de 100 clientes y vehículos con capacidad de 200.

BENCHMARK	CACVRP		PROPUESTO	
	NV	TIEMPO	NV	TIEMPO
C101	9	$6*10^{-5}$	9	$1*10^{-3}$
R101	8	$6*10^{-5}$	8	$1*10^{-3}$
RC101	9	$7*10^{-5}$	9	$3*10^{-3}$
RC110_1	94	$6*10^{-3}$	91	$4*10^{-2}$

Tabla 5. 1 Número de rutas generado.

La instancia en donde se consigue reducir en número de vehículos esta marcada con un círculo (Tabla 5. 1), es la instancia RC110_ 1 con 1000 clientes; la cantidad más grande de clientes de esta instancia otorga la posibilidad de la asignación de más clientes a cada agrupamiento, al implementar la estrategia del algoritmo propuesto de incorporar clientes que se encuentren dentro del triángulo formado por el depósito y los ángulos mayor y menor de los clientes de los grupos. El tiempo de ejecución es mayor, pero no significativo al presentar una tasa de crecimiento similar.

En la comparación con otras heurísticas se ejecutó el algoritmo con instancias de Solomon (Solomon, 2013) de 100 clientes con distribuciones: aleatoria R (random en inglés), agrupada C (cluster en inglés) y aleatoria agrupada RC. En la columna NV de la tabla 5.2 se muestra el número de vehículos. Las que seguido de la R inician con el Número 1 tienen una capacidad de 200, las seguido de la R inician con el Número tiene una capacidad de 700 y Así para todas las instancias de Solomon.

Problema	NV	Propuesto	Autores	Problema	NV	Propuesto	Autores
R101	19	**8**	H	R201	4	**2**	HG
R102	17	**8**	RT	R202	3	**2**	RGP
R103	13	**8**	LLH	R203	3	**2**	M
R104	9	**8**	M	R204	2	2	BVH
R105	14	**8**	RT	R205	3	**2**	RGP
R106	12	**8**	M	R206	3	**2**	SSSD
R107	10	**8**	S97	R207	2	2	BVH
R108	9	**8**	BBB	R208	2	2	M
R109	11	**8**	HG	R209	3	**2**	H
R110	10	**8**	M	R210	3	**2**	M
R111	10	**8**	RGP	R211	2	2	BVH
R112	9	**8**	GTA				

Tabla 5. 2 Instancias de Solomon R.

En la Tabla 5.2 pude observarse que el algoritmo propuesto en este tipo de distribución, disminuye el número de vehículos a utilizar en la formación de las rutas.

Los resultados en cuanto al número de vehículos que genera el algoritmo propuesto para el tipo de distribución agrupada, son iguales a los de la heurística RT (Rochat & Taillar, 1995) como se puede ver en la Tabla 5. 3.

Problema	NV	Propuesto	Autores	Problema	NV	Propuesto	Autores
C102	10	**10**	RT	C202	3	**3**	RT
C103	10	**10**	RT	C203	3	**3**	RT
C104	10	**10**	RT	C204	3	**3**	RT
C105	10	**10**	RT	C205	3	**3**	RT
C106	10	**10**	RT	C206	3	**3**	RT
C107	10	**10**	RT	C207	3	**3**	RT
C108	10	**10**	RT	C208	3	**3**	RT
C109	10	**10**	RT				

Tabla 5. 3 Instancias de Solomon C.

En las pruebas en distribución aleatoria distribuida (Tabla 5. 4) se consigue reducir el número de vehículos tanto en las instancia de baja capacidad (RC1..), como en las de alta (RC2..)

Problema	NV	Propuesto	Autores	Problema	NV	Propuesto	Autores
RC101	14	**9**	TBGGP	RC201	4	**2**	M
RC102	12	**9**	TBGGP	RC202	3	**2**	CC
RC103	11	**9**	S98	RC203	3	**2**	CC
RC104	10	**9**	CLM	RC204	3	**2**	M
RC105	13	**9**	BBB	RC205	4	**2**	M
RC106	11	**9**	BBB	RC206	3	**2**	H
RC107	11	**9**	S97	RC207	3	**2**	BVH
RC108	10	**9**	TBGGP	RC208	3	**2**	IKMUY

Tabla 5. 4 Instancias de Solomon RC.

En este párrafo se citan los autores de las heurísticas. BVH (Bent & Van Hentenryck, 2001), BBB (Berger, Barkaoui, & Bräysy, 2001), CC (Czech & Czarnas, 2002), CLM (Cordeau, Laporte, & Mercier, 2000), GTA (Gambardella, Taillar, & Agazzi, 1999), HG (Homberger & Gehring, 1999), H (Homberger J. , 2000), IKMUY (Ibaraki, Kubo, & Masuda, 2001), LLH (Li, Lim, & Huang, "Local Search with Annealing-like Restarts to Solve the VRPTW,", 2001), (Mester & O.), RT (Rochat & Taillar, 1995), RGP (Rousseau, Gendreau, & Pesant), SSSD (Schrimpf, Schneider, Stamm-Wilbrandt , & Dueck, 2000), S97 (Shaw, 1997), S98 (Shaw, 1998), TBGGP (Taillard, Badeau, Gendreau, Geurtin, & Potvin, 1997).

El algoritmo propuesto en comparación con Metaheuristicas en instancias de Gehring & Homberger's (Sintef, 2013) con 1000 clientes, con distribuciones: aleatoria R (random en inglés), agrupada C (cluster en inglés) y aleatoria agrupada RC.

Instancia	Vehículos	Propuesto	Autores	
c1_10_1	100	**95**	GH	
c1_10_2	90	95	VCGP	31-jul-12
c1_10_3	90	95	VCGP	31-jul-12
c1_10_4	90	95	Q	17-apr-13
c1_10_5	100	**95**	RP	25-feb-05
c1_10_6	99	**95**	BC4	12-apr-12
c1_10_7	97	**95**	BC4	12-apr-12
c1_10_8	93	95	VCGP	31-jul-12
c1_10_9	90	95	VCGP	31-jul-12
c1_10_10	90	95	VCGP	31-jul-12

Tabla 5. 5 Gehring & Homberger's c1_10.

La Tabla 5. 5 muestra que en la instancia de tipo agrupado con capacidad de 200 no se consigue mejorar el número de vehículos.

Instancia	Vehículos	Propuesto	Autores	
c2_10_1	30	**28**	LL	
c2_10_2	29	**28**	BC4	05-apr-12
c2_10_3	28	**28**	BC4	05-apr-12
c2_10_4	28	**28**	VCGP	31-jul-12
c2_10_5	30	**28**	VCGP	31-jul-12

c2_10_6	29	**28**	BC4	05-apr-12
c2_10_7	29	**28**	BC4	05-apr-12
c2_10_8	28	**28**	VCGP	31-jul-12
c2_10_9	29	**28**	VCGP	31-jul-12
c2_10_10	28	**28**	VCGP	31-jul-12

Tabla 5. 6 Gehring & Homberger's c2_10.

En la Tabla 5. 6 se puede observar que en este tipo de instancia (agrupada), pero, con capacidad alta (700), se consigue disminuir el número de vehículos, haciendo notar que el algoritmo propuesto proporciona buenos resultados cuando la capacidad de los vehículos es alta.

Instancia	Vehículos	Propuesto	Autores	
r1_10_1	100	**93**	VCGP	31-jul-12
r1_10_2	91	93	VCGP	31-jul-12
r1_10_3	91	93	VCGP	31-jul-12
r1_10_4	91	93	MB2	18-jul-12
r1_10_5	91	93	MB2	18-jul-12
r1_10_6	91	93	VCGP	31-jul-12
r1_10_7	91	93	MB2	18-jul-12
r1_10_8	91	93	MB2	18-jul-12
r1_10_9	91	93	VCGP	31-jul-12
r1_10_10	91	93	MB2	18-jul-12

Tabla 5. 7 Gehring & Homberger's r1_10.

Se pude comprobar en las tablas Tabla 5. 7 y Tabla 5. 8 que el algoritmo produce mejores resultados (menor número de vehículos) cuando la capacidad de los vehículos es alta (instancias r2.), que en este caso el resultado es igual al reportado por las metaheuristicas.

Instancia	Vehículos	Propuesto	Autores	
r2_10_1	19	**19**	BC4	05-apr-12
r2_10_2	19	**19**	VCGP	31-jul-12
r2_10_3	19	**19**	VCGP	31-jul-12
r2_10_4	19	**19**	VCGP	31-jul-12
r2_10_5	19	**19**	BC4	05-apr-12
r2_10_6	19	**19**	VCGP	31-jul-12
r2_10_7	19	**19**	BC4	05-apr-12
r2_10_8	19	**19**	BC4	05-apr-12
r2_10_9	19	**19**	BC4	05-apr-12
r2_10_10	19	**19**	VCGP	31-jul-12

Tabla 5. 8 Gehring & Homberger's r2_10.

En las instancias de tipo aleatorio distribuido se repite el comportamiento del algoritmo de agrupamiento propuesto. En la Tabla 5. 9 se puede ver que con capacidad de 200, no se consigue mejorar.

Instancia	Vehículos	Propuesto	Autores	
rc1_10_1	**90**	92	**VCGP**	**31-jul-12**
rc1_10_2	90	91	VCGP	31-jul-12
rc1_10_3	**90**	91	**VCGP**	**31-jul-12**
rc1_10_4	90	91	VCGP	31-jul-12
rc1_10_5	**90**	91	**VCGP**	**31-jul-12**
rc1_10_6	90	91	VCGP	31-jul-12
rc1_10_7	**90**	91	**VCGP**	**31-jul-12**
rc1_10_8	90	92	VCGP	31-jul-12
rc1_10_9	**90**	92	**VCGP**	**31-jul-12**
rc1_10_10	90	91	VCGP	31-jul-12

Tabla 5. 9 Gehring & Homberger's rc1_10.

Confirmando en la Tabla 5. 10 lo antes mencionado, en este tercer tipo de distribución (aleatoria agrupada) también influye la capacidad de los vehículos, en esta se logra igualar y en algunos casos reducir el número de vehículos, siendo un resultado deseable en esta investigación.

Instancia	Vehículos	Propuesto	Autores	
rc2_10_1	**20**	**18**	**BC4**	**12-apr-12**
rc2_10_2	18	**18**	VCGP	31-jul-12
rc2_10_3	**18**	**18**	**VCGP**	**31-jul-12**
rc2_10_4	18	**18**	VCGP	31-jul-12
rc2_10_5	**18**	**18**	**VCGP**	**31-jul-12**
rc2_10_6	18	**18**	BC4	05-apr-12

rc2_10_7	**18**	**18**	**VCGP**	**31-jul-12**
rc2_10_8	18	**18**	VCGP	31-jul-12
rc2_10_9	**18**	**18**	**VCGP**	**31-jul-12**
rc2_10_10	18	**18**	VCGP	31-jul-12

Tabla 5. 10 Gehring & Homberger's rc2_10.

En este párrafo se citan los autores de las Metaheurísticas. BC4 (Miroslaw & Zbigniew, 2013), BSJ (Johansen, DSolver, 2004), BSJ2 (Johansen, DSolver version2, 2005), GH (Gehring & Homberger, 2001), LL (Li & Lim, 2001), MB (Mester & O.), MB2 (Mester & Braysy), MK (Koch, 2004), PGDR (Prescott-Gagnon, Desaulniers, & Rousseau, 2007), Q (Quintiq),WW (Witoslaw, 2012), VCGP (Vidal, Crainic, Gendreau, & Prins, 2013

Capítulo 6: Conclusiones y trabajos futuros

6.1 CONCLUSIONES

El presente trabajo ha sido con el fin de presentar un método heurístico para la generación una de solución inicial factible y de calidad, al CVRP en una estrategia de dos fases, en la que se minimizó el número de rutas.

Para la construcción de la heurística presentada, se empleó una estrategia de "Agrupar primero, rutear después", la cual es aplicada por el algoritmo propuesto. Se tomó la decisión de agregar estrategias de incorporación de los clientes de los algoritmos CA-CVRP y SWEEP, obteniendo buenos resultados en diferentes tamaños de instancias mostrando mejores resultados en los casos en que la capacidad de los vehículos es alta.

De acuerdo a los resultados obtenidos, la heurística diseñada para la solución de CVRP, es una buena alternativa para la generación de soluciones iniciales a ser evaluadas y mejoradas a través de una heurística en la segunda fase. Dicha heurística está siendo construida en una investigación complementaria a esta, en la que se utiliza la técnica de Recocido Simulado para determinar el ruteo.

Es importante destacar que el algoritmo propuesto es una heurística y no refina, ni optimiza el resultado de los agrupamientos como lo hacen las metaheurístacas. En la comparación con las metaheurísticas en instancias de 1000 se logra igualar y en algunos casos mejorar los resultados en las distribuciones que tiene una capacidad alta; demostrando que es competitivo y eficaz.

6.2 TRABAJOS FUTUROS

Como trabajo futuro se contempla:

- La elección de la métrica de validación la cual se relaciona con la obtención de una agrupación o partición adecuada al problema que se quiere resolver. Es decir, se trata de medir de alguna manera si el resultado aplicar el algoritmo al problema de agrupamiento es eficaz.
-
- Probar el algoritmo propuesto junto con el algoritmo que resuelve el ruteo con la técnica de “recocido simulado” para comparar resultados de las dos fases contra otras heurísticas.
-
- Probar el algoritmo propuesto complementando las dos fases (Agrupar primero – rutear después) de la estrategia de solución al CVRP y compararlo con otras heurísticas.

REFERENCIAS

Abramowitz, M., & Stegun, L. (1964). *Handbook of mathematical functions with formulas, graphs and mathematical tables.* National Bureau of Standards.

Anderberg, M. (1973). *“Cluster Anafysis for Applications‘”.* Kluwer Academic Press.

Bent, R., & Van Hentenryck, P. (2001). *"A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows,"Technical Report CS-01-06.* Department of Computer Science, Brown University.

Berger, M., Barkaoui, M., & Bräysy, O. (2001). *"A Parallel Hybrid Genetic Algorithm for the Vehicle Routing Problem with Time Windows," Working paper.* Canada: Defense Research Establishment Valcartier.

Bezdek, J. (1981). *'Pattern Recognition with Fuzzy Objective Function Algorithms'.* Plenum Press.

Bronshtein, I., & Semendiaev, K. (1973). *Manual de matemáticas para ingenieros y estudiantes. 2da Edición.* Moscú,Rusia: Mir.

Cordeau, J., Laporte, G., & Mercier, A. (2000). "A Unified Tabu Search Heuristic for Vehicle Routing Problems with Time Windows,". *Working Paper CRT-00-03.* Montreal,Canada: Centre for Research on Transportation.

Cruz-Chavez, M., & Martinez-Oropeza, A. (2011). . Método de Agrupamiento no Supervisado para el Problema de Ruteo Vehicular con Restricciones de Capacidad en Vehículos. *CICos 2011, ISBN. 978-607-00-5091-6.*, 148-166.

Csiszár, S. (2007). *“Two-Phase Heuristic for the Vehicle Routing Problem with Time Windows”,.* Acta Polytechnica Hungarita, Vol. 4, No 2.

Czech, Z., & Czarnas, P. (2002). "A Parallel Simulated Annealing for the Vehicle Routing Problem with Time Windows," . *Proc. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, 367-386.

Dantzig, G., & Ramser, J. (1959). The truck dispatching problem. *Management Science vol n 6*, 80-91.

Diaz, A., Glover, F., & Ghaziri, h. m. (1996). *Optimización Heuristica y redes Neuronales.* Madrid: Paraninfo.

Duarte Muñoz, A., Pantrigo Fernández, J., & Gallego Carrillo., M. (2007). *Metaherísticas.* Madrid, España: Dykinson.

Fung, G. (2001). " A Comprehensive Overview of Basic Clustering Algorithms". *IEEE Citeseer*, 1-37.

Gambardella, L., Taillar, E., & Agazzi, G. (1999). "MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows,". En D. Corne, M. Dorigo, & F. Glovewr, *New Ideas in Optimization* (págs. 63-76). London: McGraw-Hill.

Gehring, H., & Homberger, J. (2001). "A Parallel Two-phase Metaheuristic for Routing Problems with Time Windows,". *"A Parallel Two-phase Metaheuristic for Routing Problems with Time Windows,"*, 18,35-47.

González Teofilo, F. (1982). "On the Computational Complexity of Clustering and Related Problems. System Modeling and Optimization". *Lecture Notes in Control and Information Sciences. Volume 38/1982*, 174-182.

Homberger, J. (2000). *"Verteilt-parallele Metaheuristiken zur Tourenplanung,".* Wiesbaden: Gaber.

Homberger, J., & Gehring, H. (1999). *"Two Evolutionary Metaheuristics for the Vehicle Routing Problem with Time Windows,".* VOL. 37, 297-318.

Homberger, J., & Gehring, H. (1999). *"Two Evolutionary Metaheuristics for the Vehicle Routing Problem with Time Windows,".* INFOR, VOL. 37, 297-318.

Homberger, J., & Gehring, H. (2005). “A two-phase hybrid metaheuristic for the vehicle routing problem with time windows”. *European journal of operational research, Vol 162, No 2*, 220-238.

Ibaraki, T., Kubo, M., & Masuda, T. (2001). *"Effective Local Search Algorithms for the Vehicle Routing Problem with General Time Windows,".* Japan: Working Paper, Department of Applied Mathematics and Physics, Kyoto University.

Johansen, B. S. (2004). *DSolver*. BjornSigurdJohansen@hotmail.com.

Johansen, B. S. (2005). *DSolver version2*. BjornSigurdJohansen@hotmail.com.

Koch, M. (2004). *"An approach combining two methods for the vehicle routing problem with time windows".* EURO XX Conference 2004.

Laporte, G. (1991). “The Vehicle Routing Problem: An overview of exact and approximate algorithms”,. *European Journal of Operational Research Vol. 59*, 345-358.

Laporte, G., Reanud, J., & Boctor, F. (1996). “An improved petal heuristic for the vheicle routeing problem”. *The Journal of the Operational*, 329-336.

Li, H., & Lim, A. (2001). *"Large Scale Time-Constrained Vehicle Routing Problems: A General Metaheuristic Framework with Extensive Experimental Results,".* Submitted to Artificial Intelligence Review.

Li, H., Lim, A., & Huang, J. (2001). *"Local Search with Annealing-like Restarts to Solve the VRPTW,".* Working Paper, Department of Computer Science, National University of Singapore,.

LIn, S. (2009). “Applying hybrid meta-heuristics for capacitated vehicle routing problem”,. *Expert Systems with Applications, Volume 36, Issue 2, Part 1*, 1505-1512.

Martello, S., & Toth, P. (1990). *Knapsack problems: algorithms and computer.* John Wiley and Sons.

Marti, R. (s.f.). *Procedimientos Metaheuristicos en Combinación Combinatoria.* Obtenido de www.uv.es/rmaratu/

Melían, B., & Pérez, J. A. (2003). "Metaheuristicas: una visión global". *Revista Iberoamericana de Inteligencia Artificial. No. 19*, 7 - 28.

Mester, D. (2002). *"An Evolutionary Strategies Algorithm for Large Scale Vehicle Routing Problem with Capacitate and Time Windows Restrictions,".* Israel: Working Paper, Institute of Evolution, University of Haifa.

Mester, D., & Braysy, O. (s.f.). *"A new powerful metaheuristic for the VRPTW”.* Israel: working paper, University of Haifa.

Mester, D., & O., B. (s.f.). “Active Guided Evolution Strategies for Large Scale Vehicle Routing Problems with Time Windows”. 32,1593-1614.

Miroslaw, B., & Zbigniew, J. (2013). "A parallel memetic algorithm for the vehicle routing problem with time windows". *3PGCIC .* 8th International Conference on P2P,Parallel, Grid, Cloud and Internet Computing.

Olivera, A. (2004). *“Heurísticas para problemas de ruteo de vehículos”,.* Montevideo, Uruguay: reporte de investigación, Instituto de Computación Universidad de la República, Montevideo.

Osman, I., & Kelly, J. (1996). *Meta-Heuristics: Theory and Applications.* Boston, UEA: Kluwer.

Polya, G. (1971). *How to Solve It: A New Aspect of Mathematical Method.* Princeton, E.U.A., 1971.: Princeton.

Prescott-Gagnon, E., Desaulniers, G., & Rousseau, L.-M. (2007). *A Branch-and-Price-Based Large Neighborhood Search Algorithm for the Vehicle Routing Problem with Time Windows.*

Quintiq. (s.f.). *optimization-world-records.* Obtenido de http://www.quintiq.com/optimization-world-records.aspx.

Rochat, Y., & Taillar, E. (1995). "Probabilistic Diversification and Intensification in Local Search for Vehicle Routing,". *Journal of Heuristics*, 1,147-167.

Rousseau, L., Gendreau, M., & Pesant, G. (s.f.). "Using Constraint-Based Operators to Solve the Vehicle Routing Problem with Time Windows,". *Journal of Heuristics forthcoming*.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt , H., & Dueck, G. (2000). "Record Breaking Optimization Results Using the Ruin and Recreate Principle," . *Journal of Computational Physics*, 159,139-171.

Shaw, P. (1997). *"A New Local Search Algorithm Providing High Quality Solutions to Vehicle Routing Problems,".* Scotland: Working Paper, University of Strathclyde, Glasgow.

Shaw, P. (1998). "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems,". En M. Maher, & J. Puget, *Principles and Practice of Constraint Programming* (págs. 417-431). New York: Springer-Verlag.

Sintef.(2013). *TOP VRPTW*. Obtenido de http://www.sintef.no/Projectweb/TOP/

Solomon, M. (2013). *Benchmark Plroblems and Solutions*. Obtenido de http://web.cba.neu.edu/~msolomon/home.htm

Taillard, E., Badeau, M., Gendreau, M., Geurtin, F., & Potvin, J. (1997). "A Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows,". *Transportation Science*, 31,170-186.

Tan, P., Steinbach, M., & Kumar, V. (2006). *Introduction to Data Mining.* Addison Wesley.

Toth, P., & Vigo, D. (2000). An Overview of Vehicle Routing Problems. Monographs. *The Vehicle Routing Problem* (págs. 1-26). SIAM.

Toth, P., & Vigo, D. (2002). “The Vehicle Routing Problem”. *Society of Industrial and Applied Mathematics (SIAM) monographs on discrete*, 1-23, 109-149.

Vidal, T., Crainic, T., Gendreau, M., & Prins, C. (2013). "A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows". *Computers & Operations Research, Vol. 40, No. 1.*, 475-489.

Witoslaw, W. (2012). *"Design and implementation of parallel programs with shared memory".* Sosnowiec: Master of Science Thesis, University of Silesia.

Apéndice A CÓDIGO FUENTE

```
                    ///  lectura de instancias para el MDVRPTW
                    ///

                    ///              IMPORTANTE
                    ///  NO EXISTE EL CLIENTE 0 CERO TODOS LOS VECTORES COMIENZAN EN 1
                    /// en las instancias grandaes al asignar en la función  ?????? se debe considerar que las ruutas no
llenen al máximo
                    /// aún no he pensado como debe ser el porcentaje para q sea generico
                    ///

                    /// YA asigne por vh  validando capacidad YA ESTÁ
                    ///contruyendo función de validacion que lee de la solucion

                    /// YA  crear los limites de los vh y verificar  q no se violen  ventanas de tiempo


                    ///ahora hacer la funcion de costo para validar carga(ya), ventanas de tiempo  y  costo
                    ///la funcion de costo no debe tener validaciones (quitarlas  solo alimenta datos  al  estructura solucion)

                    ///    en el vector de configuraion de la ruta niciar cada ruta conun  cero 0



                    ///en pr15 se cicla  en Q 0 0




#include<conio.h>///quitar
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
#include<time.h>
#define CLI_MAX 300
#define DEP_MAX 30
#define VH_MAX 180
#define H_MAX 6  ///Número de  heuristicas

// 0(VRP) 1(PVRP) 2(MDVRP) 3(SDVRP) 4(VRPTW) 5(PVRPTW) 6(MDVRPTW) 7(SDVRPTW)

/*
m: number of vehicles
n: number of customers
t: number of days (PVRP), depots (MDVRP) or vehicle types (SDVRP)
The next t lines contain, for each day (or depot or vehicle type), the following information:
D: maximum duration of a route   **vectores de tamaño m
Q: maximum load of a vehicle     **vectores de tamaño m


*/
/**************ESTRUCTURAS*********************************/
typedef struct
{   int m;
    int n;
    int t;
}instancia;  instancia datos;

typedef struct
{   int i;
    float x;
```

```
    float y;
    int d;
    int q;
    int f;
    int a;
    int list;
    int e;
    int l;
    int limit;
}conf_cli;    conf_cli cliente[CLI_MAX + DEP_MAX];

typedef struct
{   int D;///es para  cada vehiculo del deposito
    int Q;///es para  cada vehiculo del deposito
    int i;
    float x;
    float y;
    int d;
    int q;
    int f;
    int a;
    int e;
    int l;
}conf_dep;    conf_dep deposito[DEP_MAX];

typedef struct
{   int ruta[CLI_MAX]; ;///es para  cada vehiculo del deposito
    int tope;
    int sumaCarga;
    int  numCli;
    float TA[CLI_MAX];
    float TRealLlega[CLI_MAX];
    float TRealSale[CLI_MAX];
}pila;    pila s_vehiculo[VH_MAX];

typedef struct //guarda la estructura de la solucion inicial
{   int ruta[CLI_MAX];    //conjunto de clientes que forman una rutae oreden de la configuracion
    int rutaTemp[CLI_MAX];
    float rutaAngulos[CLI_MAX];
    int rutaTW[CLI_MAX];
    float timeArrivo[CLI_MAX];
    float timePartida[CLI_MAX];
    float timeEspera[CLI_MAX];
    float distMasTiAtenVh[DEP_MAX];///***************revisar
    float TiempoFinVh[VH_MAX];
    float TiempAtenRealCli[CLI_MAX];
    int limiteSuperior[DEP_MAX];//marcan los segmentos superiores de ruta en el vector.ruta
    int limiteInferior[DEP_MAX];//marcan los segmentos inferiores de ruta en el vector.ruta
    double costoVh[VH_MAX];//costo de recorrido de un solo vehiculo
    double cargaVh[VH_MAX];
    //int vh;//numero de vehiculos (rutas creadas)
    int numClientes;//cantidad de clientes
    double costoSolucion;//costo solucion de todas las rutas generadas (valor de la Funcion Objetivo)
}solucion;   solucion s_inicial[DEP_MAX];
             solucion s[DEP_MAX];
             solucion s_prima[DEP_MAX];
             solucion s_temp[DEP_MAX];

typedef struct
{   int rutaSolucion[CLI_MAX];
    double costoSolucionGlobal;
}solucionGlobal;    solucionGlobal sol;


///***** LECTURA Y ESCRITURA DE ARCHIVOS(ENTRADAS/SALIDAS)
        void leeArchivos();
        int escribeSol();
        void escribeGraf();

///*****   FUNCIONES DE CALCULO
```

```
    void distCli(instancia datos,float mDistCli[CLI_MAX + DEP_MAX][CLI_MAX + DEP_MAX],conf_cli
cliente[CLI_MAX]);
        //void distDep(instancia datos,float mDistDep[DEP_MAX][CLI_MAX],conf_cli cliente[CLI_MAX],conf_dep
deposito[DEP_MAX],float anguloCliDep[DEP_MAX][CLI_MAX],float mDistCli[CLI_MAX + DEP_MAX][CLI_MAX +
DEP_MAX]);
        //void ordenaTW(conf_cli cliente[CLI_MAX],int listaInicialTW[CLI_MAX],instancia datos);

    void  costo(solucion s_inicial[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],conf_dep deposito[DEP_MAX],
                instancia datos,float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],int
listaInicialTW[CLI_MAX]);
        void costoVh(solucion p[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],
                conf_dep deposito[DEP_MAX],
                instancia datos,float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],
                int indiceDep,int indiceVh);



    void ruteoIniFact(solucion s_inicial[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],conf_dep
deposito[DEP_MAX],instancia datos,float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],int
listaTabu[CLI_MAX]);
        //void ruteoIniFact2(solucion s_inicial[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],conf_dep
deposito[DEP_MAX],instancia datos,float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],float
anguloCliDep[DEP_MAX][CLI_MAX]);
    void ruteoIniFact3(solucion s_inicial[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],conf_dep
deposito[DEP_MAX],
                instancia datos,float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],int
listaInicialTW[CLI_MAX]);
        void ruteoIniFact4(solucion s_inicial[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],conf_dep
deposito[DEP_MAX],
                                                instancia                                datos,float
mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],int listaInicialTW[CLI_MAX]);



///***** ESTRUCTURAS DE VECINDAD
  ///INTRA

    solucion swapVhIntra(solucion p[DEP_MAX],int indiceDep,int indiceVh);///OK
    solucion parAdyacente(solucion p[DEP_MAX],int indiceDep, int indiceVh);///OK

  ///INTER

    void swapVhInter(solucion p[DEP_MAX],int indiceDep,int indiceVh1,int indiceVh2);///Ok
    void insercionVhInter(solucion p[DEP_MAX],int indiceDep,int indiceVh1,int indiceVh2);///OK

  ///DEP

    void swapVhDep(solucion p[DEP_MAX],int indiceDep1,int indiceDep2,int indiceVh1,int indiceVh2);///Ok
    void insercionVhDep(solucion p[DEP_MAX],int indiceDep1,int indiceDep2,int indiceVh1,int indiceVh2);///OK

  ///CON CONOCIMIENTO
      ///swap con unmbral  en la venana y la distacia
      ///swap de vecino con tiempo de espera
      ///


      solucion expulsaVecinoLejano(solucion p[DEP_MAX],int indiceDep, int indiceVh,float
mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX]);
      solucion expulsaVecino(solucion p[DEP_MAX],int indiceDep, int indiceVh);


  ///no clasificadas

      solucion swapVh(solucion p[DEP_MAX],int indiceDep);

///******* FUNCIONES DE VALIDACION

      //int validaTWCli(solucion s_inicial[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],conf_dep
deposito[DEP_MAX],instancia datos,float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX]);
```

```
	int evaluaTwVh(solucion p[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],int indiceDep,int indiceVh);///OK
	int evaluaCargaVh(solucion p[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],int indiceDep,int indiceVh);///OK

///*******FUNCIONES AUXILIARES
	void imprimeRutas(solucion p[DEP_MAX],instancia datos);///Ok


/// Funciones de la Hyperheuristica

		void perturba(solucion p[DEP_MAX],int heuriSel,int indiceDep1,int indiceDep2,int indiceVh1,int indiceVh2);



///*************************************************************************************
int type;

///*****************************************************************************************************************************************
///*****************************************************************************************************************************************
int H[H_MAX];



///*****************************************************************************************************************************************
///*****************************************************************************************************************************************

int main(int argc, char **argv)
{
///VARIABLESDE VALIDACION

   int banderaValidaTWCli=0;


			int k,i,j,w;



   float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX];
			//float mDistDep[DEP_MAX][CLI_MAX];
			//float anguloCliDep[DEP_MAX][CLI_MAX];
   int listaInicialTW[CLI_MAX];
   int listaTabu[CLI_MAX]={0};
			//int listaAncla[CLI_MAX];

			/// variable s temporales
   float sumademadep=0;
			int indiceDep;

   srand(time(NULL));
 ///inicializa vector de heuriticas
   for(i=0;i<H_MAX;i++)H[i]=i;




   leeArchivos();
   distCli(datos, mDistCli, cliente);
			//distDep( datos, mDistDep, cliente, deposito, anguloCliDep, mDistCli);

///asignacion  de clientes a rutas y depositos con una sol inicial factibler0
///*************************************************************************
///*************************************************************************

///*********   Ordenar clientes respecto a la tw
```

```
                        //ordenaTW( cliente, listaInicialTW,datos);

    ruteoIniFact(s_inicial, cliente,deposito, datos, mDistCli, listaTabu);
    ruteoIniFact3(s_inicial, cliente,deposito, datos, mDistCli, listaInicialTW);
          ///falta un  for pra copir s_inicial a s_temp/*************************************************************************

              ///*****************************************************************************************************
//while(){









//}///espacio donde  se tolera las infactibilidades


                        /// PRUEBA DE LAS ESTRUCTURAS
                            // s_temp[4]=swapVhIntra(s_inicial,4,2);
                                // imprimeRutas(s_inicial, datos);
                                // getch();
                            // s_temp[4]=parAdyacente(s_inicial,4,2);
                                // imprimeRutas(s_temp, datos);
                                // getch();

                           //  swapVhInter(s_inicial,4,2,1);
                                // imprimeRutas(s_temp, datos);
                                // getch();

                            // insercionVhInter(s_inicial,4,2,1);



                            // swapVhDep(s_inicial,3,4,2,1);


                            // perturba(s_inicial ,1,1,2,1,2);


                            //insercionVhDep(s_inicial,3,4,2,1);
                                // imprimeRutas(s_temp, datos);
                                // getch();



                            // evaluaTwVh(s_temp, cliente, mDistCli,1,1);
                            // printf( "CARGA %d",evaluaCargaVh(s_inicial,cliente,1,2));




   /// validaTWCli(s_inicial, cliente,deposito, datos, mDistCli);



    costo(s_inicial, cliente,deposito, datos, mDistCli, listaInicialTW);

                        ///  haz una prueba de impresion de las rutas de los depositos
/*
                        for (i=1; i<=datos.t ; i++)
                          {
                        printf("\n");
                            for (j=1; j<=s_inicial[i].numClientes+datos.m; j++)
```

```
                                {

                                   printf("\n %d -- %0.2f",s_inicial[i].ruta[j],s_inicial[i].ruta[j]);
                                  // printf("\nR %d ",s_inicial[i].ruta[j]);

                                }

                               }

                             printf("\n");printf("\n");
                             ///*********
                             ///*********
                             ///*********
                             ///*********
                             ///*********


                              ///   una prueba de impresin de las rutas de los depositos

                             sumademadep=0.0;

                             for (i=1; i<=datos.t ; i++)
                               {
                                  //sumademadep=0.0;
                                 // printf("\n DEP %d    ",i);


                                  for (j=1; j<=s_inicial[i].numClientes; j++)
                                  {
                                 printf(" %d  [%d]",s_inicial[i].ruta[j],cliente[s_inicial[i].ruta[j]].q);
                                    // printf(" %d  [%d]",s_inicial[i].ruta[j],cliente[s_inicial[i].ruta[j]].q);
                                    // sumademadep=sumademadep+cliente[s_inicial[i].ruta[j]].q;

                                  }}
                           */
                           /*

                                printf("\nsumademadep %f \n",sumademadep);
                                  for                    (k=1;                    k<=datos.m;                    k++)printf("(%d){%d}
\n",s_inicial[i].limiteInferior[k],s_inicial[i].limiteSuperior[k]);

                               }

                              printf("\n ");


                             for(k=1;k<=80;k++){

                                   indiceDep=(rand()%datos.t)+1;
                                   parAdyacente(s_inicial,indiceDep);
                                   getch();

                               }
*/
/*
for(j=1;j<=datos.t;j++){
   for(k=1;k<=datos.m;k++){printf(" DEP %d  RUTA %d \n",j,k);
         for( w=(s_inicial[j].limiteInferior[k]+1);w<=(s_inicial[j].limiteSuperior[k]);w++)
            {
               printf("%d\t[%d] \t %4.1f--%4.1f  \t[%d]   \n",s_inicial[j].ruta[w],cliente[s_inicial[j].ruta[w]].e,

s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]],s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]+cliente[s_inicial[j].ruta[
w]].d,
                     cliente[s_inicial[j].ruta[w]].l);
            }}}

   banderaValidaTWCli=validaTWCli( s_inicial, cliente)   ;
     printf(" \nBANDERA %d  \n",banderaValidaTWCli);

*/
```

```
///s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]
   escribeGraf();
   escribeSol();

return (0);
}

void leeArchivos()
{
   FILE *archDatos;//DECLARACION DEL ARCHIVO TIPO FILE
   char NomArchivo[20]; //Para ver el nombre
   int k, j,iValor;
   int cantidad=0;//numero de clientes leidos
   printf("Escriba el nombre  de la instancia  ");
   scanf("%s",NomArchivo);//lee archivo de lectura//se activan cuando te pida el archivo

   if((archDatos = fopen(NomArchivo,"rt"))==NULL)//comprobar que existe el archivo
      printf("No existe el archivo de datos");
     else{
      fscanf(archDatos," %d\t ",&type);//lee el archivo de datos en sus tres columnas x, y, demanda
      fscanf(archDatos," %d\t ",&datos.m);
      fscanf(archDatos," %d\t ",&datos.n);
      fscanf(archDatos," %d\t ",&datos.t);

      for(k=1;k<=datos.t;k++){//lee instancia para cargar
         fscanf(archDatos," %d\t ",&deposito[k].D);
         fscanf(archDatos," %d\t ",&deposito[k].Q);
      }
               for(k=1;k<=datos.n;k++){//lee instancia para cargar las coordenadas
         fscanf(archDatos," %d\t ",&cliente[k].i);
         fscanf(archDatos," %f\t ",&cliente[k].x);
         fscanf(archDatos," %f\t ",&cliente[k].y);
         fscanf(archDatos," %d\t ",&cliente[k].d);
         fscanf(archDatos," %d\t ",&cliente[k].q);
         fscanf(archDatos," %d\t ",&cliente[k].f);
         fscanf(archDatos," %d\t ",&cliente[k].a);

         for(j=1;j<=cliente[k].a;j++)
         fscanf(archDatos," %d\t ",&cliente[k].list);

         fscanf(archDatos," %d\t ",&cliente[k].e);
         fscanf(archDatos," %d\t ",&cliente[k].l);

         ///aqui calcula el tiempo limite de ser atendido dentro de la ventana
         cliente[k].limit =cliente[k].l - cliente[k].d;

      }

      for(k=1;k<=datos.t;k++){//lee instancia para cargar las coordenadas
         fscanf(archDatos," %d\t ",&deposito[k].i);
         deposito[k].i=k;
         fscanf(archDatos," %f\t ",&deposito[k].x);
         fscanf(archDatos," %f\t ",&deposito[k].y);
         cliente[k+datos.n].i=k+datos.n;
         cliente[k+datos.n].x=deposito[k].x;
         cliente[k+datos.n].y=deposito[k].y;
         fscanf(archDatos," %d\t ",&deposito[k].d);
         fscanf(archDatos," %d\t ",&deposito[k].q);
         fscanf(archDatos," %d\t ",&deposito[k].f);
         fscanf(archDatos," %d\t ",&deposito[k].a);
         fscanf(archDatos," %d\t ",&deposito[k].e);
         fscanf(archDatos," %d\t ",&deposito[k].l);
      }

      fclose(archDatos);
      // printf("proceso realizado con exito");
      }
}

void distCli(instancia datos,float mDistCli[CLI_MAX + DEP_MAX][CLI_MAX + DEP_MAX],conf_cli cliente[CLI_MAX])
```

```
{

int i,j;//j son los clientes
float xx,yy;

          // printf("\n  %d  ",s_actual.numClientes);
   for(i=1;i<=datos.n + datos.t;i++)

      {
      for(j=i+1;j<=datos.n + datos.t ;j++)
      {
         xx=(cliente[i].x-cliente[j].x)*(cliente[i].x-cliente[j].x);
         yy=(cliente[i].y-cliente[j].y)*(cliente[i].y-cliente[j].y);
         mDistCli[j][i] = mDistCli[i][j] = sqrt(xx + yy );
         //printf("\n  %d-%d  %f ",i,j,mDistCli[i][j]);
      }
     }
}

///***********************************************
/// esta función hace los agrupamientos por depósito  asignando el cliente más lejano a todos los
///depositos al deposito más cercano a él


void ruteoIniFact(solucion s_inicial[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],conf_dep
deposito[DEP_MAX],instancia datos,float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],int
listaTabu[CLI_MAX]){
   int i,j,k,posfil,poscol;
   int asignados=0;
   float max=0,min;
   float mDistDepTemp[DEP_MAX][CLI_MAX];
   int sumDemDep[DEP_MAX]={0};
   int cima[DEP_MAX];

   ///
   ///PRUEBA PARA VALANCEAR LA CARGA DE LOS DEPOSITOS
   ///
   ///

   int sumaDemandasCli=0;
   int demaProm= 0;
   int capDep =0;
   int  mayor=0;
   int contador=0;

   ///buscala demanda mas alta de los clientes
   for (i=1; i<= datos.n; i++)
      {
      sumaDemandasCli+=cliente[i].q;

         if (mayor<cliente[i].q)
            mayor=cliente[i].q;

      }

   ///demaProm=(sumaDemandasCli/datos.t+((deposito[1].Q *datos.t -  sumaDemandasCli/datos.t  )/2 ) );
   capDep =deposito[1].Q*datos.m;

   ///es la demanda promedio  para cada deposito
   demaProm=sumaDemandasCli/datos.t;
   printf("******DEMACLI *****\n%d  %d\n  %d\n  **%d\n ",sumaDemandasCli,demaProm, capDep, mayor);

   ///demaProm=(demaProm+capDep)/2;
   demaProm=(demaProm+mayor);///requiere ajuste
   printf("****** %d\n ",demaProm);

   ///copia a una matriz temp
   for (i=1; i<= datos.t; i++)
      {
       cima[i]=1;
```

```
      sumDemDep[i]=0;
      for (j=1; j<=datos.n; j++)
        {
         mDistDepTemp[i][j]=mDistCli[i+datos.n][j];
        }
     }
        //printf("prueba %f  ",mDistDepTemp[2][13]);



   /// busca el mayor es decir el clinte mas lejano a todos los depositos
   while(asignados<datos.n)
   {
         posfil=0;poscol=0,max=0;

         for (i=1; i<=datos.t ; i++)
         {
            for (j=1; j<=datos.n; j++)
            {

               if (mDistDepTemp[i][j] > max)
               {
               max=mDistDepTemp[i][j] ;
               // posfil=i;
               poscol=j;//identifica a un cliente
               }
            }
         }

         ///busca al deposito más cercano del cliente q fue encontrado como más lejano
         min=max+1;
         for (i=1; i<=datos.t; i++)
         {
            if (mDistDepTemp[i][poscol] < min && mDistDepTemp[i][poscol] > 0.0)
            {
            min=mDistDepTemp[i][poscol] ;
            posfil=i;//identifica a un deposito
            }


         }

            //printf("\nMaxximo  %f  %d  %d",mDistDepTemp[posfil][poscol],posfil,poscol);

            //getch();

         ///asigna  a lista tabu
         /*
                                    for (i=1; i<=datos.t ; i++)
                                 {
                                    for (j=1; j<=datos.n; j++)
                                    {

                                       printf("%2.0f\t",mDistDepTemp[i][j]);
                                    }
                                    printf("\n ");
                                 }



                     printf("\nselec %d %d\n",posfil,poscol);
         */

         ///hacer una simulacion de este proceso (video)
               ///**** aqui una modificacio de porcentaje  para equilibrar carga
               ///***************************este valor debe ajustarse
         if(sumDemDep[posfil]+cliente[poscol].q    <=  demaProm)//          if(sumDemDep[posfil]+cliente[poscol].q    <=
((deposito[posfil].Q)*datos.m)  - 190)
```

```
            { // printf("\nSDD %d   %d   CLI q %d %d DQ %d   %d",posfil,sumDemDep[posfil],poscol,cliente[poscol].q
,posfil,(deposito[posfil].Q)*datos.m);
                asignados++;
                // printf("\ncdem %d  %d ",poscol,cliente[poscol].q);
                sumDemDep[posfil]=sumDemDep[posfil]+cliente[poscol].q ;
                listaTabu[poscol]=posfil;/// su indice es el cliente su valor es el deposito al que se asigno
                for (i=1; i<=datos.t ; i++)///aqui  quitamos las distancias del cliente para que ya no sea seleccionado
                {
                mDistDepTemp[i][poscol]=0.0;
                }
            }
            else
            {
                printf("\nQ %d %d ",posfil,poscol);
                for (j=1; j<=datos.n; j++)
                {
                mDistDepTemp[posfil][j]=0.0;
                // printf("\nCERO %f ",mDistDepTemp[posfil][j]);
                }
            }

  }

                              /*   for (i=1; i<=datos.t ; i++)
                              {
                                 for (j=1; j<=datos.n; j++)
                                 {

                                    printf("%2.0f\t",mDistDepTemp[i][j]);
                                 }
                                 printf("\n ");
                              }
                              */
                /*impresion de prueba
                for (i=1; i<=datos.n; i++)
                  {
                  printf("\nTabu %d %d\n",i,listaTabu[i]);

                  }
                 */

///*************************************
///llena las rutas de los depositos


   for (i=1; i<=datos.n; i++)
     {
      s_inicial[listaTabu[i]].rutaTemp[cima[listaTabu[i]]]=i;
      cima[listaTabu[i]]=cima[listaTabu[i]] + 1;
     }

   for (i=1; i<=datos.t ; i++)
     {
      s_inicial[i].numClientes = cima[i]-1;
     }
  ///rutaTemp[]   tiene los clientes por dep


            ///  haz una prueba de impresin de las rutas de los ddepositos
            for (i=1; i<=datos.t ; i++)
              {
                printf("\n[dep %d] ",i);
                for (j=1; j<=s_inicial[i].numClientes; j++)
                {

                   printf(" %d ",s_inicial[i].rutaTemp[j]);contador++;
                }

              }
                  printf("\n ----verifica el num de  clientes \n%d   %d \n",datos.n,contador);
```

```
                printf("\n\n  \n  \n");
               /// temporal para ver  como se lleno cada dep , es la suma de la demanda de los clientes


                                for (i=1; i<=datos.t ; i++)
                                        {contador=0;
                                          printf("\n[dep %d] ",i);
                                          for (j=1; j<=s_inicial[i].numClientes; j++)
                                          {

                                             contador=contador +cliente[s_inicial[i].rutaTemp[j]].q;
                                          }
                                printf(" %d ",contador);
                                        }
                                 printf("\n\n  \n  \n");

}
///*****************************************
/// Esta funcion agrupa las rutas de acuerdo a las ventansa de tiempo  de momento
///distriye  la misma cantidad de clientes a cada ruta

void  ruteoIniFact3(solucion s_inicial[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],conf_dep deposito[DEP_MAX],
                instancia datos,float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],int listaInicialTW[CLI_MAX])
{
int i,j,a,b,t,w,k,salto=0;
float t2;
int listaAsignados[DEP_MAX][CLI_MAX];
int asignados=0;


   ///crea  una lista de las ventanas de tiempo de los clientes  por dep
   for (i=1; i<=datos.t ; i++)
     {
       for (j=1; j<=s_inicial[i].numClientes; j++)
       {
          s_inicial[i].rutaTW[j]=cliente[s_inicial[i].rutaTemp[j]].limit;
       }
     }
//impresion de prueba
                        ///verifica la salida
                        for (i=1; i<=datos.t ; i++)
                              {
                                for (j=1; j<=s_inicial[i].numClientes; j++)
                                {
                                   printf("%d\t",s_inicial[i].rutaTW[j]);
                                }printf("\n");
                              }


   ///ordena de acuerdo a TW aun son vectores de clientes por dep
   for(i=1; i<=datos.t ; i++)
   {
       for(a=1; a<s_inicial[i].numClientes;++a)
          for(b=(s_inicial[i].numClientes); b>=a;--b)
          {
             if(s_inicial[i].rutaTW[b-1] > s_inicial[i].rutaTW[b])
             {
             /*intercambio de los elementos*/
             t=s_inicial[i].rutaTW[b-1];
             t2=s_inicial[i].rutaTemp[b-1];
             s_inicial[i].rutaTW[b-1]=s_inicial[i].rutaTW[b];
             s_inicial[i].rutaTemp[b-1]=s_inicial[i].rutaTemp[b];
             s_inicial[i].rutaTW[b]=t;
             s_inicial[i].rutaTemp[b]=t2;
             }
          }
   }
/*impresion de prueba*/
                         ///  haz una prueba de impresin de las rutas de los ddepositos
                        printf("prueba de impresin de las rutas de los ddepositos");
```

```
                        for (i=1; i<=datos.t ; i++)
                          {


                            printf("\n[dep %d] ",i);
                            for (j=1; j<=s_inicial[i].numClientes; j++)
                            {

                               printf(" %d ",s_inicial[i].rutaTemp[j]);
                            }

                          }
                              printf("\n");

 printf("prueba de impresin de las rutas de los ddepositos VENTANAS\n");
                        for (i=1; i<=datos.t ; i++)
                          {
                            for (j=1; j<=s_inicial[i].numClientes; j++)







                            {
                               printf("%d\t",s_inicial[i].rutaTW[j]);
                            }printf("\n");
                          }

///cada dep tiene un numero de clientes difente  s_inicial[i].numClientes
/// las rutas iniciales por dep estan en   s_inicial[i].rutaTemp[k];

/// la ruta inicial debe estar en     s_inicial[i].ruta[w]

   ///  asigna clientes a las rutas

                              /// esta prueba asignaba de acuerdo al la ventana



                            salto=((datos.m));
                              for(i=1; i<=datos.t ; i++)
                                {
                                       //for (j=1; j<=datos.m; j++){
                                           s_inicial[i].ruta[1]=0;                          s_inicial[i].limiteInferior[1]= 1 ;


                                          for (k=1,j=1,w=2; w<=datos.m+s_inicial[i].numClientes; k=k+salto)
                                          {
                                            if(k>s_inicial[i].numClientes)
                                                {
                                                j++;
                                                k=j-salto;
                                                s_inicial[i].ruta[w]=0;s_inicial[i].limiteInferior[j]=   w   ;s_inicial[i].limiteSuperior[j-
1]= w -1;
                                                w++;
                                                continue;
                                                }
                                             s_inicial[i].ruta[w]   =s_inicial[i].rutaTemp[k];
                                             w++;
                                          }s_inicial[i].limiteSuperior[j]= w -1;
                                     //  }
                                }


                                ///*****************************************************************************************
                                              printf("\n");
```

```
                                                    for (i=1; i<=datos.t ; i++)
                                                      {
                                                        for (j=1; j<=s_inicial[i].numClientes+datos.m; j++)
                                                        {
                                                           printf("%d\t",s_inicial[i].ruta[j]);
                                                        }printf("\n");printf("\n");
                                                      }


                                                        printf("LIMITES\n");


                                                    for (i=1; i<=datos.t ; i++)
                                                      {
                                                        for (j=1; j<=datos.m; j++)
                                                        {

printf("[Inf]%d\t[Sup]%d\t",s_inicial[i].limiteInferior[j],s_inicial[i].limiteSuperior[j]);
                                                        }printf("\n");printf("\n");
                                                      }




///ya calcule los limites al construir las rutas

   ///para el calculo de los límites se determina con saltos  de  clientes / vhs
   ///esto porque las rutas son del mismo tamaño inicialmente , asi se les asignaron
   /// los clientes

/*
   for(i=1; i<=datos.t ; i++)
   {
      salto=s_inicial[i].numClientes/datos.m;
      w=1;
      for (k=1; k<=datos.m; k++,w=w+salto)
      {
         //printf("---->%d  %d  %d  \n",i,k,w);
         s_inicial[i].limiteInferior[k]= w ;
         if(k>1) s_inicial[i].limiteSuperior[k-1]= w-1 ;
      }
            ///esta linea es para el último limite superior de cada deposito
   s_inicial[i].limiteSuperior[k-1]= s_inicial[i].numClientes ;

   }
   ///Ya establecidos los limites se debe verificar que no se violen las restriccione
   ///bb y de ser necesario reparar*/

getch();
}


///*****************************************************************************************
void  ruteoIniFact4(solucion s_inicial[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],conf_dep deposito[DEP_MAX],
                instancia datos,float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],int listaInicialTW[CLI_MAX])
{
int  i,j,k,m,pos,pos2,pos3,pos0;
int clienteTemp,contador=0;
//int sumaRutaTemp[datos.t];
int anterior[datos.m];///******este vector puede ser cambiado por variable
float menor=0;
  int puntero;                              int x;//variable tempral para controlar el numero de clientes que swe agrega acada
pila
```

```
for(i=1; i<=datos.t ; i++)
   {
      contador=0;
      puntero=1;///
      for (j=1; j<=datos.m; j++)
      {
         s_inicial[i].limiteInferior[j]=puntero;///
         s_inicial[i].ruta[puntero]=0;///
         puntero++;///
            ///iniciar  los valores de la pila  para cada dep
            for (k=1; k<=s_inicial[i].numClientes; k++)s_vehiculo[j].ruta[k] =0;
            s_vehiculo[j].tope=0;
            /// añadir suma de carga
            x=0;
///*************                                                    PRIMER          CLIENTE          DE       CADA          VH
*****************************************************************************************
            ///busca en el vector rutatemp el cliente mas cercano al dep (dist + tw
            ///recorre el vector buscando el primer valor diferente de 0
            for (m=1; m<=s_inicial[i].numClientes; m++)
            {
               if(s_inicial[i].rutaTemp[m]!=0)
               {
                  clienteTemp=s_inicial[i].rutaTemp[m];
                  //  y el indice ? de clientetemp
                  pos0=m;
                  //provoca un break
                  m=s_inicial[i].numClientes+1;
               }
            }
            clienteTemp =s_inicial[i].rutaTemp[pos0];

            ///busca al cliente mas cercano al dep y tw baja
            menor= cliente[clienteTemp].e + mDistCli[datos.n +j][clienteTemp];

            for (m=1; m<=s_inicial[i].numClientes; m++)
            {

                  clienteTemp=s_inicial[i].rutaTemp[m];
                  if(((cliente[clienteTemp].e + mDistCli[datos.n +j][clienteTemp]) <= menor) && clienteTemp != 0)
                  {
                  menor= (cliente[clienteTemp].e + mDistCli[datos.n +j][clienteTemp]);
                  pos= m;
                  }
            }

               ///asigna el primer cliente de cada ruta

                anterior[j]=s_inicial[i].rutaTemp[pos];
                s_vehiculo[j].ruta[1] =anterior[j];
                s_vehiculo[j].tope++;
                s_vehiculo[j].sumaCarga=cliente[anterior[j]].q;
                contador++;
                s_inicial[i].ruta[puntero]=s_vehiculo[j].ruta[1];///
                puntero++;///


               ///tiempos para las tw
            /*   if (cliente[anterior[j]].e>mDistCli[datos.n +j][anterior[j]])
                     s_vehiculo[j].TRealLlega[1]=cliente[anterior[j]].e;
                else
                   s_vehiculo[j].TRealLlega[1]=mDistCli[datos.n +j][anterior[j]];

               s_vehiculo[j].TRealSale[1]=s_vehiculo[j].TRealLlega[1]+cliente[anterior[j]].d;
               s_vehiculo[j].TA[1]=s_vehiculo[j].TRealSale[1];
            */
                              printf("\n CLI %d \n TOP  %d\n SC %d\n TA %f\n TRL %f\n TRS %f",
                              s_vehiculo[j].ruta[1],s_vehiculo[j].tope,s_vehiculo[j].sumaCarga
                              ,s_vehiculo[j].TA[1],s_vehiculo[j].TRealLlega[1],s_vehiculo[j].TRealSale[1]);
printf("  sumac %d  %d\n ",j,s_vehiculo[j].sumaCarga);
printf("  conta %d  %d\n ",j,contador);
```

```
                /// elimina al cliente del vector temp
                s_inicial[i].rutaTemp[pos]=0;

///**********                                                segundo          CLIENTE          DE          CADA          VH
*******************************************************************************

pos2=-1  ;

printf("\n                    ANT %d",anterior[j]);
               while(s_vehiculo[j].sumaCarga<=(deposito[j].Q)*4/8  &&  contador<s_inicial[i].numClientes  )///conjuto  de
condicion del vh  x<3 && contador<=s_inicial[i].numClientes

              {

                 ///recorre el vector buscando el primer valor diferente de 0
                  for (m=1; m<=s_inicial[i].numClientes; m++)
                  {
                     if(s_inicial[i].rutaTemp[m]!=0)///agegagar condicionepara validar al cliente
                     {
                        clienteTemp=s_inicial[i].rutaTemp[m];
                        //  y el indice ? de clientetemp
                        pos2=m;
                        //provoca un break
                        m=s_inicial[i].numClientes+1;
                     }
                  }
                  clienteTemp =s_inicial[i].rutaTemp[pos2];



                ///*************************************************************

                /// aqui la distancia es del cliente anterior  no del deposit*****************************************************
                  menor= (cliente[clienteTemp].e + mDistCli[anterior[j]][clienteTemp]);
                  for (m=1; m<=s_inicial[i].numClientes; m++)
                  {
                       clienteTemp=s_inicial[i].rutaTemp[m];
                       pos3=pos2;
                       ///*********agragar a la condicion TRsalida del anterior
                       if(((cliente[clienteTemp].e + mDistCli[anterior[j]][clienteTemp]) < menor) && clienteTemp !=0)
                       {

                        //faltan validaciones
                        menor= (cliente[clienteTemp].e + mDistCli[anterior[j]][clienteTemp]);
                        pos3= m;
                       }
                  }

x++;
                  ///asigna el SEGUNDO cliente de cada ruta
                  anterior[j] =s_inicial[i].rutaTemp[pos3];
                  s_vehiculo[j].tope++;
                  s_vehiculo[j].ruta[s_vehiculo[j].tope]=anterior[j];
                  s_vehiculo[j].sumaCarga=s_vehiculo[j].sumaCarga+cliente[anterior[j]].q;
                  s_inicial[i].rutaTemp[pos3]=0;
                  contador++;

                  s_inicial[i].ruta[puntero]=anterior[j];///
                  puntero++;///



printf("\n                    ANT %d",anterior[j]);

                             /*

                                 if (cliente[anterior[j]].e>mDistCli[datos.n +j][anterior[j]])
                                    s_vehiculo[j].TRealLlega[1]=cliente[anterior[j]].e;
                                 else
                                    s_vehiculo[j].TRealLlega[1]=mDistCli[datos.n +j][anterior[j]];
```

```
                                s_vehiculo[j].TRealSale[1]=s_vehiculo[j].TRealLlega[1]+cliente[anterior[j]].d;
                                s_vehiculo[j].TA[1]=s_vehiculo[j].TRealSale[1];
                                printf("\n CLI %d \n TOP  %d\n SC %d\n TA %f\n TRL %f\n TRS %f",
                                      s_vehiculo[j].ruta[1],s_vehiculo[j].tope,s_vehiculo[j].sumaCarga[1]
                                      ,s_vehiculo[j].TA[1],s_vehiculo[j].TRealLlega[1],s_vehiculo[j].TRealSale[1]);

                               */


                 printf("  sumac %d  %d\n ",j,s_vehiculo[j].sumaCarga);
                 //getch();
                }

//prueba de imp de vector temp de rutas . en el que se van marcando los cliente seleccionados
for (m=1; m<=s_inicial[i].numClientes; m++)printf("\n   %d -> %d ",m,s_inicial[i].rutaTemp[m]);
for (m=1; m<=s_inicial[i].numClientes; m++)printf("\n  vh %d  %d ",j,s_vehiculo[j].ruta[m]);

//getch();
       s_inicial[i].limiteSuperior[j]=puntero-1;///
       }///fin for nivel vh



for (j=1; j<=datos.m; j++)
   for (m=1; m<=s_vehiculo[j].tope; m++)
   printf("\nPILA DEP %d, VH %d  CLI %d",i,j,s_vehiculo[j].ruta[m]);




   }///fin for nivel dep

/// contruir los limites
/*for(i=1; i<=datos.t ; i++)
   {
      puntero=1;
      for (j=1; j<=datos.m; j++)
      {   s_inicial[i].limiteInferior[j]=puntero;
          s_inicial[i].ruta[puntero]=0;
          puntero++;
          for (m=1; m<=s_vehiculo[j].tope; m++)
          {
            s_inicial[i].ruta[puntero]=s_vehiculo[j] .ruta[m];
            puntero++;
          }
          s_inicial[i].limiteSuperior[j]=puntero-1;
      }
   }

*/

/// al construir los limetes registrar el nuevo valor de numero de client



 printf("\n");


                                                for (i=1; i<=datos.t ; i++)
                                                  {
                                                    for (j=1; j<=s_inicial[i].numClientes+datos.m; j++)
                                                    {
                                                       printf("%d\t",s_inicial[i].ruta[j]);
                                                    }printf("\n");printf("\n");
                                                  }
```

```
                                                printf("LIMITES\n");


                                              for (i=1; i<=datos.t ; i++)
                                                {
                                                  for (j=1; j<=datos.m; j++)
                                                  {

printf("[Inf]%d\t[Sup]%d\t",s_inicial[i].limiteInferior[j],s_inicial[i].limiteSuperior[j]);
                                                  }printf("\n");printf("\n");
                                                }



}


///*************************************************************************************************************************


///revisar parametros  enviar sólo los necesdarios

void  costo(solucion s_inicial[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],conf_dep deposito[DEP_MAX],
            instancia datos,float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],int listaInicialTW[CLI_MAX])
{
   float suma,sumadist,sumataten,sumadistTotal,distmastaten,tespera,sumEspera;
   int i,j,k,w;
   int clientes;
   float solucion =0;

   //printf("\n        COSTO \n");
   //printf("\n");
   for(j=1;j<=datos.t;j++)
      {

      for(k=1;k<=datos.m;k++)/// k inicia en 2 porque saltamos el marcador de inicio de ruta que  es el valor cero
         {        ///   cargas
         suma=0;///para carga
         sumataten=0;
         distmastaten=0;
         sumadist=0;
         tespera=0;
         sumEspera=0;
         int primerClienteVh =s_inicial[j].limiteInferior[k]+1;
         //printf("pcvH  %d   ",primerClienteVh);

         sumadist=sumadist+mDistCli[s_inicial[j].ruta[primerClienteVh]][cliente[datos.n+j].i];///modifque k
          //printf("pre____%d",s_inicial[j].ruta[primerClienteVh]);

         ///caso especial, en el primer cliente no se calcula tiempo de espera
         ///pero si el tiempo real de llegada
         if( mDistCli[s_inicial[j].ruta[
            primerClienteVh]][cliente[datos.n+j].i] > cliente[s_inicial[j].ruta[primerClienteVh]].e)

s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[primerClienteVh]]=mDistCli[s_inicial[j].ruta[primerClienteVh]][cliente[datos.
n+j].i] ;
         else
         s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[primerClienteVh]]=cliente[s_inicial[j].ruta[primerClienteVh]].e;


         for(w=primerClienteVh;w<s_inicial[j].limiteSuperior[k];w++)///aqui modifique k
            {
               sumadist=sumadist+mDistCli[s_inicial[j].ruta[w]][s_inicial[j].ruta[w+1]];
               suma=suma+cliente[s_inicial[j].ruta[w]].q;
               sumataten=sumataten+cliente[s_inicial[j].ruta[w]].d;
```

```
                if(s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]+cliente[s_inicial[j].ruta[w]].d+
mDistCli[s_inicial[j].ruta[w]][s_inicial[j].ruta[w+1]]> cliente[s_inicial[j].ruta[w+1]].e)

s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w+1]]=s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]+cliente[s_inicial[j].r
uta[w]].d+ mDistCli[s_inicial[j].ruta[w]][s_inicial[j].ruta[w+1]];
                else
                {
                s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w+1]]=cliente[s_inicial[j].ruta[w+1]].e;

                tespera=(s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]+cliente[s_inicial[j].ruta[w]].d+
mDistCli[s_inicial[j].ruta[w]][s_inicial[j].ruta[w+1]] - cliente[s_inicial[j].ruta[w+1]].e)* -1;
                ///printf("%d ESP %0.2f  ",s_inicial[j].ruta[w],tespera);
                sumEspera=sumEspera+tespera;
                }


                if(s_inicial[j].TiempAtenRealCli[w+1]    > (s_inicial[j].TiempAtenRealCli[w] +   cliente[s_inicial[j].ruta[w]].d+
mDistCli[s_inicial[j].ruta[w]][s_inicial[j].ruta[w+1]] )  )
                  {
                   tespera=      tespera+((s_inicial[j].TiempAtenRealCli[w+1]  )      -  (s_inicial[j].TiempAtenRealCli[w] +
cliente[s_inicial[j].ruta[w]].d+ mDistCli[s_inicial[j].ruta[w]][s_inicial[j].ruta[w+1]]));
                  // printf("%d ESP %0.2f",s_inicial[j].ruta[w],tespera);
                  }


            }
            sumadist=sumadist+mDistCli[s_inicial[j].ruta[w]][cliente[datos.n+j].i];


            //if(j==1 && k== 1)//printf("pos____%f",sumadist);

            suma=suma+cliente[s_inicial[j].ruta[w]].q;
            sumataten=sumataten+cliente[s_inicial[j].ruta[w]].d;

            solucion = solucion + sumadist;
            s_inicial[j].cargaVh[k]=suma;
            //printf("CARGA [%d %d]%f \n ",j,k,s_inicial[j].cargaVh[k]);
            s_inicial[j].TiempoFinVh[k]=s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]+cliente[s_inicial[j].ruta[w]].d+
mDistCli[s_inicial[j].ruta[w]][cliente[datos.n+j].i];

            ///ESTA ES UNA VALIDACION QUE ENVIA UN MENSAJE EN TIEMPO DE EJECUCION
            if(  deposito[j].D < suma){
           ///IMPORTANTE NO QUITAR*//*
           //printf("ALERTA VIOLACIÓN DE CARGA",j,k,s_inicial[j].cargaVh[k]);

            }


            printf("\n");
            ///  distancia  + tiempo de atención
            // printf(" %d %d %.2f\t Taten   ",j,k,sumataten);

            //   printf("***%.2f\t esp***\n ",tespera);
            distmastaten=sumadist+sumEspera +  sumataten;//sumadist+tespera +  sumataten
            s_inicial[j].distMasTiAtenVh[k] = distmastaten;



           // printf("\n");
            }     //  printf("\n");
      }

   sol.costoSolucionGlobal=solucion;///hacer una funcion que calcule la solucion global para no tener variable globales
}

///*********************************************

int escribeSol()
{
```

```
int j,k,w,dep,cli;
FILE *salida_txt;
//char nombre[20];
//sprintf(nombre, "salida%d.txt", rank);
salida_txt = fopen("salida.txt","w");
 if (!salida_txt) {
        return -1;
   }
   fprintf(salida_txt,"%.2f  \n",sol.costoSolucionGlobal);


   for(j=1;j<=datos.t;j++)
      {
         //li=1;
        for(k=1;k<=datos.m;k++)
           {
            fprintf(salida_txt,"%d ",j);
            //fscanf(archDatos," %d\t ",&dep);
            fprintf(salida_txt,"%d\t",k);
            fprintf(salida_txt,"\t");
            //fscanf(archDatos," %d\t ",&veh);
            fprintf(salida_txt,"%4.2f\t",s_inicial[j].distMasTiAtenVh[k]);
            //fscanf(archDatos," %f\t ",s_inicial[j].&num);
            fprintf(salida_txt,"\t");
            fprintf(salida_txt,"%4.2f\t",s_inicial[j].cargaVh[k]);
            dep=j+datos.n;

            fprintf(salida_txt,"%d ",dep);
            fprintf(salida_txt,"(0.00) ");

            //printf("\n %f\t%f ",cliente[dep].x,cliente[dep].y);


            for(w=s_inicial[j].limiteInferior[k]+1;w<=s_inicial[j].limiteSuperior[k];w++)
               {
               /* cli=s_inicial[j].ruta[w];
                  printf("\n %f\t%f",cliente[cli].x,cliente[cli].y);
               */

               fprintf(salida_txt,"%d ",s_inicial[j].ruta[w]);
               fprintf(salida_txt,"(%0.2f) ",s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]);

               }

            fprintf(salida_txt,"%d ",dep);
            fprintf(salida_txt,"(%0.2f) ",s_inicial[j].TiempoFinVh[k]);


            fprintf(salida_txt,"\n");
            fprintf(salida_txt,"\n");
            }
      }
   fclose(salida_txt);
                                    // printf("proceso realizado con exito");
                                    //

return (0);
}
///*************************************************************************************************************************************************
*****
void escribeGraf()
{
 int j,k,w,dep,cli;
 FILE *graf_txt;
    // printf("RUTA\n\n\n");
graf_txt = fopen("grafica.txt","w");
 if (!graf_txt) {
        return -1;
   }
```

```
 for(j=1;j<=datos.t;j++)
   {

   for(k=1;k<=datos.m;k++)
      {
      dep=j+datos.n;
      printf("\n%f\t%f ",cliente[dep].x,cliente[dep].y);
      fprintf(graf_txt,"%f\t%f\t[%d--%d] \n",cliente[dep].x,cliente[dep].y,j,k);
         for(w=s_inicial[j].limiteInferior[k];w<=s_inicial[j].limiteSuperior[k];w++)
            {
            cli=s_inicial[j].ruta[w];

            if (s_inicial[j].ruta[w]!=0)
               {
              // printf("\n ----------------------------------------------------------------\n ");
               printf("%f\t%f\t[%d--%d]\n",cliente[cli].x,cliente[cli].y,cliente[cli].e,cliente[cli].l);
               fprintf(graf_txt,"%f\t%f\t[%d--%d]\n",cliente[cli].x,cliente[cli].y,cliente[cli].e,cliente[cli].l);
               }



            }     //      printf("\n ----------------------------------------------------------------\n ");

            printf("\n%f\t%f ",cliente[dep].x,cliente[dep].y);
            fprintf(graf_txt,"%f\t%f\n ",cliente[dep].x,cliente[dep].y);


            printf("\n");
            fprintf(graf_txt,"\n");


      }


   }
    fclose(graf_txt);
}
///*************************************************************************************************************************
solucion swapVhIntra(solucion p[DEP_MAX],int indiceDep,int indiceVh)
{
     int i,j,num1=0,num2,inf=0, sup=0,infOtro=0, supOtro=0,aux=0,indiceOtroVh;
   srand(time(NULL));
   solucion pTemp[1];
   pTemp[0]=p[indiceDep];///vh actual
   inf=pTemp[0].limiteInferior[indiceVh]+1;
   sup=pTemp[0].limiteSuperior[indiceVh];

   do{

      num1=inf+rand()%((sup-inf)+1);


      num2=inf+rand()%((sup-inf)+1);

               printf("\n >SWAP INTRA            DEP %d VH %d inf  %d  sup   %d  num1   %d  num2   %d",indiceDep,
indiceVh,inf,sup, num1,num2);

               printf("\n n1 %d n2 %d ",pTemp[0].ruta[num1],pTemp[0].ruta[num2]);

   }while( num1==  num2);


   /*intercambio*/

     aux=pTemp[0].ruta[num1]   ;
     pTemp[0].ruta[num1]=pTemp[0].ruta[num2];
     pTemp[0].ruta[num2]=aux;
```

```
            printf("\n n1 %d n2 %d ",pTemp[0].ruta[num1],pTemp[0].ruta[num2]);

    return(pTemp[0]);
}

///************************************************
solucion parAdyacente(solucion p[DEP_MAX],int indiceDep,int indiceVh)
{
    int i,j,num1=0,num2,inf=0, sup=0,aux=0;
   srand(time(NULL));
   solucion pTemp[1];
   pTemp[0]=p[indiceDep];
   inf=pTemp[0].limiteInferior[indiceVh]+1;
   sup=pTemp[0].limiteSuperior[indiceVh]-1;

   num1=inf+rand()%((sup-inf)+1);
            printf("\n ADYACENTE DEP %d VH %d inf  %d  sup   %d  num1   %d",indiceDep, indiceVh,inf,sup, num1);
   num2=num1+1;
            printf("\n n1 %d n2 %d ",pTemp[0].ruta[num1],pTemp[0].ruta[num2]);
   /*intercambio*/

     aux=pTemp[0].ruta[num1]   ;
     pTemp[0].ruta[num1]=pTemp[0].ruta[num2];
     pTemp[0].ruta[num2]=aux;

            printf("\n n1 %d n2 %d ",pTemp[0].ruta[num1],pTemp[0].ruta[num2]);

///*******************
///**********************AGREGAR AQUI LAS 3 VALIDACIONES
///*******************
///costo
///TW cliente
///Duracion de la ruta
///si el cambio es valido devolver pTemp  sino devolver  p

    return(pTemp[0]);
}
///*************************************************************************
///*****************************************************************************************************************
void swapVhInter(solucion p[DEP_MAX],int indiceDep,int indiceVh1,int indiceVh2)///corregir*********
{
    int i,j,num1=0,num2,inf=0, sup=0,infOtro=0, supOtro=0,aux=0,indiceOtroVh;
   srand(time(NULL));
   //solucion pTemp[1];
   //pTemp[0]=p[indiceDep];///vh actual
   s_temp[indiceDep]=p[indiceDep];


   inf=s_temp[indiceDep].limiteInferior[indiceVh1]+1;
   sup=s_temp[indiceDep].limiteSuperior[indiceVh1];

   infOtro=s_temp[indiceDep].limiteInferior[indiceVh2]+1;
   supOtro=s_temp[indiceDep].limiteSuperior[indiceVh2];

   //do{

      num1=inf+rand()%((sup-inf)+1);


      num2=infOtro+rand()%((supOtro-infOtro)+1);


            printf("\n >SWAP INTER          DEP %d VH1  %d  inf %d  sup %d  num1  %d \n                          VH2
%d infotro  %d  supotro   %d  num2   %d",indiceDep, indiceVh1,inf,sup, num1, indiceVh2,infOtro,supOtro, num2);


            printf("\n n1 %d n2 %d ",s_temp[indiceDep].ruta[num1],s_temp[indiceDep].ruta[num2]);

   //}while( num1==  num2);
```

```
    //intercambio*/

      aux=s_temp[indiceDep].ruta[num1]   ;
      s_temp[indiceDep].ruta[num1]=s_temp[indiceDep].ruta[num2];
      s_temp[indiceDep].ruta[num2]=aux;

                printf("\n n1 %d n2 %d ",s_temp[indiceDep].ruta[num1],s_temp[indiceDep].ruta[num2]);

      //return(pTemp[0]);
}

///************************************************
void swapVhDep(solucion p[DEP_MAX],int indiceDep1,int indiceDep2,int indiceVh1,int indiceVh2)
{
 int i,j,num1=0,num2,inf=0, sup=0,infOtro=0, supOtro=0,aux=0,indiceOtroVh;
    srand(time(NULL));
    //solucion pTemp[1];
    //pTemp[0]=p[indiceDep];///vh actual
    s_temp[indiceDep1]=p[indiceDep1];
    s_temp[indiceDep2]=p[indiceDep2];

    inf=s_temp[indiceDep1].limiteInferior[indiceVh1]+1;
    sup=s_temp[indiceDep1].limiteSuperior[indiceVh1];

    infOtro=s_temp[indiceDep2].limiteInferior[indiceVh2]+1;
    supOtro=s_temp[indiceDep2].limiteSuperior[indiceVh2];


        num1=inf+rand()%((sup-inf)+1);


        num2=infOtro+rand()%((supOtro-infOtro)+1);


                printf("\n >SWAP DEP           DEP1 %d VH1  %d  inf %d  sup %d  num1  %d \n                    DEP2 %d
VH2     %d  infotro     %d     supotro        %d     num2        %d",indiceDep1,  indiceVh1,inf,sup,  num1,indiceDep2,
indiceVh2,infOtro,supOtro, num2);
                printf("\n n1 %d n2 %d ",s_temp[indiceDep1].ruta[num1],s_temp[indiceDep2].ruta[num2]);

    //}while( num1==  num2);

    /*intercambio*/

      aux=s_temp[indiceDep1].ruta[num1]   ;
      s_temp[indiceDep1].ruta[num1]=s_temp[indiceDep2].ruta[num2];
      s_temp[indiceDep2].ruta[num2]=aux;

                printf("\n n1 %d n2 %d ",s_temp[indiceDep1].ruta[num1],s_temp[indiceDep2].ruta[num2]);

}

///************************************************
void insercionVhInter(solucion p[DEP_MAX],int indiceDep,int indiceVh1,int indiceVh2)
{
    int rSale, rHueco,huecoInf=0,huecoSup,saleInf,saleSup,hueco,sale,aux,c=0,i;
    srand(time(NULL));
    s_temp[indiceDep]=p[indiceDep];



/// elije las rutas para la insercion
/*
do{
      rSale=rand()%p.vh;
      rHueco=rand()%p.vh;
```

```
}while(rSale==rHueco);
                printf("\n rSale %d rHueco %d \n",rSale , rHueco);
*/
///las rutas se mandan como parametro    int indiceVh1,int indiceVh2

   rSale=indiceVh1;
   rHueco=indiceVh2;

///selecciona a los clientes

     huecoInf=s_temp[indiceDep].limiteInferior[rHueco]+1;
      huecoSup=s_temp[indiceDep].limiteSuperior[rHueco];

      saleInf=s_temp[indiceDep].limiteInferior[rSale]+1;
      saleSup=s_temp[indiceDep].limiteSuperior[rSale];

     hueco=huecoInf+rand()%((huecoSup-huecoInf)+1);///clientes seleccionados
     sale=saleInf+rand()%((saleSup-saleInf)+1);


                printf("\n >INSERT VH      DEP %d   VH1   %d   inf %d   sup %d   SALE   %d      \n ",indiceDep,
rSale,saleInf,saleSup, sale);
                printf("\n >INSERT VH      VH2  %d  inf %d  sup %d  hueco  %d    \n  ", rHueco,huecoInf,huecoSup,
hueco);

                printf("\n sale %d hueco %d ",s_temp[indiceDep].ruta[sale],s_temp[indiceDep].ruta[hueco]);

/// corrimiento a la der o izq



                printf(" Insercion Mueve cliente de posicion  \n");
       if (hueco < sale)
          {
             aux= s_temp[indiceDep].ruta[sale];//le damos el valor de num2 a aux para que se lo guarde

             for (c=sale; c > hueco; c--)
                s_temp[indiceDep].ruta[c]= s_temp[indiceDep].ruta[c-1];

                s_temp[indiceDep].ruta[hueco]=aux;

          }//del if

       else
          {
             aux= s_temp[indiceDep].ruta[sale];//le damos el valor de num2 a aux para que se lo guarde

             for (c=sale; c < hueco; c++)
                s_temp[indiceDep].ruta[c]= s_temp[indiceDep].ruta[c+1];

                s_temp[indiceDep].ruta[hueco]=aux;

          }//del else

///ajuste de los indices de los limites inf y sip


       if (rHueco < rSale)
          {
             s_temp[indiceDep].limiteSuperior[rHueco]=s_temp[indiceDep].limiteSuperior[rHueco]+1;///huecoSup + 1;

             for(i=rHueco + 1  ;i< rSale   ;i++)
                {
                   s_temp[indiceDep].limiteInferior[i]= s_temp[indiceDep].limiteInferior[i] + 1;
                   s_temp[indiceDep].limiteSuperior[i]= s_temp[indiceDep].limiteSuperior[i] + 1;

                }


             s_temp[indiceDep].limiteInferior[rSale]=s_temp[indiceDep].limiteInferior[rSale]+1;///saleInf + 1;
```

```
          }//del if

         else
          {
             s_temp[indiceDep].limiteSuperior[rSale]=s_temp[indiceDep].limiteSuperior[rSale] -1;///saleSup - 1;

             for(i=rSale + 1  ;i< rHueco   ;i++)
                {
                   s_temp[indiceDep].limiteInferior[i]=s_temp[indiceDep].limiteInferior[i]- 1;
                   s_temp[indiceDep].limiteSuperior[i]= s_temp[indiceDep].limiteSuperior[i]- 1;

                }

             s_temp[indiceDep].limiteInferior[rHueco]=s_temp[indiceDep].limiteInferior[rHueco]-1;///huecoInf       -       1;
///******************último cambio

          }//del else

}

///***********************************************
void imprimeRutas(solucion p[DEP_MAX],instancia datos)
{
   int i,j;
          printf("\nRUTAS\n");
   for (i=1; i<=datos.t ; i++)
      {
      for (j=1; j<=p[i].numClientes+datos.m; j++)
         {
         printf("%d\t",p[i].ruta[j]);
         }
         printf("\n");printf("\n");
      }


         printf("LIMITES\n");

   for (i=1; i<=datos.t ; i++)
      {
      for (j=1; j<=datos.m; j++)
         {
         printf("[Inf]%d\t[Sup]%d\t",p[i].limiteInferior[j],p[i].limiteSuperior[j]);
         }
         printf("\n");printf("\n");
      }
}

///***********************************************

int          evaluaTwVh(solucion          p[DEP_MAX],conf_cli          cliente[CLI_MAX          +          DEP_MAX],float
mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],int indiceDep,int indiceVh)
{
int j=1,k=1;
int w=1,banderaTW=1;


float sumadist,sumataten,sumadistTotal,distmastaten,tespera,sumEspera,solucionDis;
int primerClienteVh =p[j].limiteInferior[k]+1;


int clientes;

j=indiceDep;
k=indiceVh;


printf("\n       VALIDACION   TW  DE UNA RUTA  \n");
printf("\n");
```

```
                    printf(" DEP %d  RUTA %d \n",j,k);

     sumataten=0;
     distmastaten=0;
     sumadist=0;
     tespera=0;



    // sumadist=sumadist+mDistCli[p[j].ruta[p[j].limiteInferior[k]]][cliente[datos.n+j].i];///distancia del dep al primer cliente

    if(p[j].limiteInferior[k]==p[j].limiteSuperior[k])///revisa si la ruta está vacia
    {
        printf("0.00\n0.00    RUTA VACIA\n\n");/// si la ruta está vacia no se vioa la restriccion  banderaTW=1;
    }
    else
    {
        sumadist=sumadist+mDistCli[p[j].ruta[primerClienteVh]][cliente[datos.n+j].i];

        ///caso especial, en el primer cliente no se calcula tiempo de espera
        ///pero si el tiempo real de llegada
        if( mDistCli[p[j].ruta[primerClienteVh]][cliente[datos.n+j].i] > cliente[p[j].ruta[primerClienteVh]].e)
           p[j].TiempAtenRealCli[p[j].ruta[primerClienteVh]]=mDistCli[p[j].ruta[primerClienteVh]][cliente[datos.n+j].i] ;
        else
           p[j].TiempAtenRealCli[p[j].ruta[primerClienteVh]]=cliente[p[j].ruta[primerClienteVh]].e;



        for(w=primerClienteVh;w<(p[j].limiteSuperior[k]);w++)
              {
              sumadist=sumadist+mDistCli[p[j].ruta[w]][p[j].ruta[w+1]];
              sumataten=sumataten+cliente[p[j].ruta[w]].d;
              //clientes=p[j].ruta[w];///?


              if(p[j].TiempAtenRealCli[p[j].ruta[w]]+cliente[p[j].ruta[w]].d+ mDistCli[p[j].ruta[w]][p[j].ruta[w+1]]>
cliente[p[j].ruta[w+1]].e)
                 {
                 p[j].TiempAtenRealCli[p[j].ruta[w+1]]=p[j].TiempAtenRealCli[p[j].ruta[w]]+cliente[p[j].ruta[w]].d+
mDistCli[p[j].ruta[w]][p[j].ruta[w+1]];
                 }
              else
                 {
                 p[j].TiempAtenRealCli[p[j].ruta[w+1]]=cliente[p[j].ruta[w+1]].e;

                 tespera=(p[j].TiempAtenRealCli[p[j].ruta[w]]+cliente[p[j].ruta[w]].d+  mDistCli[p[j].ruta[w]][p[j].ruta[w+1]]  -
cliente[p[j].ruta[w+1]].e)* -1;
                 ///printf("%d ESP %0.2f  ",P[j].ruta[w],tespera);
                 sumEspera=sumEspera+tespera;
                 }

              if(p[j].TiempAtenRealCli[p[j].ruta[w]+1]     > (p[j].TiempAtenRealCli[p[j].ruta[w]] +    cliente[p[j].ruta[w]].d+
mDistCli[p[j].ruta[w]][p[j].ruta[w+1]] )  )
                 {
                 tespera=    tespera+((p[j].TiempAtenRealCli[p[j].ruta[w]+1]  )    - (p[j].TiempAtenRealCli[p[j].ruta[w]] +
cliente[p[j].ruta[w]].d+ mDistCli[p[j].ruta[w]][p[j].ruta[w+1]]));
                 }

if(cliente[p[j].ruta[w]].e<=p[j].TiempAtenRealCli[p[j].ruta[w]]&&p[j].TiempAtenRealCli[p[j].ruta[w]]+cliente[p[j].ruta[w]].d<=
cliente[p[j].ruta[w]].l)
                    {

                     printf("%d\t[%d] \t %4.1f--%4.1f  \t[%d]  \t OK \n",p[j].ruta[w],cliente[p[j].ruta[w]].e,
                     p[j].TiempAtenRealCli[p[j].ruta[w]],p[j].TiempAtenRealCli[p[j].ruta[w]]+cliente[p[j].ruta[w]].d,
                     cliente[p[j].ruta[w]].l);
                    }
```

```
                    else
                       {
                          banderaTW=0;
                         printf("%d\t[%d] \t %4.1f--%4.1f  \t[%d]  \t X \n",p[j].ruta[w],cliente[p[j].ruta[w]].e,

p[j].TiempAtenRealCli[p[j].ruta[w]],p[j].TiempAtenRealCli[p[j].ruta[w]]+cliente[p[j].ruta[w]].d,cliente[p[j].ruta[w]].l);
                       }

                 }


if(cliente[p[j].ruta[w]].e<=p[j].TiempAtenRealCli[p[j].ruta[w]]&&p[j].TiempAtenRealCli[p[j].ruta[w]]+cliente[p[j].ruta[w]].d<=
                          cliente[p[j].ruta[w]].l)
                          {

                             printf("%d\t[%d]          \t          %4.1f--%4.1f                    \t[%d]                    \t          OK
\n",p[j].ruta[w],cliente[p[j].ruta[w]].e,p[j].TiempAtenRealCli[p[j].ruta[w]],
                               p[j].TiempAtenRealCli[p[j].ruta[w]]+cliente[p[j].ruta[w]].d,cliente[p[j].ruta[w]].l);
                           }
                       else
                       {
                          banderaTW=0;
                          printf("%d\t[%d]          \t          %4.1f--%4.1f                    \t[%d]                    \t          X
\n",p[j].ruta[w],cliente[p[j].ruta[w]].e,p[j].TiempAtenRealCli[p[j].ruta[w]],
                               p[j].TiempAtenRealCli[p[j].ruta[w]]+cliente[p[j].ruta[w]].d,cliente[p[j].ruta[w]].l);

                       }


       printf("\nbanderaTW %d \n ",banderaTW)  ;


       }


return(banderaTW);

}

///************************************************
int evaluaCargaVh(solucion p[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],int indiceDep,int indiceVh)
{
 float suma;
    int j,k,w;
    int banderaQ=1;
                                  /// falta la bandera*********************************************************
    printf("\n        CARGA \n");
    printf("\n");


       ///   cargas
          j=indiceDep;
          k=indiceVh;
          suma=0;///para carga

          int primerClienteVh =p[j].limiteInferior[k]+1;
          //printf("pcvH  %d   ",primerClienteVh);



          for(w=primerClienteVh;w<=p[j].limiteSuperior[k];w++)
             {
                suma=suma+cliente[p[j].ruta[w]].q;

             }



          p[j].cargaVh[k]=suma;
```

```
            printf("CARGA [%d %d]%f \n ",j,k,p[j].cargaVh[k]);

            ///ESTA ES UNA VALIDACION QUE ENVIA UN MENSAJE EN TIEMPO DE EJECUCION
            if(  deposito[j].D < suma){
           ///IMPORTANTE NO QUITAR*//*
           printf("ALERTA VIOLACIÓN DE CARGA",j,k,p[j].cargaVh[k]);
            banderaQ=0;
            }


            printf("\n");

return(banderaQ);
}

///***********************************************

void insercionVhDep(solucion p[DEP_MAX],int indiceDep1,int indiceDep2,int indiceVh1,int indiceVh2)
{
int rSale, rHueco,huecoInf=0,huecoSup,saleInf,saleSup,hueco,sale,aux,c=0,i;
int ultimaVhSupSale   ;

   srand(time(NULL));

   ///copia de las estructuras de dos depositos
   s_temp[indiceDep1]=p[indiceDep1];
   s_temp[indiceDep2]=p[indiceDep2];



/// elije las rutas para la insercion
/*
do{
     rSale=rand()%p.vh;
     rHueco=rand()%p.vh;
}while(rSale==rHueco);
               printf("\n rSale %d rHueco %d \n",rSale , rHueco);
*/
///las rutas se mandan como parametro    int indiceVh1,int indiceVh2

   rSale=indiceVh1;///del Dep1
   rHueco=indiceVh2;///del Dep2

///selecciona a los clientes



      saleInf=s_temp[indiceDep1].limiteInferior[rSale]+1;
      saleSup=s_temp[indiceDep1].limiteSuperior[rSale];

      huecoInf=s_temp[indiceDep2].limiteInferior[rHueco]+1;
      huecoSup=s_temp[indiceDep2].limiteSuperior[rHueco];

    hueco=huecoInf+rand()%((huecoSup-huecoInf)+1);///clientes seleccionados
    sale=saleInf+rand()%((saleSup-saleInf)+1);

                  printf("\n >INSERT VH      DEP %d   VH1   %d   inf %d   sup %d   SALE   %d      \n   ",indiceDep1,
rSale,saleInf,saleSup, sale);
                  printf("\n >INSERT VH      DEP %d   VH2   %d   inf %d   sup %d   hueco %d      \n   ",indiceDep2,
rHueco,huecoInf,huecoSup, hueco);
                  printf("\n sale %d hueco %d ",s_temp[indiceDep1].ruta[sale],s_temp[indiceDep2].ruta[hueco]);



/// corrimiento del Dep1 sale

   aux= s_temp[indiceDep1].ruta[sale];//le damos el valor de num2 a aux para que se lo guarde

///calcula el limite sup de la ultima ruta
   ultimaVhSupSale=s_temp[indiceDep1].limiteSuperior[datos.m];
```

```
   for (c=sale;  c < ultimaVhSupSale; c++)
      s_temp[indiceDep1].ruta[c]= s_temp[indiceDep1].ruta[c+1];



///ajuste de los indices de los limites inf y sup del Dep1 sale
   s_temp[indiceDep1].limiteSuperior[rSale]=s_temp[indiceDep1].limiteSuperior[rSale] -1;///saleSup - 1;
   for(i=rSale + 1  ;i< datos.m  ;i++)
      {
      s_temp[indiceDep1].limiteInferior[i]=s_temp[indiceDep1].limiteInferior[i]- 1;
      s_temp[indiceDep1].limiteSuperior[i]= s_temp[indiceDep1].limiteSuperior[i]- 1;
      }

      s_temp[indiceDep1].limiteSuperior[i]=s_temp[indiceDep1].limiteSuperior[i]-1;///huecoInf                -                1;
///******************último cambio
      s_temp[indiceDep1].numClientes=s_temp[indiceDep1].numClientes-1;



///abre hueco para insertar en Dep2
      s_temp[indiceDep2].numClientes=s_temp[indiceDep2].numClientes+1;
      for (c=s_temp[indiceDep2].limiteSuperior[datos.m]+1;c >= hueco; c--)
          s_temp[indiceDep2].ruta[c]= s_temp[indiceDep2].ruta[c-1];


      s_temp[indiceDep2].ruta[hueco]=aux;


                                  ///AQUI VOY BIEN

///ajuste de los limites Dep2

  s_temp[indiceDep2].limiteSuperior[rHueco]=s_temp[indiceDep2].limiteSuperior[rHueco] +1;///saleSup - 1;
   for(i=rHueco +1  ;i<= datos.m  ;i++)
      {
      s_temp[indiceDep2].limiteInferior[i]=s_temp[indiceDep2].limiteInferior[i]+ 1;
      s_temp[indiceDep2].limiteSuperior[i]=s_temp[indiceDep2].limiteSuperior[i]+ 1;
      }
}

///*************************************************************************

void perturba(solucion p[DEP_MAX],int heuriSel,int indiceDep1,int indiceDep2,int indiceVh1,int indiceVh2)
{


                        switch (heuriSel)
                        {
                                case    1:s_temp[indiceDep1]=swapVhIntra(s_inicial,indiceDep1,indiceVh1);///printf("
Heuristica 1 ");
                                        break;
                                case  2:s_temp[indiceDep1]=parAdyacente(s_inicial,indiceDep1,indiceVh1);///printf("
Heuristica 2 ");
              break;
                                case 3:swapVhInter(s_inicial,indiceDep1,indiceVh1,indiceVh2);///printf(" Heuristica 3
");
                                        break;
                                case            4:insercionVhInter(s_inicial,indiceDep1,indiceVh1,indiceVh2);///printf("
Heuristica 4 ");
                                        break;
          case 5:swapVhDep(s_inicial,indiceDep1,indiceDep2,indiceVh1,indiceVh2);///printf(" Heuristica 5 ");
                                        break;
          case 6:insercionVhDep(s_inicial,indiceDep1,indiceDep2,indiceVh1,indiceVh2);///printf(" Heuristica 6 ");
                                        break;

                        }
```

```
}

///**************************************************************************

///*****************************************************************************************************************************************
solucion swapVh(solucion p[DEP_MAX],int indiceDep)
{
     int num1=0,num2,aux=0,indiceVh;

   solucion pTemp[1];
   pTemp[0]=p[indiceDep];


   while((num1 == num2) || (pTemp[0].ruta[num1] == 0 )|| (pTemp[0].ruta[num2] == 0))
     {

                ///delimitar  el rand al num de clientes de la ruta
                num1=rand()%p[indiceDep].numClientes;
                num2=rand()%p[indiceDep].numClientes;
   }


printf("\n SWAP VH");

printf("\n n1 %d n2 %d ",num1,num2);
          printf("\n n1 %d n2 %d ",p[0].ruta[num1],p[0].ruta[num2]);
   /*intercambio*/
     aux=pTemp[0].ruta[num1]   ;
     pTemp[0].ruta[num1]=pTemp[0].ruta[num2];
     pTemp[0].ruta[num2]=aux;
     //fin de validación If
          printf("\n n1 %d n2 %d ",p[0].ruta[num1],p[0].ruta[num2]);



     return(pTemp[0]);
}


///*****************************************************************

solucion swapSegmenVh(solucion p[DEP_MAX],int indiceDep)
{
     int num1=0,num2,aux=0,indiceVh;

   solucion pTemp[1];
   pTemp[0]=p[indiceDep];

/// buscar la forma  en la que los cabio se hagan de acuerdo a semejanzas entre las ctw de los cli
/// partiendo los sub-vectores
   while((num1 == num2) || (pTemp[0].ruta[num1] == 0 )|| (pTemp[0].ruta[num2] == 0))
     {
                num1=rand()%p[indiceDep].numClientes;
                num2=rand()%p[indiceDep].numClientes;
   }




          //printf("\n n1 %d n2 %d ",p[0].ruta[num1],p[0].ruta[num2]);
   /*intercambio*/
     aux=pTemp[0].ruta[num1]   ;
     pTemp[0].ruta[num1]=pTemp[0].ruta[num2];
     pTemp[0].ruta[num2]=aux;
     //fin de validación If
          //printf("\n n1 %d n2 %d ",p[0].ruta[num1],p[0].ruta[num2]);

///*********************
```

```
///************************AGREGAR AQUI LAS 3 VALIDACIONES
///*******************

///TW cliente
///Duracion de la ruta
///costo


///si el cambio es valido devolver pTemp  sino devolver  p

     return(pTemp[0]);
}

void  costoVh(solucion p[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],conf_dep deposito[DEP_MAX],
                instancia datos,float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],int indiceDep,int indiceVh)
{
   float suma,sumadist,sumataten,sumadistTotal,distmastaten,tespera,sumEspera;
   int i,j,k,w;
   int clientes;
   float solucion =0;

  // printf("\n         COSTO \n");
  // printf("\n");
   for(j=1;j<=datos.t;j++)
      {
      for(k=1;k<=datos.m;k++)/// k inicia en 2 porque saltamos el marcador de inicio de ruta que  es el valor cero
         {        ///   cargas
         suma=0;///para carga
         sumataten=0;
         distmastaten=0;
         sumadist=0;
         tespera=0;
         sumEspera=0;



         int primerClienteVh =p[0].limiteInferior[indiceVh]+1;
         //printf("pcvH  %d   ",primerClienteVh);

         sumadist=sumadist+mDistCli[p[0].ruta[primerClienteVh]][cliente[datos.n+j].i];///modifque k
          //printf("pre____%d",p[0].ruta[primerClienteVh]);

         ///caso especial, en el primer cliente no se calcula tiempo de espera
         ///pero si el tiempo real de llegada
         if( mDistCli[p[0].ruta[primerClienteVh]][cliente[datos.n+j].i] > cliente[p[0].ruta[primerClienteVh]].e)
         p[0].TiempAtenRealCli[p[0].ruta[primerClienteVh]]=mDistCli[p[0].ruta[primerClienteVh]][cliente[datos.n+j].i] ;
         else
         p[0].TiempAtenRealCli[p[0].ruta[primerClienteVh]]=cliente[p[0].ruta[primerClienteVh]].e;


         for(w=primerClienteVh;w<p[0].limiteSuperior[k];w++)///aqui modifique k
            {
            sumadist=sumadist+mDistCli[p[0].ruta[w]][p[0].ruta[w+1]];
            suma=suma+cliente[p[0].ruta[w]].q;
            sumataten=sumataten+cliente[p[0].ruta[w]].d;



            if(p[0].TiempAtenRealCli[p[0].ruta[w]]+cliente[p[0].ruta[w]].d+          mDistCli[p[0].ruta[w]][p[0].ruta[w+1]]>
cliente[p[0].ruta[w+1]].e)
            p[0].TiempAtenRealCli[p[0].ruta[w+1]]=p[0].TiempAtenRealCli[p[0].ruta[w]]+cliente[p[0].ruta[w]].d+
mDistCli[p[0].ruta[w]][p[0].ruta[w+1]];
            else
            {
            p[0].TiempAtenRealCli[p[0].ruta[w+1]]=cliente[p[0].ruta[w+1]].e;

            tespera=(p[0].TiempAtenRealCli[p[0].ruta[w]]+cliente[p[0].ruta[w]].d+   mDistCli[p[0].ruta[w]][p[0].ruta[w+1]]   -
cliente[p[0].ruta[w+1]].e)* -1;
            ///printf("%d ESP %0.2f  ",p[0].ruta[w],tespera);
            sumEspera=sumEspera+tespera;
```

```
            }


            if(p[0].TiempAtenRealCli[w+1]              >     (p[0].TiempAtenRealCli[w]     +          cliente[p[0].ruta[w]].d+
mDistCli[p[0].ruta[w]][p[0].ruta[w+1]] )  )
               {
               tespera=   tespera+((p[0].TiempAtenRealCli[w+1] )   - (p[0].TiempAtenRealCli[w] +    cliente[p[0].ruta[w]].d+
mDistCli[p[0].ruta[w]][p[0].ruta[w+1]]));
              //  printf("%d ESP %0.2f",p[0].ruta[w],tespera);
               }


            }
            sumadist=sumadist+mDistCli[p[0].ruta[w]][cliente[datos.n+j].i];
           // if(j==2 && k== 2)printf("pos____%f",sumadist);

            suma=suma+cliente[p[0].ruta[w]].q;
            sumataten=sumataten+cliente[p[0].ruta[w]].d;

            solucion = solucion + sumadist;
            p[0].cargaVh[k]=suma;
            //printf("CARGA [%d %d]%f \n ",j,k,p[0].cargaVh[k]);
            p[0].TiempoFinVh[k]=p[0].TiempAtenRealCli[p[0].ruta[w]]+cliente[p[0].ruta[w]].d+
mDistCli[p[0].ruta[w]][cliente[datos.n+j].i];

            ///ESTA ES UNA VALIDACION QUE ENVIA UN MENSAJE EN TIEMPO DE EJECUCION
            if(  deposito[j].D < suma)
           ///IMPORTANTE NO QUITAR*//*printf("ALERTA VIOLACIÓN DE CARGA",j,k,p[0].cargaVh[k]);



           // printf("\n");
            ///  distancia  + tiempo de atención
            // printf(" %d %d %.2f\t Taten   ",j,k,sumataten);

            //   printf("***%.2f\t esp***\n ",tespera);
            distmastaten=sumadist+sumEspera +  sumataten;//sumadist+tespera +  sumataten
            p[0].distMasTiAtenVh[k] = distmastaten;



           // printf("\n");
            }    //   printf("\n");
      }


}

///*************************************   FUNCIONES DE VALIDACION


int validaTWCli(solucion s_inicial[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],conf_dep deposito[DEP_MAX],
                                instancia datos,float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX])
{
float suma,sumadist,sumataten,sumadistTotal,distmastaten,tespera,sumEspera;
   int i,j,k,w;
   int clientes;
   float solucion =0;
   int fTWVh[datos.m+1];

int bandera;

 int banderaTWCli=0,cli;
int itera=1000;



printf("\n ----------------------------------------------------------------\n ");
 printf("\n  Validación antes\n");
```

```
 for(j=1;j<=datos.t;j++)
   {
///aqui ciclo para reparacion while de banderas
do{
bandera=0;
for(k=1;k<=datos.m;k++)fTWVh[k]=0;
   for(k=1;k<=datos.m;k++)
      {
         printf("DEP %d VH %d\n",j,k);
         suma=0;///para carga
         sumataten=0;
         distmastaten=0;
         sumadist=0;
         tespera=0;
         sumEspera=0;
         int primerClienteVh =s_inicial[j].limiteInferior[k]+1;
         //printf("pcvH  %d   ",primerClienteVh);

         sumadist=sumadist+mDistCli[s_inicial[j].ruta[primerClienteVh]][cliente[datos.n+j].i];///modifque k
                           //printf("pre____%d",s_inicial[j].ruta[primerClienteVh]);

                           ///caso especial, en el primer cliente no se calcula tiempo de espera
                           ///pero si el tiempo real de llegada
                           if( mDistCli[s_inicial[j].ruta[
                              primerClienteVh]][cliente[datos.n+j].i] > cliente[s_inicial[j].ruta[primerClienteVh]].e)

s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[primerClienteVh]]=mDistCli[s_inicial[j].ruta[primerClienteVh]][cliente[datos.
n+j].i] ;
                           else

s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[primerClienteVh]]=cliente[s_inicial[j].ruta[primerClienteVh]].e;


                           for(w=primerClienteVh;w<s_inicial[j].limiteSuperior[k];w++)///aqui modifique k
                              {
                                 sumadist=sumadist+mDistCli[s_inicial[j].ruta[w]][s_inicial[j].ruta[w+1]];
                                 suma=suma+cliente[s_inicial[j].ruta[w]].q;
                                 sumataten=sumataten+cliente[s_inicial[j].ruta[w]].d;



                                 if(s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]+cliente[s_inicial[j].ruta[w]].d+
mDistCli[s_inicial[j].ruta[w]][s_inicial[j].ruta[w+1]]> cliente[s_inicial[j].ruta[w+1]].e)

s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w+1]]=s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]+cliente[s_inicial[j].r
uta[w]].d+ mDistCli[s_inicial[j].ruta[w]][s_inicial[j].ruta[w+1]];
                                 else
                                 {
                                 s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w+1]]=cliente[s_inicial[j].ruta[w+1]].e;

                                 tespera=(s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]+cliente[s_inicial[j].ruta[w]].d+
mDistCli[s_inicial[j].ruta[w]][s_inicial[j].ruta[w+1]] - cliente[s_inicial[j].ruta[w+1]].e)* -1;
                                 ///printf("%d ESP %0.2f  ",s_inicial[j].ruta[w],tespera);
                                 sumEspera=sumEspera+tespera;
                                 }


                                 if(s_inicial[j].TiempAtenRealCli[w+1]      >     (s_inicial[j].TiempAtenRealCli[w]     +
cliente[s_inicial[j].ruta[w]].d+ mDistCli[s_inicial[j].ruta[w]][s_inicial[j].ruta[w+1]] )  )
                                    {
                                    tespera=          tespera+((s_inicial[j].TiempAtenRealCli[w+1]         )          -
(s_inicial[j].TiempAtenRealCli[w] +   cliente[s_inicial[j].ruta[w]].d+ mDistCli[s_inicial[j].ruta[w]][s_inicial[j].ruta[w+1]]));
                                    // printf("%d ESP %0.2f",s_inicial[j].ruta[w],tespera);
                                    }
/// validacion de la TWCli fin

if(s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]+cliente[s_inicial[j].ruta[w]].d > cliente[s_inicial[j].ruta[w]].l)
{
 fTWVh[k]=1; //identifica a la rutra
```

```
banderaTWCli=1;//idenfifica al cliente
bandera=1;// identifica la violación en el dep
}
else
{
banderaTWCli=0;
}


printf("%d\t[%d--%d]\tTR %f\tDist %f\tAten %d\tTWCli %d\n",s_inicial[j].ruta[w],cliente[s_inicial[j].ruta[w]].e,
      cliente[s_inicial[j].ruta[w]].l,s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]],
       mDistCli[s_inicial[j].ruta[w-1]][s_inicial[j].ruta[w]],cliente[s_inicial[j].ruta[w]].d,banderaTWCli);
//getch();



                                }///fin for


 /// validacion de la TWCli fin del ultimo cliente

if(s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]+cliente[s_inicial[j].ruta[w]].d > cliente[s_inicial[j].ruta[w]].l)
{
 fTWVh[k]=1; //identifica a la rutra
banderaTWCli=1;//idenfifica al cliente
bandera=1;// identifica la violación en el dep
}
else
{
banderaTWCli=0;
}


printf("%d\t[%d--%d]\tTR %f\tDist %f\tAten %d\tTWCli %d\n",s_inicial[j].ruta[w],cliente[s_inicial[j].ruta[w]].e,
      cliente[s_inicial[j].ruta[w]].l,s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]],
       mDistCli[s_inicial[j].ruta[w-1]][s_inicial[j].ruta[w]],cliente[s_inicial[j].ruta[w]].d,banderaTWCli);
//getch();


                                sumadist=sumadist+mDistCli[s_inicial[j].ruta[w]][cliente[datos.n+j].i];


                               // if(j==1 && k== 1)//printf("pos____%f",sumadist);

                                suma=suma+cliente[s_inicial[j].ruta[w]].q;
                                sumataten=sumataten+cliente[s_inicial[j].ruta[w]].d;

                                solucion = solucion + sumadist;
                                s_inicial[j].cargaVh[k]=suma;
                                //printf("CARGA [%d %d]%f \n ",j,k,s_inicial[j].cargaVh[k]);

s_inicial[j].TiempoFinVh[k]=s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]+cliente[s_inicial[j].ruta[w]].d+
mDistCli[s_inicial[j].ruta[w]][cliente[datos.n+j].i];

                                ///ESTA ES UNA VALIDACION QUE ENVIA UN MENSAJE EN TIEMPO DE EJECUCION
                                if(  deposito[j].D < suma){
                               ///IMPORTANTE NO QUITAR*//*
                               //printf("ALERTA VIOLACIÓN DE CARGA",j,k,s_inicial[j].cargaVh[k]);

                                }


                                printf("\n");
                                ///  distancia  + tiempo de atención
                                // printf(" %d %d %.2f\t Taten   ",j,k,sumataten);

                                //   printf("***%.2f\t esp***\n ",tespera);
                               distmastaten=sumadist+sumEspera +  sumataten;//sumadist+tespera +  sumataten
                               s_inicial[j].distMasTiAtenVh[k] = distmastaten;
```

```
                                    // printf("\n")



/*

          for(w=s_inicial[j].limiteInferior[k];w<=s_inicial[j].limiteSuperior[k];w++)
             {
             cli=s_inicial[j].ruta[w];

             if (s_inicial[j].ruta[w]!=0)
                {
               // printf("\n ----------------------------------------------------------------\n ");
                printf("%d\t[%d--%d]\n",cli,cliente[cli].e,cliente[cli].l);

                }



             }      //      printf("\n ----------------------------------------------------------------\n ");

             printf("\n");
*/
       }




  printf("\nreparar solución \n");

for(k=1;k<=datos.m;k++)
{
//printf("\n fTWVh %d  %d \n",k ,fTWVh[k]);

       if(fTWVh[k]!=0)
       {

            if(itera > 999)
              {

s_inicial[j]=expulsaVecinoLejano(s_inicial,j,k, mDistCli);

             printf("\nitera----------------------------------------------------------------\n");itera++;
getch();
              }
             if(itera%5==0)
            //s_inicial[j]=swapVhIntra(s_inicial,j,k);
            s_inicial[j]=parAdyacente(s_inicial,j,k);
       }
             else

itera--;
getch();
}
if(itera < 1)itera=100;

}while(bandera!=0);
getch();



   }
```

```
printf("\n ----------------------------------------------------------------\n ");

/*for(j=1;j<=datos.t;j++){
   for(k=1;k<=datos.m;k++){
          for( w=(s_inicial[j].limiteInferior[k]+1);w<(s_inicial[j].limiteSuperior[k]);w++)
             {
                if(s_inicial[j].ruta[w],cliente[s_inicial[j].ruta[w]].e>s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]
                   ||                    s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]+cliente[s_inicial[j].ruta[w]].d          >
cliente[s_inicial[j].ruta[w]].l)
             bandera= 1;
              }}}

///s_inicial[j].TiempAtenRealCli[s_inicial[j].ruta[w]]

*/
return(bandera);



}

///**********************************

solucion expulsaVecino(solucion p[DEP_MAX],int indiceDep,int indiceVh)
{
     int i,j,num1=0,num2,inf=0, sup=0,infOtro=0, supOtro=0,aux=0,indiceOtroVh,indiceVhSig;

   solucion pTemp[1];
   pTemp[0]=p[indiceDep];///vh actual



/*do{

 indiceOtroVh= (rand()%datos.m)+1  ; ///vh para cambio

}while( indiceVh ==  indiceOtroVh);

*/


   /// expulsar al clliente de esta vh mas lejano del dep y traer al cliente de otr5a vhb  que sea cercano al dep



   //indiceVh=(rand()%datos.m)+1;

                          //   costoVh(pTemp[0], cliente,deposito, datos, mDistCli,indiceDep,indiceVh);


                     /*  printf("LIMITES DENTRO\n");


                     for (i=1; i<=datos.t ; i++)
                       {
                         for (j=1; j<=datos.m; j++)
                         {
                          printf("[Inf]%d\t[Sup]%d\t",pTemp[0].limiteInferior[j],pTemp[0].limiteSuperior[j]);
                         }printf("\n");printf("\n");
                       }*/


indiceVhSig=indiceVh+1;
if(indiceVhSig>datos.m)
{
indiceVhSig=1;
///aqui caso especial
printf("\n caso especialVh %d Vsig %d ", indiceVh,indiceVhSig);
```

```
num1=pTemp[0].limiteSuperior[indiceVh];
num2=pTemp[0].limiteInferior[indiceVhSig];

printf("\n DEP %d VH %d   num1  %d  num2   %d",indiceDep, indiceVh,num1, num2);
printf("\n n1 %d n2 %d ",pTemp[0].ruta[num1],pTemp[0].ruta[num2]);



aux=pTemp[0].ruta[num1]   ;
      pTemp[0].ruta[num1]=pTemp[0].ruta[num2];
      pTemp[0].ruta[num2]=aux;

              printf("\n n1 %d n2 %d ",pTemp[0].ruta[num1],pTemp[0].ruta[num2]);
              printf("\n /////////////////////////// ");


}
else
{

printf("\nEXPULSA                Vh %d Vsig %d ", indiceVh,indiceVhSig);
    //inf=pTemp[0].limiteInferior[indiceVh]+1;
    num1=pTemp[0].limiteSuperior[indiceVh];
// printf("\n DEP %d VH %d inf  %d  sup   %d  num1   %d",indiceDep, indiceVh,inf,sup, num1);
    num2=pTemp[0].limiteInferior[indiceVhSig];
   // supOtro=pTemp[0].limiteSuperior[indiceOtroVh];

/// aqui busca al mas lejano al dep en vh actual y asinalo a num1
/// busca mas cercano al deop en vhotro y asignalo a mun2   y quita el rand

/*

    num1=inf+rand()%((sup-inf)+1);
    printf("\n DEP %d VH %d inf  %d  sup   %d  num1   %d",indiceDep, indiceVh,inf,sup, num1);



    num2=infOtro+rand()%((supOtro-infOtro)+1);
              printf("\n n1 %d n2 %d ",pTemp[0].ruta[num1],pTemp[0].ruta[num2]);

*/

  //  num1=inf+rand()%((sup-inf)+1);
   printf("\n DEP %d VH %d   num1  %d  num2   %d",indiceDep, indiceVh,num1, num2);



   // num2=infOtro+rand()%((supOtro-infOtro)+1);
              printf("\n n1 %d n2 %d ",pTemp[0].ruta[num1],pTemp[0].ruta[num2]);

    /*intercambio*/

      aux=pTemp[0].ruta[num1]   ;
      pTemp[0].ruta[num1]=pTemp[0].ruta[num2];
      pTemp[0].ruta[num2]=aux;

              printf("\n n1 %d n2 %d ",pTemp[0].ruta[num1],pTemp[0].ruta[num2]);

               printf("\n ********************************** ");

  ///actualización de inices
  printf("\n S %d P %d ",pTemp[0].limiteSuperior[indiceVh],pTemp[0].limiteInferior[indiceVhSig]);
 pTemp[0].limiteSuperior[indiceVh]-- ;
 pTemp[0].limiteInferior[indiceVhSig]--;

printf("\n S %d P %d ",pTemp[0].limiteSuperior[indiceVh],pTemp[0].limiteInferior[indiceVhSig]);

}
```

```
            getch();

///********************
///************************AGREGAR AQUI LAS 3 VALIDACIONES
///********************
///costo
/*COSTO
                suma=suma+mDistancias[0][p.ruta[p.limiteInferior[indice]]];//distancia del deposito al primer clliente
                for(j=inf;j<sup;j++)
                {
                        suma=suma+mDistancias[p.ruta[j]][p.ruta[j+1]];
                }
                suma=suma+mDistancias[p.ruta[j]][0];//distancia del ultimo cliente al deposito

                p.costoVh[indice]=suma;*/



///TW cliente
///Duracion de la ruta




///si el cambio es valido devolver pTemp  sino devolver  p

    return(pTemp[0]);
}
///*************************************************************************************************************************

solucion expulsaVecinoLejano(solucion p[DEP_MAX],int indiceDep,int indiceVh,float
mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX])
{
    int i,j,m,num1=0,num2,inf=0, sup=0,infOtro=0, supOtro=0,aux=0,indiceOtroVh,indiceVhSig,clienteTemp,pos;
float lejano;
   solucion pTemp[1];
   pTemp[0]=p[indiceDep];///vh actual


   /// expulsar al clliente de esta vh mas lejano del dep
   /// todos los calculos de distancia con relacion al dep

   clienteTemp=pTemp[0].limiteInferior[indiceVh]+1;
   lejano= mDistCli[s_inicial[j].ruta[clienteTemp]][cliente[datos.n+indiceVh].i];
                for (m=pTemp[0].limiteInferior[indiceVh]+1; m<=pTemp[0].limiteSuperior[indiceVh]; m++)
                {
                    clienteTemp=s_inicial[i].rutaTemp[m];

                    ///*********agregar a la condicion TRsalida del anterior
                    if(( mDistCli[cliente[datos.n+indiceVh].i][clienteTemp]) > lejano )
                    {

                     //faltan validaciones
                     lejano= mDistCli[cliente[datos.n+indiceVh].i][clienteTemp];
                     pos= m;
                    }
                }
  printf("\n lejano  pos %d ", pos);
  getch();
  ///cambar al cliente legido




///corregir la seleccion de las rutas , esto no es funciona bien
**************************************************************************
///solo hay que  validar que la otra ruta sea difernte de la elegida  en el
```

```
/// rango de 1 a datos.m o buscar un deposito cercano


indiceVhSig=indiceVh+1;
if(indiceVhSig>datos.m)
{
indiceVhSig=1;
///aqui caso especial
printf("\n caso especialVh %d Vsig %d ", indiceVh,indiceVhSig);
///esto esta mal se necesecitan los limites de dos rutas
num1=pTemp[0].limiteSuperior[indiceVh];
num2=pTemp[0].limiteInferior[indiceVhSig];

printf("\n DEP %d VH %d   num1  %d  num2   %d",indiceDep, indiceVh,num1, num2);
printf("\n n1 %d n2 %d ",pTemp[0].ruta[num1],pTemp[0].ruta[num2]);


///el intercambio esta conciderado en dos rutas del mismo dep
aux=pTemp[0].ruta[num1]   ;
      pTemp[0].ruta[num1]=pTemp[0].ruta[num2];
      pTemp[0].ruta[num2]=aux;
/// agregar en la impression a que ruta pertenece cada  cliente  posicion y numero de cliente

              printf("\n n1 %d n2 %d ",pTemp[0].ruta[num1],pTemp[0].ruta[num2]);
              printf("\n /////////////////////////// ");


}
else
{

printf("\nEXPULSA                Vh %d Vsig %d ", indiceVh,indiceVhSig);
    //inf=pTemp[0].limiteInferior[indiceVh]+1;
    num1=pTemp[0].limiteSuperior[indiceVh];
// printf("\n DEP %d VH %d inf  %d  sup   %d  num1   %d",indiceDep, indiceVh,inf,sup, num1);
    num2=pTemp[0].limiteInferior[indiceVhSig];
   // supOtro=pTemp[0].limiteSuperior[indiceOtroVh];

/// aqui busca al mas lejano al dep en vh actual y asinalo a num1
/// busca mas cercano al deop en vhotro y asignalo a mun2   y quita el rand

/*

    num1=inf+rand()%((sup-inf)+1);
    printf("\n DEP %d VH %d inf  %d  sup   %d  num1   %d",indiceDep, indiceVh,inf,sup, num1);



    num2=infOtro+rand()%((supOtro-infOtro)+1);
              printf("\n n1 %d n2 %d ",pTemp[0].ruta[num1],pTemp[0].ruta[num2]);

*/

  //  num1=inf+rand()%((sup-inf)+1);
   printf("\n DEP %d VH %d   num1  %d  num2   %d",indiceDep, indiceVh,num1, num2);



   // num2=infOtro+rand()%((supOtro-infOtro)+1);
              printf("\n n1 %d n2 %d ",pTemp[0].ruta[num1],pTemp[0].ruta[num2]);




    /*intercambio*/

      aux=pTemp[0].ruta[num1]   ;
      pTemp[0].ruta[num1]=pTemp[0].ruta[num2];
```

```
    pTemp[0].ruta[num2]=aux;

            printf("\n n1 %d n2 %d ",pTemp[0].ruta[num1],pTemp[0].ruta[num2]);

             printf("\n ********************************** ");

  ///actualización de inices
  printf("\n S %d P %d ",pTemp[0].limiteSuperior[indiceVh],pTemp[0].limiteInferior[indiceVhSig]);
 pTemp[0].limiteSuperior[indiceVh]-- ;
 pTemp[0].limiteInferior[indiceVhSig]--;

printf("\n S %d P %d ",pTemp[0].limiteSuperior[indiceVh],pTemp[0].limiteInferior[indiceVhSig]);

}


            getch();

///*******************
///***********************AGREGAR AQUI LAS 3 VALIDACIONES
///*******************
///costo
/*COSTO
                suma=suma+mDistancias[0][p.ruta[p.limiteInferior[indice]]];//distancia del deposito al primer clliente
                for(j=inf;j<sup;j++)
                {
                        suma=suma+mDistancias[p.ruta[j]][p.ruta[j+1]];
                }
                suma=suma+mDistancias[p.ruta[j]][0];//distancia del ultimo cliente al deposito

                p.costoVh[indice]=suma;*/

///TW cliente
///Duracion de la ruta


///si el cambio es valido devolver pTemp  sino devolver  p

    return(pTemp[0]);
}

/*


*/
/*
///*****************************************
///Esta fue una prueba usando ángulos, y no se está usando


void ruteoIniFact2(solucion s_inicial[DEP_MAX],conf_cli cliente[CLI_MAX + DEP_MAX],conf_dep
deposito[DEP_MAX],instancia datos,float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],float
anguloCliDep[DEP_MAX][CLI_MAX]){
int i,j,a,b,t;
float t2;
int listaAsignados[DEP_MAX][CLI_MAX];
int asignados=0;



///asina angulos  y ordena en funcion de ellos
for (i=1; i<=datos.t ; i++)
  {

    for (j=1; j<=s_inicial[i].numClientes; j++)
   {

     s_inicial[i].rutaAngulos[j]=anguloCliDep[i][s_inicial[i].rutaTemp[j]];
    }
```

```
   }
 for(i=1; i<=datos.t ; i++){

 for(a=1; a<s_inicial[i].numClientes;++a)
for(b=(s_inicial[i].numClientes); b>=a;--b){
if(s_inicial[i].rutaAngulos[b-1] > s_inicial[i].rutaAngulos[b]){
/*intercambio de los elementos*/
/*
t=s_inicial[i].rutaAngulos[b-1];
t2=s_inicial[i].rutaTemp[b-1];
s_inicial[i].rutaAngulos[b-1]=s_inicial[i].rutaAngulos[b];
s_inicial[i].rutaTemp[b-1]=s_inicial[i].rutaTemp[b];
s_inicial[i].rutaAngulos[b]=t;
s_inicial[i].rutaTemp[b]=t2;
}

}
}

}

*/



/*
///**********************************************
void ordenaTW(conf_cli cliente[CLI_MAX],int listaInicialTW[CLI_MAX],instancia datos){
int a,b,t,t2;
float tw[CLI_MAX];

for(a=1; a<=datos.n;++a){
	listaInicialTW[a]=a;
	tw[a]=cliente[a].e;

}
/*
for(a=1; a<=datos.n;++a){
	printf("%d  %f \n",a,tw[a]);
}
*/
/*
for(a=1; a<datos.n;++a)
for(b=(datos.n); b>=a;--b){
if(tw[b-1] > tw[b]){
/*intercambio de los elementos*/
/*
t=listaInicialTW[b-1];
t2=tw[b-1];
listaInicialTW[b-1]=listaInicialTW[b];
tw[b-1]=tw[b];
listaInicialTW[b]=t;
tw[b]=t2;
}

}
/*
for(a=1; a<=datos.n;++a){
	printf("%d  %d %f \n",a,listaInicialTW[a],tw[a]);
}
*/
/*
}
*/
/*
void distDep(instancia datos,float mDistDep[DEP_MAX][CLI_MAX],conf_cli cliente[CLI_MAX],conf_dep
deposito[DEP_MAX],float anguloCliDep[DEP_MAX][CLI_MAX],float mDistCli[CLI_MAX + DEP_MAX][CLI_MAX +
DEP_MAX])
{
 int i,j;//j son los clientes
```

```
   float xx,yy;
printf("\n"  );
  /// printf("\n  %d  ",s_actual.numClientes);
   for(i=1;i<=datos.t;i++)
      {
      for(j=1;j<=datos.n;j++)
      {

        mDistDep[i][j]= mDistCli[ i + datos.n-1][j];
         /* xx=(deposito[i].x-cliente[j].x)*(deposito[i].x-cliente[j].x);
          yy=(deposito[i].y-cliente[j].y)*(deposito[i].y-cliente[j].y);
          mDistDep[i][j] = sqrt(xx + yy );*/
/*
          anguloCliDep[i][j] = (180/3.14159)*atan2((cliente[j].y-deposito[i].y),cliente[j].x-deposito[i].x);//(180/3.14159)*
                                  //                    =atan2(datos.coordY[i]-datos.coordY[0],datos.coordX[i]-datos.coordX[0])*
(360/3.14159/2);

          if(anguloCliDep[i][j] < 0)
               anguloCliDep[i][j]  += 360;

//printf("ángulo %d  %d  %0.2f  \n",i,j,anguloCliDep[i][j]   );


      }
//printf("\n"  );
     }


}
*/


/*
imprime los datos de la instancia
for(k=1;k<=datos.n;k++){//lee instancia para cargar las coordenadas
      printf("%d\t%0.2f\t%0.2f\t%d\t%d\t%d\t%d\t
%d\t%d\t\n",cliente[k].i,cliente[k].x,cliente[k].y,cliente[k].d,cliente[k].q,cliente[k].f,cliente[k].a,cliente[k].e,cliente[k].l);



for (i=1; i<=datos.t ; i++)
  {
printf("\n");
      for (j=1; j<=s_inicial[i].numClientes; j++)
    {

       printf("\n %d -- %0.2f",s_inicial[i].rutaTemp[j],s_inicial[i].rutaAngulos[j]);
      }

  }
for(k=1;k<=datos.t;k++){//lee instancia para cargar las coordenadas
          printf(" %d\t ",deposito.D[k]);
          printf(" %d\t \n",deposito.Q[k]);


      }
for(k=1;k<=datos.n;k++){//lee instancia para cargar las coordenadas
          printf(" %d\t ",cliente[k].i);
          printf(" %f\t \n",cliente[k].x);


      }
for(k=1;k<=datos.t;k++){//lee instancia para cargar las coordenadas
          printf(" %d\t ",deposito.i[k]);
          printf(" %f\t \n",deposito.x[k]);


      }
*/
/*
```

```
for(k=1;k<=datos.n;k++){//lee instancia para cargar las coordenadas


            printf("\n1/%d  %.2f \t",k,mDistCli[1][k]);
            printf("2/%d  %.2f \t",k,mDistCli[2][k]);
            printf("3/%d  %.2f \t",k,mDistCli[3][k]);
            printf("4/%d  %.2f \t",k,mDistCli[4][k]);
            //printf("5/%d  %.2f \t",k,mDistCli[5][k]);

            printf("48/%d  %.2f ",k,mDistCli[48][k]);

        }
  ///   una prueba de impresin de las rutas de los depositos
for (i=1; i<=datos.t ; i++)
   {
printf("\n DEP %d    ",i);
       for (j=1; j<=s_inicial[i].numClientes; j++)
     {

        printf(" %d ",s_inicial[i].rutaTemp[j]);
       }

   }

 printf("\n ");


for(k=1;k<=5;k++){//lee instancia para cargar las coordenadas



            printf("48 %d  %.2f ",k,mDistCli[48][k]);
            printf("49 %d  %.2f ",k,mDistCli[49][k]);;
            printf("50 %d  %.2f ",k,mDistCli[50][k]);
            printf("51 %d  %.2f ",k,mDistCli[51][k]);
            printf("\n ");
        }
printf("\n ");
for(k=1;k<=5;k++){//lee instancia para cargar las coordenadas



            printf("48 %d  %.2f ",k,mDistDep[1][k]);
            printf("49 %d  %.2f ",k,mDistDep[2][k]);;
            printf("50 %d  %.2f ",k,mDistDep[3][k]);
            printf("51 %d  %.2f ",k,mDistDep[4][k]);
            printf("\n ");
        }



*/


///costo
/*COSTO
                suma=suma+mDistancias[0][p.ruta[p.limiteInferior[indice]]];//distancia del deposito al primer clliente
                for(j=inf;j<sup;j++)
                {
                        suma=suma+mDistancias[p.ruta[j]][p.ruta[j+1]];
                }
                suma=suma+mDistancias[p.ruta[j]][0];//distancia del ultimo cliente al deposito

                p.costoVh[indice]=suma;*/



/// aqui busca al mas lejano al dep en vh actual y asinalo a num1
/// busca mas cercano al deop en vhotro y asignalo a mun2   y quita el rand
```

```
/*
do{

 indiceOtroVh= (rand()%datos.m)+1  ; ///vh para cambio

}while( indiceVh ==  indiceOtroVh);


*/

    /// expulsar al clliente de esta vh mas lejano del dep y traer al cliente de otr5a vhb  que sea cercano al dep


    //indiceVh=(rand()%datos.m)+1;

                                //   costoVh(pTemp[0], cliente,deposito, datos, mDistCli,indiceDep,indiceVh);


                            /*  printf("LIMITES DENTRO\n");


                            for (i=1; i<=datos.t ; i++)
                               {
                                 for (j=1; j<=datos.m; j++)
                                 {
                                  printf("[Inf]%d\t[Sup]%d\t",pTemp[0].limiteInferior[j],pTemp[0].limiteSuperior[j]);
                                 }printf("\n");printf("\n");
                               }*/
```

Apéndice B CÓDIGO FUENTE VERIFICADOR DE SOLUCIUONES

```
  ///Para mejora envia la solucion a un archivo de texto
///
#include<conio.h>///quitar para linux
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
#include<time.h>
#include<string.h>
#define CLI_MAX 300//recordar add el númer de rutas porque hay un cero en el vector por cada ruta
#define DEP_MAX 30
#define VH_MAX 30

// 0(VRP) 1(PVRP) 2(MDVRP) 3(SDVRP) 4(VRPTW) 5(PVRPTW) 6(MDVRPTW) 7(SDVRPTW)

/*
m: number of vehicles
n: number of customers
t: number of days (PVRP), depots (MDVRP) or vehicle types (SDVRP)
The next t lines contain, for each day (or depot or vehicle type), the following information:
D: maximum duration of a route   **vectores de tamaño m
Q: maximum load of a vehicle     **vectores de tamaño m


*/

/*
falta validar que no exita violacion a las ventasnas de los
clientes  con un barrido

validar ventanas de los depositos   con bandera en el validador
printf("*    nota: a la solucion hay que agregar -1        *\n");
printf("*    al  final de linea                            *\n");

*/
/**************ESTRUCTURAS*********************************/
typedef struct
{
    int m;
    int n;
    int t;


}instancia;
instancia datos;

typedef struct
{

    int i;
    float x;
    float y;
    int d;
    int q;
    int f;
    int a;
```

```
    int list;
    int e;
    int l;
}conf_cli;
conf_cli cliente[CLI_MAX + DEP_MAX];



typedef struct
{
    int D;///es para  cada vehiculo del deposito  ****cambiar por esta en la validacion de de duracion de la ruta
    int Q;///es para  cada vehiculo del deposito
    int i;
    float x;
    float y;
    int d;
    int q;
    int f;
    int a;
    int e;
    int l;
}conf_dep;
conf_dep deposito[DEP_MAX];

typedef struct //guarda la estructura de la solucion inicial
{
    int ruta[CLI_MAX];    //conjunto de clientes que forman una rutae oreden de la configuracion
    float TiempAtenRealCli[CLI_MAX];    //conjunto de clientes que forman una rutae oreden de la configuracion

    int limiteSuperior[VH_MAX];//marcan los segmentos superiores de ruta en el vector.ruta
    int limiteInferior[VH_MAX];//marcan los segmentos inferiores de ruta en el vector.ruta
    double costoVh[VH_MAX];//costo de recorrido de un solo vehiculo
    double cargaVh[VH_MAX];
    float distMasTiAtenVh[DEP_MAX];
    int vh[VH_MAX];//numero de vehiculos (rutas creadas)
    int numClientes;//cantidad de clientes
    double costoSolucion;//costo solucion de todas las rutas generadas (valor de la Funcion Objetivo)

    int limitIniDep;//marcan los segmentos superiores de cada deposito
    int limitFinDep;//marcan los segmentos inferiores de cada deposito
    double TiempLlegDep;
}solucion;
solucion s_inicial[DEP_MAX];


typedef struct
{
 int rutaSolucion[CLI_MAX];
 float costoSolucionGlobal;

}solucionGlobal;
solucionGlobal sol;
char nombre[20];

void leeArchivos(char nombre[20]);
void leeSol();
void distCli(instancia datos,float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],conf_cli cliente[CLI_MAX]);
void escribeGraf();
int main(int argc, char **argv)
{
FILE *salida_txt;char nombre[20];
int type;
int j=1,k=1,w=1;
float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX];
float mDistDep[CLI_MAX][DEP_MAX];
int listaTabu[CLI_MAX];
int listaAncla[CLI_MAX];
float suma,sumadist,sumataten,sumadistTotal,distmastaten,tespera;
float solucion =0,solucionGlobal=0;
int cubeta[CLI_MAX];
```

```
int banderaRepetido=0;
int banderaFaltante=0;
int clientes;

printf("*****************************************************\n");
printf("*                                                   *\n");
printf("*    Evaluador para para soluciones de MDVRPTW   * \n");
printf("*                                                   *\n");
printf("*                                                   *\n");

printf("*****************************************************\n\n");
for(j=1; j<=datos.n;j++) cubeta[j]=0;

leeArchivos(nombre);

strcat(nombre, ".eva");
//sprintf(nombre, "salida%d.txt", rank);
     salida_txt = fopen(nombre,"w");
leeSol();
distCli(datos, mDistCli, cliente);

printf("\n         VALIDACION \n");
printf("\n");


for(j=1;j<=datos.t;j++){
   for(k=1;k<=datos.m;k++){printf(" DEP %d  RUTA %d \n",j,k);fprintf(salida_txt," DEP %d  RUTA %d \n",j,k);
    ///   cargas
      suma=0;///para carga
      sumataten=0;
      distmastaten=0;
      sumadist=0;
      tespera=0;
      sumadist=sumadist+mDistCli[s_inicial[j].ruta[s_inicial[j].limiteInferior[k]]][cliente[datos.n+j].i];

     if(s_inicial[j].limiteInferior[k]==s_inicial[j].limiteSuperior[k])
     {
       printf("0.00\n0.00    RUTA VACIA\n\n");fprintf(salida_txt,"0.00\n0.00    RUTA VACIA\n\n");
     }
     else
     {
         for(w=(s_inicial[j].limiteInferior[k]);w<(s_inicial[j].limiteSuperior[k]-1);w++)
            {
               sumadist=sumadist+mDistCli[s_inicial[j].ruta[w]][s_inicial[j].ruta[w+1]];
               suma=suma+cliente[s_inicial[j].ruta[w]].q;
               sumataten=sumataten+cliente[s_inicial[j].ruta[w]].d;
               clientes=s_inicial[j].ruta[w];
               cubeta[clientes]=cubeta[clientes]+1;
               if(s_inicial[j].TiempAtenRealCli[w+1]    > (s_inicial[j].TiempAtenRealCli[w] +   cliente[s_inicial[j].ruta[w]].d+
mDistCli[s_inicial[j].ruta[w]][s_inicial[j].ruta[w+1]] )  )
                  {
                  tespera=      tespera+((s_inicial[j].TiempAtenRealCli[w+1]   )      -   (s_inicial[j].TiempAtenRealCli[w]   +
cliente[s_inicial[j].ruta[w]].d+ mDistCli[s_inicial[j].ruta[w]][s_inicial[j].ruta[w+1]]));
                  }

if(cliente[s_inicial[j].ruta[w]].e<=s_inicial[j].TiempAtenRealCli[w]&&s_inicial[j].TiempAtenRealCli[w]+cliente[s_inicial[j].ru
ta[w]].d<=cliente[s_inicial[j].ruta[w]].l)
                     {printf("%d\t[%d] \t %4.1f--%4.1f  \t[%d]  \t OK \n",s_inicial[j].ruta[w],cliente[s_inicial[j].ruta[w]].e,
                     s_inicial[j].TiempAtenRealCli[w],s_inicial[j].TiempAtenRealCli[w]+cliente[s_inicial[j].ruta[w]].d,
                     cliente[s_inicial[j].ruta[w]].l);
                     fprintf(salida_txt,"%d\t[%d]          \t          %4.1f--%4.1f                    \t[%d]                    \t          OK
\n",s_inicial[j].ruta[w],cliente[s_inicial[j].ruta[w]].e,
                     s_inicial[j].TiempAtenRealCli[w],s_inicial[j].TiempAtenRealCli[w]+cliente[s_inicial[j].ruta[w]].d,
                     cliente[s_inicial[j].ruta[w]].l);
                      }
                     else
                        {printf("%d\t[%d] \t %4.1f--%4.1f  \t[%d]  \t X \n",s_inicial[j].ruta[w],cliente[s_inicial[j].ruta[w]].e,

s_inicial[j].TiempAtenRealCli[w],s_inicial[j].TiempAtenRealCli[w]+cliente[s_inicial[j].ruta[w]].d,cliente[s_inicial[j].ruta[w]].l
);
```

```
                        fprintf(salida_txt,"%d\t[%d]  \t  %4.1f--%4.1f  \t[%d]  \t  X
\n",s_inicial[j].ruta[w],cliente[s_inicial[j].ruta[w]].e,
                        s_inicial[j].TiempAtenRealCli[w],s_inicial[j].TiempAtenRealCli[w]+cliente[s_inicial[j].ruta[w]].d,
                        cliente[s_inicial[j].ruta[w]].l);
                         }
            }

if(cliente[s_inicial[j].ruta[w]].e<=s_inicial[j].TiempAtenRealCli[w]&&s_inicial[j].TiempAtenRealCli[w]+cliente[s_inicial[j].ru
ta[w]].d<=
                       cliente[s_inicial[j].ruta[w]].l)
                        {printf("%d\t[%d]  \t  %4.1f--%4.1f  \t[%d]  \t  OK
\n",s_inicial[j].ruta[w],cliente[s_inicial[j].ruta[w]].e,s_inicial[j].TiempAtenRealCli[w],
                            s_inicial[j].TiempAtenRealCli[w]+cliente[s_inicial[j].ruta[w]].d,cliente[s_inicial[j].ruta[w]].l);
                        fprintf(salida_txt,"%d\t[%d]  \t  %4.1f--%4.1f  \t[%d]  \t  OK
\n",s_inicial[j].ruta[w],cliente[s_inicial[j].ruta[w]].e,s_inicial[j].TiempAtenRealCli[w],
                            s_inicial[j].TiempAtenRealCli[w]+cliente[s_inicial[j].ruta[w]].d,cliente[s_inicial[j].ruta[w]].l);
                         }
                      else
                      {

                        printf("%d\t[%d]  \t  %4.1f--%4.1f  \t[%d]  \t  X
\n",s_inicial[j].ruta[w],cliente[s_inicial[j].ruta[w]].e,s_inicial[j].TiempAtenRealCli[w],
                            s_inicial[j].TiempAtenRealCli[w]+cliente[s_inicial[j].ruta[w]].d,cliente[s_inicial[j].ruta[w]].l);
                        fprintf(salida_txt,"%d\t[%d]  \t  %4.1f--%4.1f  \t[%d]  \t  X
\n",s_inicial[j].ruta[w],cliente[s_inicial[j].ruta[w]].e,s_inicial[j].TiempAtenRealCli[w],
                            s_inicial[j].TiempAtenRealCli[w]+cliente[s_inicial[j].ruta[w]].d,cliente[s_inicial[j].ruta[w]].l);

                      }


            sumadist=sumadist+mDistCli[s_inicial[j].ruta[w]][cliente[datos.n+j].i];
            suma=suma+cliente[s_inicial[j].ruta[w]].q;
            sumataten=sumataten+cliente[s_inicial[j].ruta[w]].d;
            ///impresion del ultimo cliente   *****no esta validada
           /* printf("%d\t[%d] \t %4.1f--%4.1f  \t[%d]  \t OK \n",s_inicial[j].ruta[w],cliente[s_inicial[j].ruta[w]].e,
                    s_inicial[j].TiempAtenRealCli[w],s_inicial[j].TiempAtenRealCli[w]+cliente[s_inicial[j].ruta[w]].d,
                    cliente[s_inicial[j].ruta[w]].l);
                    fprintf(salida_txt,"%d\t[%d]  \t  %4.1f--%4.1f  \t[%d]  \t  OK
\n",s_inicial[j].ruta[w],cliente[s_inicial[j].ruta[w]].e,
                    s_inicial[j].TiempAtenRealCli[w],s_inicial[j].TiempAtenRealCli[w]+cliente[s_inicial[j].ruta[w]].d,
                    cliente[s_inicial[j].ruta[w]].l);

*/

            clientes=s_inicial[j].ruta[w];
            cubeta[clientes]=cubeta[clientes]+1;
            solucion =solucion + sumadist;
            if(s_inicial[j].cargaVh[k]==suma)
               {printf("%.2f\t  Carga  OK\t",s_inicial[j].cargaVh[k]);fprintf(salida_txt,"%.2f\t  Carga
OK\t",s_inicial[j].cargaVh[k]);
                printf("\n");fprintf(salida_txt,"\n");
               }
            else
            {printf("%.2f\t  Carga  X  \t%.2f",s_inicial[j].cargaVh[k],suma);fprintf(salida_txt,"%.2f\t  Carga  X
\t%.2f",s_inicial[j].cargaVh[k],suma);
            printf("\n");fprintf(salida_txt,"\n");
            }

            distmastaten=sumataten+sumadist+tespera;
               if(s_inicial[j].distMasTiAtenVh[k] >distmastaten- 1.0 && s_inicial[j].distMasTiAtenVh[k] < distmastaten+ 1.0
)
               {
               printf("%.2f\t Duraci%cn de la ruta  OK\t",distmastaten,162);fprintf(salida_txt,"%.2f\t Duraci%cn de la ruta
OK\t",distmastaten,162);
                  if(s_inicial[j].distMasTiAtenVh[k] >  (deposito[j].D ))
                  {
                  printf("   ***%d",deposito[j].D);fprintf(salida_txt,"   ***%d",deposito[j].D);
                  }
               }
               else
```

```
                {
                printf("%.2f\t Duraci%cn de la ruta  X\t",distmastaten,162);fprintf(salida_txt,"%.2f\t Duraci%cn de la ruta
X\t",distmastaten,162);
                   if(s_inicial[j].distMasTiAtenVh[k] > (deposito[j].D))
                   {
                   printf("  %d",deposito[j].D);fprintf(salida_txt,"  %d",deposito[j].D);
                   }
                }
                printf("\n");fprintf(salida_txt,"\n");
                printf("\n");fprintf(salida_txt,"\n");
      }
       printf("\n");fprintf(salida_txt,"\n");
   }
}


   if(solucion > sol.costoSolucionGlobal - 0.5 &&  solucion < sol.costoSolucionGlobal + 0.5)
      {printf("Solucion Global %.2f\t   OK",solucion);fprintf(salida_txt,"Solucion Global %.2f\t   OK",solucion);
      }
   else
      {printf("Solución Global %.2f\t   X",solucion);fprintf(salida_txt,"Solucion Global %.2f\t   X",solucion);
      }
printf("\n");fprintf(salida_txt,"\n");
printf("--------------------------------------------------");fprintf(salida_txt,"--------------------------------------------------");
printf("\n");fprintf(salida_txt,"\n");
for(j=1; j<=datos.n;j++)
   {
  if(cubeta[j]>1)   banderaRepetido=1;
  if(cubeta[j]<1)   banderaFaltante=1;
   }
if(banderaRepetido==1)
 {
      printf("Repetidos los clientes  ");fprintf(salida_txt,"Repetidos los clientes  ");
      for(j=1; j<=datos.n;j++)
         if(cubeta[j]>1)
         {

          printf("%d  ",j);fprintf(salida_txt,"%d  ",j);
         }
      printf("\n\n");fprintf(salida_txt,"\n\n");
 }
 else printf("Repite clientes NO \n\n");fprintf(salida_txt,"Repite clientes NO \n\n");
 if(banderaFaltante==1)
 {
    printf("Faltan los clientes  ");fprintf(salida_txt,"Faltan los clientes  ");
      for(j=1; j<=datos.n;j++)
         if(cubeta[j]<1)
         {
         printf("%d  ",j);fprintf(salida_txt,"%d ",j);
         }
    printf("\n\n");fprintf(salida_txt,"\n\n");
 }
 else  {printf("Faltan  clientes NO \n\n");fprintf(salida_txt,"Faltan  clientes NO \n\n");
       }
 fclose(salida_txt);

escribeGraf();



return (0);
}
///*************************************************************************************************************
void leeArchivos(char nombre[20])
{
   FILE *archDatos;//DECLARACION DEL ARCHIVO TIPO FILE
   char NomArchivo[20]; //Para ver el nombre
   int k, j,iValor;
   int type;
   int cantidad=0;//numero de clientes leidos
```

```
    printf("Escriba el nombre  de la instancia  ");
     scanf("%s",NomArchivo);//lee archivo de lectura//se activan cuando te pida el archivo
     strcpy(nombre,NomArchivo);

    //archivo = NomArchivo;//asignación a archivo
    //if((archDatos = fopen("CH50.txt","rt"))==NULL)
    if((archDatos = fopen(NomArchivo,"rt"))==NULL)//comprobar que existe el archivo
        printf("No existe el archivo de datos");
       else{
        //printf(" El archivo es %s\n",archivo);
        fscanf(archDatos," %d\t ",&type);//lee el archivo de datos en sus tres columnas x, y, demanda
        fscanf(archDatos," %d\t ",&datos.m);
        fscanf(archDatos," %d\t ",&datos.n);
        fscanf(archDatos," %d\t ",&datos.t);
        for(k=1;k<=datos.t;k++){//lee instancia para cargar las coordenadas
            fscanf(archDatos," %d\t ",&deposito[k].D);
            fscanf(archDatos," %d\t ",&deposito[k].Q);

        }
                    for(k=1;k<=datos.n;k++){//lee instancia para cargar las coordenadas
fscanf(archDatos," %d\t ",&cliente[k].i);
            fscanf(archDatos," %f\t ",&cliente[k].x);
            fscanf(archDatos," %f\t ",&cliente[k].y);
            fscanf(archDatos," %d\t ",&cliente[k].d);
            fscanf(archDatos," %d\t ",&cliente[k].q);
            fscanf(archDatos," %d\t ",&cliente[k].f);
            fscanf(archDatos," %d\t ",&cliente[k].a);
            for(j=1;j<=cliente[k].a;j++)
            fscanf(archDatos," %d\t ",&cliente[k].list);
            fscanf(archDatos," %d\t ",&cliente[k].e);
            fscanf(archDatos," %d\t ",&cliente[k].l);
        }
        for(k=1;k<=datos.t;k++){//lee instancia para cargar las coordenadas
            fscanf(archDatos," %d\t ",&deposito[k].i);
            deposito[k].i=k;
            fscanf(archDatos," %f\t ",&deposito[k].x);
            fscanf(archDatos," %f\t ",&deposito[k].y);
            cliente[k+datos.n].i=k+datos.n;
            cliente[k+datos.n].x=deposito[k].x;
            cliente[k+datos.n].y=deposito[k].y;
            fscanf(archDatos," %d\t ",&deposito[k].d);
            fscanf(archDatos," %d\t ",&deposito[k].q);
            fscanf(archDatos," %d\t ",&deposito[k].f);
            fscanf(archDatos," %d\t ",&deposito[k].a);
            fscanf(archDatos," %d\t ",&deposito[k].e);
            fscanf(archDatos," %d\t ",&deposito[k].l);
        }
        fclose(archDatos);
        }
}
void distCli(instancia datos,float mDistCli[CLI_MAX+DEP_MAX][CLI_MAX+DEP_MAX],conf_cli cliente[CLI_MAX])
{
int i,j;//j son los clientes
    float xx,yy;
    for(i=1;i<=datos.n + datos.t;i++)
        {
        for(j=i+1;j<=datos.n + datos.t ;j++)
        {
            xx=(cliente[i].x-cliente[j].x)*(cliente[i].x-cliente[j].x);
            yy=(cliente[i].y-cliente[j].y)*(cliente[i].y-cliente[j].y);
            mDistCli[j][i] = mDistCli[i][j] = sqrt(xx + yy );
         }
      }
}
void leeSol()
{
    FILE *archDatos;//DECLARACION DEL ARCHIVO TIPO FILE
   // FILE *archSol;//DECLARACION DEL ARCHIVO TIPO FILE
    char NomArchivo[20]; //Para ver el nombre
    int k, j,contador;
```

```
    int cli,dep,veh,li,ls;
    char basura1;char basura2[1];
    float num;
    int bandera =1;
    int banderaRutaVacia =1;
    printf("Escriba el nombre  de la solución  ");
     scanf("%s",NomArchivo);//lee archivo de lectura//se activan cuando te pida el archivo
    //archivo = NomArchivo;//asignación a archivo
    //if((archDatos = fopen("CH50.txt","rt"))==NULL)
    if((archDatos = fopen(NomArchivo,"rt"))==NULL)//comprobar que existe el archivo
        printf("No existe el archivo de datos");
       else{
        fscanf(archDatos," %f\t ",&num);//lee el archivo de datos en sus tres columnas x, y, demanda
        sol.costoSolucionGlobal=num;

            for(j=1;j<=datos.t;j++){
            li=1;
                  for(k=1;k<=datos.m;k++){
                     fscanf(archDatos," %d\t ",&dep);
                     fscanf(archDatos," %d\t ",&veh);
                     fscanf(archDatos," %f\t ",&num);
                     s_inicial[dep].distMasTiAtenVh[veh]=num;
                     fscanf(archDatos," %f\t ",&num);
                     s_inicial[dep].cargaVh[veh]=num;
                     s_inicial[dep].limiteInferior[veh] =li;
                     ls=li;
                     contador=1;
                                 bandera =1;
                       banderaRutaVacia =1;
                     do{
                     fscanf(archDatos," %d\t ",&cli);

                          if(cli>datos.n){         ///si es depodito
                           bandera--;
                           fgetc(archDatos);
                           fscanf(archDatos," %f\t ",&num);
                           s_inicial[dep].TiempLlegDep=num;
                           fgetc(archDatos);
                                    }
                        else{                      /// si es cliente
                           s_inicial[dep].ruta[ls]=cli;
                           fgetc(archDatos);
                           fscanf(archDatos," %f\t ",&num);
                           s_inicial[dep].TiempAtenRealCli[ls]=num;
                           fgetc(archDatos);
                             contador++;
                           ls++;
                           banderaRutaVacia =0;
                        }

                        }while(bandera != -1);

                       s_inicial[dep].limiteSuperior[veh] =ls;
                       s_inicial[dep].vh[veh]=contador;
                       li=ls+1;
                     }

          }
          fclose(archDatos);
          // printf("proceso realizado con exito");
          }
}

void escribeGraf()
{
 int j,k,w,dep,cli;
 FILE *graf_txt;
    // printf("RUTA\n\n\n");
graf_txt = fopen("grafEva.txt","w");
 if (!graf_txt) {
```

```
        return -1;
    }


 for(j=1;j<=datos.t;j++)
    {

    for(k=1;k<=datos.m;k++)
        {
        dep=j+datos.n;
        printf("\n%f\t%f ",cliente[dep].x,cliente[dep].y);
        fprintf(graf_txt,"%f\t%f\t[%d--%d] \n",cliente[dep].x,cliente[dep].y,j,k);
            for(w=s_inicial[j].limiteInferior[k];w<=s_inicial[j].limiteSuperior[k];w++)
                {
                cli=s_inicial[j].ruta[w];

                if (s_inicial[j].ruta[w]!=0)
                    {
                   // printf("\n ----------------------------------------------------------------\n ");
                    printf("%f\t%f\t[%d--%d]\n",cliente[cli].x,cliente[cli].y,cliente[cli].e,cliente[cli].l);
                    fprintf(graf_txt,"%f\t%f\t[%d--%d]\n",cliente[cli].x,cliente[cli].y,cliente[cli].e,cliente[cli].l);
                    }



                }      //      printf("\n ----------------------------------------------------------------\n ");

                printf("\n%f\t%f ",cliente[dep].x,cliente[dep].y);
                fprintf(graf_txt,"%f\t%f\n ",cliente[dep].x,cliente[dep].y);


                printf("\n");
                fprintf(graf_txt,"\n");


        }


    }
     fclose(graf_txt);
}
```