UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

CENTRO DE INVESTIGACIÓN EN INGENIERÍA Y CIENCIAS APLICADAS

Algoritmo distribuido de Recocido Simulado para el modelo del transporte vehicular con capacidades homogéneas

TESIS PROFESIONAL PARA

OBTENER EL GRADO DE:

DOCTOR EN INGENIERÍA Y CIENCIAS APLICADAS

CON OPCIÓN TERMINAL EN TECNOLOGÍA ELÉCTRICA

PRESENTA:

M.I. Jesús del Carmen Peralta Abarca

Asesor de Tesis: Dr. Marco Antonio Cruz Chávez

Co - Asesor: Dr. Martín Heriberto Cruz Rosales

Cuernavaca, Morelos; enero 2017.

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

INSTITUTO DE INVESTIGACIÓN EN CIENCIAS BÁSICAS Y APLICADAS
COORDINACIÓN DE POSGRADO EN INGENIERÍA Y CIENCIAS APLICADAS

ASUNTO: APROBACIÓN DE TESIS
Cuernavaca, Morelos., 17 de agosto de 2016
OFICIO No. 17/2016

JESÚS DEL CARMEN PERALTA ABARCA
PRESENTE

Por este conducto le notifico que su tesis de Doctorado titulada,

"Algoritmo Distribuido de Recocido Simulado para el Modelo del Transporte Vehicular con Capacidades Homogéneas"

Fue aprobada en su totalidad por el jurado revisor y examinador integrado por los ciudadanos

NOMBRE	FIRMA
DRA. MARGARITA TECPOYOTL TORRES	
DR. ÁLVARO ZAMUDIO LARA	
DR. MARTÍN GERARDO MARTÍNEZ RANGEL	
DR. FEDERICO ALONSO PECINA	
DR. BEATRIZ MARTÍNEZ BAHENA	
DRA. MARTÍN HERIBERTO CRUZ ROSALES	
DR. MARCO ANTONIO CRUZ CHÁVEZ	

Por consiguiente, se autoriza a editar la presentación definitiva de su trabajo de investigación para culminar en la defensa oral del mismo.

Sin otro particular aprovecho la ocasión para enviarle un cordial saludo.

A T E N T A M E N T E
"Por una Humanidad Culta"

JEFATURA DE POSGRADO EN INGENIERÍA Y CIENCIAS APLICADAS
CIICAp

DR. JOSÉ ALFREDO HERNÁNDEZ PÉREZ
COORDINADOR DE POSGRADO DE INGENIERÍA Y CIENCIAS APLICADAS

Av. Universidad 1001, Col. Chamilpa, Cuernavaca, Morelos, México, C.P. 62209
Tel: (+52) 777 329 7000, ext. 6212 (+52) 777 329 7084
Correo: *ciicap.posgrado@uaem.mx*

DISTINTIVO Empresa Incluyente

Agradecimientos

A…

Dios Padre Todopoderoso, creador del cielo y la tierra y a Jesucristo su único hijo, por toda su bondad y misericordia, hacia mi persona y mi familia. Sólo tú Señor sabes lo que me conviene, hágase tu voluntad y no la mía.

Mi esposo Ing. Salomón Paul, por haber sacrificado sus metas en favor de las mías, por todas las desventuras y alegrías que pasamos en la realización de este proyecto. Gracias por tanto amor y cariño, gracias a Dios que te tengo a mi lado. Te amo mucho.

Mis hijos Alan Salomón, Paul Paris y a ti que me tienes en algún lugar de tu corazón; por ser la razón de mi vida.

Mis padres Micaela y José Ángel por darme la vida y la oportunidad de estudiar. Gracias por todo.

Mi asesor de tesis, Dr. Marco Antonio Cruz Chávez, por su apoyo, consejos y paciencia que me mostró durante el desarrollo de este trabajo; gracias de todo corazón.

El maestro Alfonso D'Granda, te debo mucho, gracias por tu infinito apoyo.

La Mtra. Juana Enríquez Urbano y a su esposo el Mtro. Daniel Frías, por su gran ayuda, gracias mil.

Los integrantes de mi Comité Revisor: Dra. Margarita Tecpoyotl Torres, Dra. Beatriz Martínez Bahena, Dr. Federico Alonso Pecina, Dr. Álvaro Zamudio Lara, Dr. Gerardo Martínez Rangel, Dr. Heriberto Cruz Rosales.

Mis compañeros: Pedro Moreno, Ariadna Ortiz, Jazmín Juárez, Beatriz Martínez, Jesús Colín, Blanquita Flores, Boris Jiménez y Ricardo Casasola.

CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

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.

CAPÍTULO 1 INTRODUCCIÓN

1.1 Antecedentes

Las empresas en México y en el mundo, deben enfrentar diariamente problemas relacionados con el uso eficiente u optimización de sus recursos, entre éstos se encuentran los referentes a los de servicios enfocados a la logística del transporte del producto, (materiales, personas, residuos, materia prima, productos, etc.), de un destino a otro que además se encuentran dispersas geográficamente.

Este escenario se presenta desde hace tiempo en el ámbito logístico pero en años recientes ésta 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.

Las empresas del transporte son conscientes de esta necesidad y de sus beneficios. Se basan en la experticia de sus administradores para definir las rutas de sus flotillas (Muñoz, Claderón Sotero, & Hernán, 2009); el hacerlo de manera empírica ha generado bastantes situaciones delicadas tales como accidentes fatales debido a que los choferes deben de ir de un lugar a otro sin descanso, el tener que regresar transportes “de vacío” o que no tengan camiones para trasladar producto hacia puntos de distribución o los clientes, exceso de carga, daños en la mercancía, entre otras situaciones.

El planteamiento inicial del problema de la logística de distribución en una empresa consiste en encontrar un plan de entrega de bienes/servicios a un conjunto de clientes geográficamente dispersos minimizando el costo de recorrido de los vehículos considerando además las diversas 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

El transporte por lo general representa el elemento más importante en los costos de logística para la mayoría de las empresas, así que un sistema eficiente y económico de transporte contribuye a una mayor competencia en el mercado. Un sistema de transporte poco desarrollado limita la amplitud de su mercado porque el área que puede abarcar se reduciría a solo un pequeño entorno de su punto de producción. A este inconveniente se enfrentan diariamente cientos de distribuidores de todo el mundo y tiene por consiguiente una significativa importancia económica.

Los costos de transporte se encuentran entre uno y dos tercios de los costos distribución de mercancías totales, mejorar la eficiencia mediante el máximo uso del equipo de transporte y de su personal es importante hoy en día. El transporte de bajo costo y de alta calidad también impulsa, de una forma indirecta, la competencia, al hacer que los bienes estén disponibles en un mercado que normalmente no podría solventar este precio.

Así, un problema frecuente en la toma de decisiones logísticas, es reducir los costos de transporte y mejorar el servicio al cliente encontrando los mejores caminos que debería seguir un vehículo 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).

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.

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 más famosos problemas de optimización combinatoria y ha sido intensamente estudiado debido a muchas aplicaciones prácticas en el campo de la logística (Pop, Pop, Zelina, Lupse, & Chira, 2011) ya que consiste en un diseño óptimo de redes logísticas, los llamaremos *rutas* de ahora en adelante, para la entrega o recolección de bienes o personas desde un depósito central hacia un conjunto de puntos geográficamente dispersos, el cual está sujeto a varias limitaciones, tales 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).

Este tipo de problema fue analizado por los años cincuenta por el científico estadounidense George Bernard Dantzig y con esto creó un amplio campo de investigación; aunque no existe una definición que sea aceptada por la comunidad científica de lo que es VRP debido a una diversidad de restricciones encontradas en la práctica.

La mayoría de los investigadores se han concentrado en una versión estandarizada del problema que se ha llamado VRP clásico. 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.

Tres de sus aplicaciones más importantes del VRP son:

Robótica: Para minimizar la cantidad de desplazamientos que una máquina debe realizar para perforar una tarjeta de circuito integrado, cada desplazamiento posee un costo, por lo que se busca que estos desplazamientos realizados sean mínimos.

Industria: En la secuenciación de tareas que una máquina debe realizar con un mínimo de tiempo posible.

Áreas de logística de transporte: En diversos tipos de situaciones es necesario conocer la ruta mínima desde un punto de origen, pasando por todos los clientes una sola vez y finalizar en el origen. En el diseño del ruteo vehícular se busca minimizar el consumo de combustible, el tiempo: de reparto, la recolección de basura, transporte público, etc.

Algunas de las empresas que usan el modelo del VRP son: la entrega de alimentos y bebidas a tiendas de abarrotes o CeDis (Centros de Distribución) (Finnish Food Safety Authority (EVIRA) y Omnitracs México), transporte escolar y de personal (QS3), recolección de basura (Waste Management USA), empresas de mensajería (Logística de Nuevo León), etc.

1.1.1 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)

En esta investigación se trabajará con el modelo de CVRP, el cual se describe como un conjunto de n clientes, un depósito único, las distancias entre los clientes y el depósito 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.

Los modelos presentados se explicarán en la siguiente sección para conocer su origen y su forma de trabajar~~,~~ se realiza una revisión del estado del arte de la evolución del Problema de Ruteo Vehicular comenzando con el Problema del Agente Viajero (TSP) planteado en 1956 hasta los trabajos recientes.

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).

El problema del Agente Viajero ***TSP*** *(Travelling Salesman Problem* por sus siglas en inglés) fue introducido por Flood en 1956. *El problema recibe este nombre porque puede describirse en términos de un agente vendedor que debe visitar cierta cantidad de ciudades en un solo viaje, de tal manera que inicie y termine su recorrido en la ciudad "origen"; el agente debe determinar cuál ruta debe seguir para visitar cada ciudad una sola vez y regresar de tal manera que la distancia total recorrida sea mínima.* (Rocha, González, & Orjuela, 2011).

Dantzing y Ramser generalizaron en 1959 el problema en el cual se modela la venta de combustible a través de una flota de camiones a diferentes estaciones de servicio desde un depósito. Este trabajo fue la base para un desarrollo posterior de otros planteamientos que han ido ampliando el número de variables y restricciones. (Rocha, González, & Orjuela, 2011) (Olivera, 2004)

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. Lo anterior se puede observar en la Figura 1.

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).

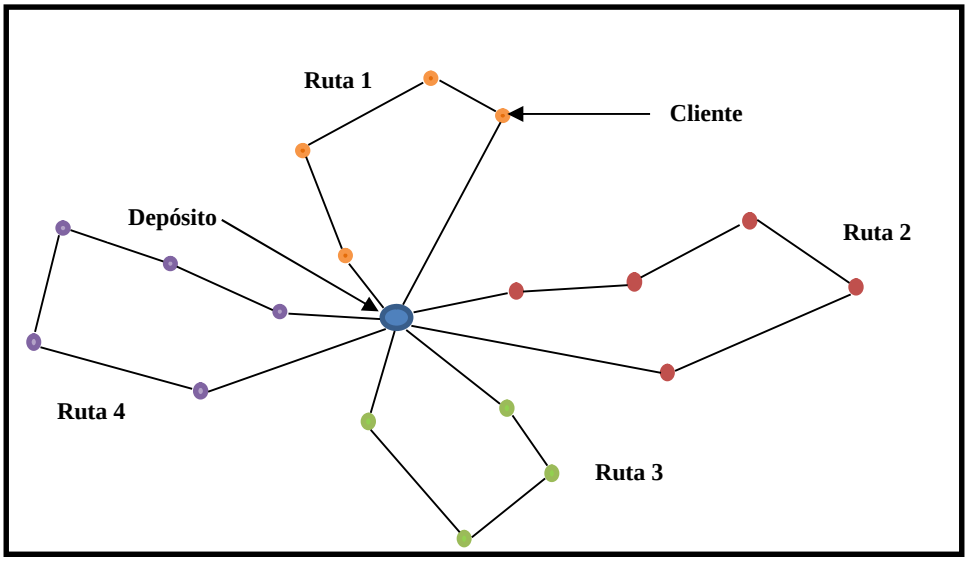


Figura 1 Modelo m-TSP o VR.

El **TSP** generalizado puede ser asumido como un VRP o como un CVRP (siglas en inglés del Problema de Ruteo Vehicular con Capacidades Homogéneas Capacited Vehicle Routing Problem), es decir como un problema de ruteo de vehículos donde la capacidad de la flota se convierte en restricción para la formulación del problema. La función objetivo del CVRP es minimizar el costo total para surtir a todos los consumidores que tienen una demanda conocida (determinística), todos los vehículos son iguales en capacidad y salen de un centro de distribución.

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.

Un modelo CVRP se denomina Simétrico (Symmetric CVRP, SCVRP) cuando el costo de ir de un cliente i al cliente j es igual al costo de ir del cliente j al cliente i.

En caso contrario se denomina CVRP Asimétrico (Asymmetric CVRP, ACVRP), una aplicación real de los asimétricos son las calles de un solo sentido. (Bianchi, Birattari, Chiarandini, Manfrin, & Mastrolili, 2004)

En la Figura 2, se muestra el VRP asimétrico. En azul el camino de ir del punto verde al rojo, en rojo el recorrido inverso, porque el sentido de las calles no permite realizar el mismo recorrido.

Figura 2. Representación gráfica de los tipos de VRP.

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.

El TSP probabilístico (PTSP), el cual surge del trabajo de Tillman en 1969, pretende encontrar el recorrido mínimo a través de un conjunto de nodos (clientes) con probabilidades asociadas al uso o no de los servicios. Éste junto con el modelo de m-TSP generan otros dos modelos, el Problema del agente viajero Múltiple con Ventanas de Tiempo (m-TSPTW) y el Problema del Agente viajero Múltiple Probabilístico (m-PTSP).

En la Figura 3 se presenta un cuadro en donde se resumen los tipos de VRP arriba mencionados:

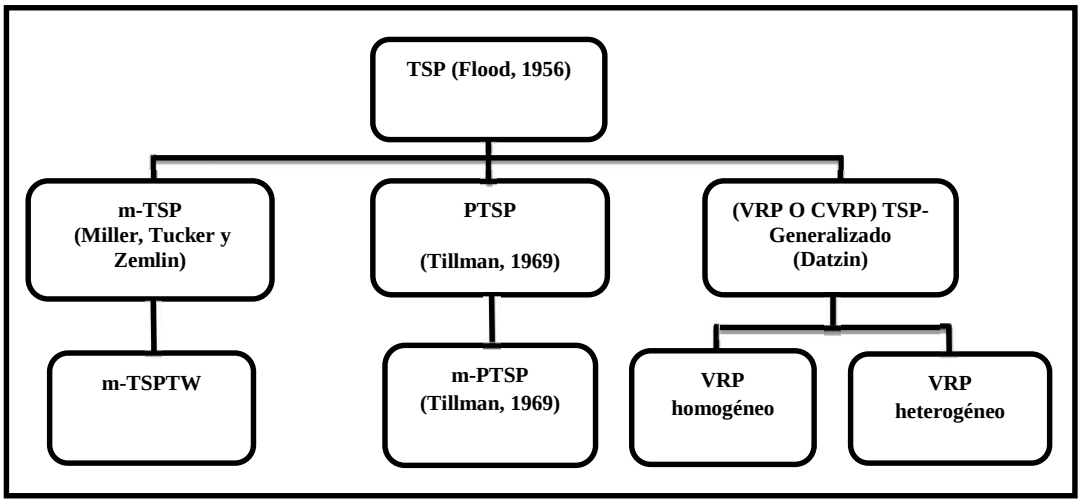


Figura 3. Modelos del VRP.

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 4 resume los modelos arriba comentados.

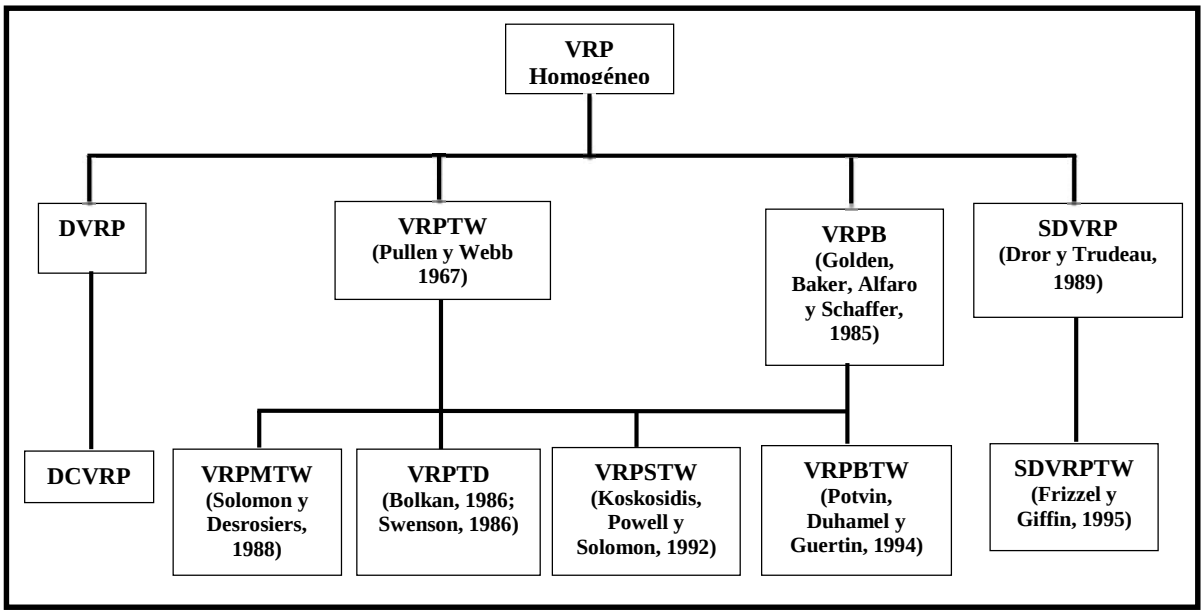


Figura 4. Modelos del VRP Homogéneo

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 5.

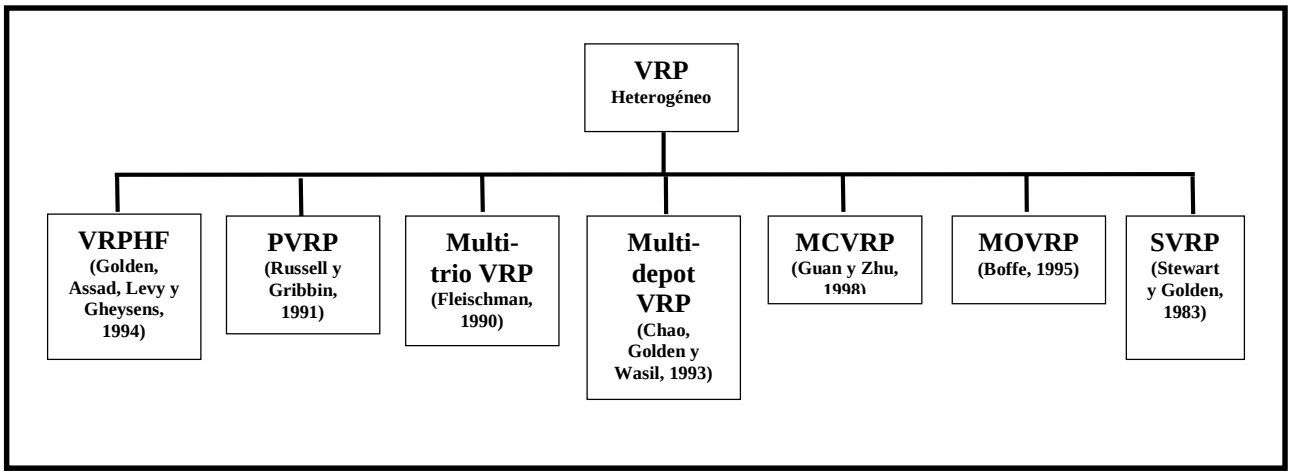


Figura 5. Modelos del VRP Heterogéneo

1.2 Complejidad del problema CVRP

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. Este grupo incluye problemas como el del agente viajero, el problema de la mochila, el transporte, la asignación de horarios para cursos universitarios o la asignación de maquinaria en talleres de manufactura.

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).

[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.

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.

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.

Como se mencionó en el inciso 1.1.1 el problema del VRP es una extensión del TSP y éste 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.

Los problemas VRP son problemas que están ubicados dentro de la optimización combinatoria y son considerados del tipo NP-completos (Daza, Montoya, & Narducci, 2009) (Dasgupta, Papadimitriou, & Vazirani, 2006).

[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.

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

1.3 Planteamiento del Problema

De manera general las empresas en México, no aplican un método para la planeación de sus rutas para el reparto de sus productos a sus clientes, esto conlleva a que no se tiene una metodología que le permita visualizar dónde iniciar el reparto y dónde terminarlo sin dejar fuera a ningún cliente o evitar entregas irregulares con la consecuente pérdida económica y de recursos (Guerrero-Campanur, Pérez-Loaiza, & Olivares-Benitez, 2011).

1.4 Hipótesis

El usar una heurística como Recocido Simulado con reinicio y aplicando estructuras de vecindades híbridas dentro de su búsqueda local, permitirá obtener estrategias de solución eficientes y eficaces al Problema del Ruteo Vehicular con Capacidades Homogéneas (CVRP).

1.5 Objetivo General

Desarrollar un algoritmo con distribución de procesos utilizando una estructura híbrida de vecindad que sea eficiente y eficaz en su desempeño para el problema de Ruteo Vehicular con Capacidades Homogéneas que se ejecute en un clúster computacional.

Objetivos Específicos

1. Diseñar y programar el algoritmo secuencial de Recocido Simulado con reinicio, y una estructura de vecindad que sea eficiente, que permita obtener soluciones eficaces para el problema del CVRP.
2. Realizar el análisis estadístico para la validación de los resultados del algoritmo propuesto con otras metaheurísticas aplicadas.
3. Programar el algoritmo de Recocido Simulado con reinicio, 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 150 clientes.
4. Medir la eficacia y speep-up del algoritmo distribuido para valorar su ejecución al trabajar con diferente número de procesadores.

1.6 Alcance de la investigación

La investigación que se realiza, trabaja con el modelo del Transporte Vehicular con Capacidades Homogéneas (por sus siglas en inglés Capacitated Vehicle Routing Problem, CVRP) aplicando la metaheurística del algoritmo de Recocido Simulado con una estructura híbrida de vecindad y reinicio. 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.

1.7 Contribucción

Las aportaciones de la presente investigación doctoral son:

a) El algoritmo de Recocido Simulado con reinicio para el modelo de ruteo vehicular con capacidades homogéneas, que presente buena eficiencia y eficacia.
b) Inclusión de una heurística computacional que implementa una estructura híbrida de vecindad.
c) La evaluación del desempeño de la distribución propuesta para el algoritmo de recocido simulado con reinicio.
d) El diseño y programación de un nuevo movimiento de perturbación para la generación de soluciones vecinas para ser aplicado en la estructura híbrida de vecindad.
e) La paralelización del algoritmo de recocido simulado con reinicio y una estructura híbrida de vecindad.

1.8 Organización de la Tesis

El trabajo ha sido organizado de la manera siguiente;

- Capítulo 1. Se presentan los antecedentes del problema CVRP, 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 metaheurística de recocido simulado 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, la paralelización de algoritmos y el análisis de rendimiento del algoritmo paralelo.
- Capítulo 5. Se muestran los resultados experimentales obtenidos de la ejecución de los algoritmos secuencial y distribuido de recocido simulado 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 MÉTODOS APLICADOS PARA LA SOLUCIÓN DEL VRP

2.1 Introducción a las Técnicas de Optimización

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

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

Las decisiones son parte de la vida diaria y se hacen necesarias en todas partes en donde haya una problemática, a medida que el mundo se vuelve más complejo y competitivo, las decisiones deben tomarse de una manera más racional y eficiente. La toma de decisiones es una metodología que sigue una serie de pasos, para un proceso de optimización de un problema puede representarse de la forma siguiente (Figura 6):

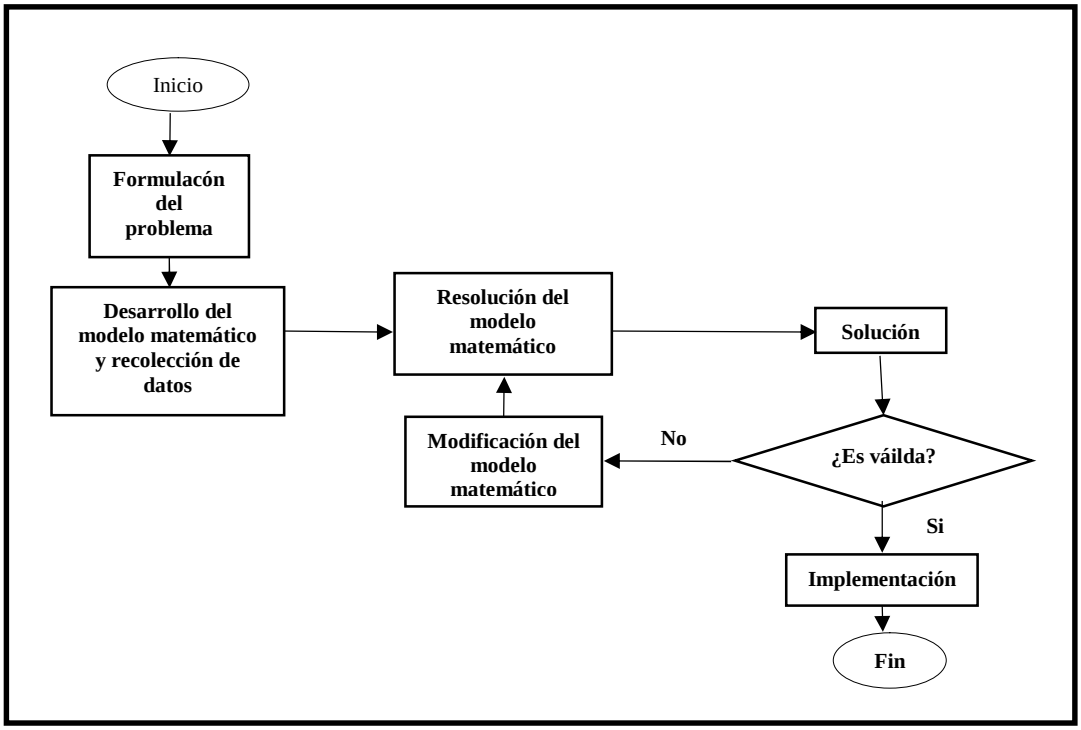


Figura 6. Proceso de Optimización de un problema

Formular el problema: En este primer paso se identifica un determinado problema de decisión. A continuación, se realiza una definición inicial del mismo. Esta primera definición puede ser imprecisa. El objetivo principal es esbozar los primeros factores internos y externos, junto con los objetivos que se quieren optimizar.

Modelado del problema: En este paso se construye un modelo matemático resumido del problema. El modelador puede inspirarse en modelos similares presentes en la literatura. Esto reduce el problema a simples modelos de optimización conocidos. Normalmente, estos modelos son simplificaciones de la realidad y se realizan aplicando aproximaciones y esquivando, a veces, procesos demasiado complejos. De aquí pueden surgir cuestiones como ¿por qué resolver de forma exacta problemas que no son exactos por naturaleza?

Resolución del problema: Una vez modelado el problema, el proceso de resolución genera una "buena" solución para el problema. La solución puede ser óptima o cuasi-óptima. Nótese como se está buscando una solución para un modelo resumido del problema y no para el problema real original. Por ello, la solución obtenida para el modelo resumido es válida cuando el modelo está bien diseñado y en ocasiones hay

que modificar el modelo con base a los resultados obtenidos. El diseñador del algoritmo puede aplicar otros algoritmos bien conocidos o integrar el conocimiento obtenido en uno de ellos.

Implementación de una solución: La solución obtenida es probada y analizada por el tomador de decisiones e implementada siempre que sea de una calidad "aceptable". Si disponemos de nueva información práctica, esta puede ser introducida en la solución y re-implementarse. Si por el contrario, la solución obtenida no es aceptable, el modelo y/o el algoritmo debe ser mejorado y el proceso de toma de decisiones repetido.

2.2 Clasificación de las Técnicas Metaheurísticas

Cuando se desea resolver un problema de optimización existen dos maneras básicas de abordarlo, 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, sino que se han desarrollado una gran cantidad de métodos de optimización para resolver diferentes tipos de problemas.

Sin embargo, muchos de los problemas que se presentan en la práctica no pueden ser resueltos mediante métodos exactos; debido a su complejidad, el tiempo requerido para hacerlo es extremadamente alto aumentando exponencialmente con el tamaño del problema.

Teniendo en cuenta la complejidad de un determinado problema de optimización, este puede ser abordado mediante un método exacto o uno aproximado (Figura 7). 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.

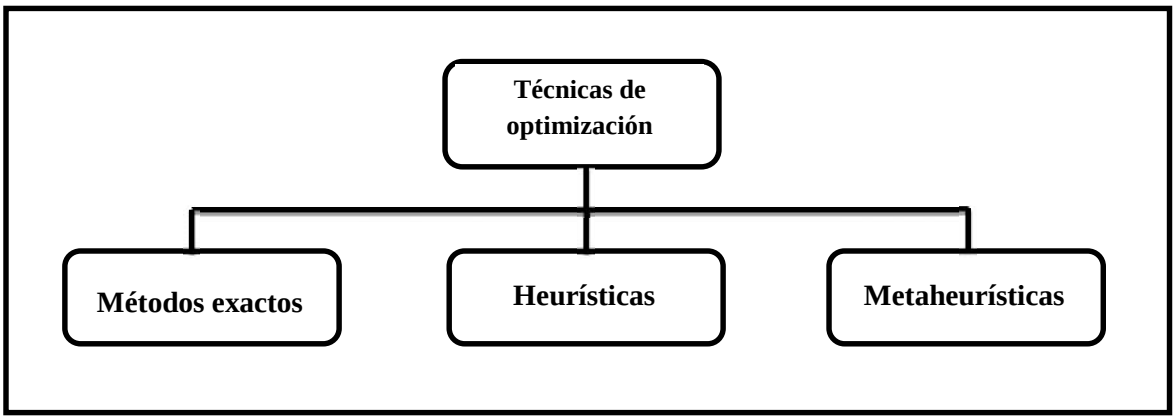


Figura 7. Clasificación de las Técnicas de Optimización.

2.2.1 Métodos Exactos

Los métodos exactos, son aquellos que conducen a una solución óptima exacta para el problema combinatorio, empleando técnicas que reduzcan la búsqueda de soluciones.

Dentro de los métodos exactos se pueden encontrar los siguientes tipos de algoritmos clásicos:

1. Búsqueda directa de árbol (de ramificación)
2. Los basados en programación dinámica,
3. Y en programación lineal (con restricciones).

Estos métodos enumerativos pueden verse como algoritmos de búsqueda basada en árboles donde dicha exploración se realiza sobre el espacio de búsqueda que interesa, y donde el problema se resuelve dividiéndolo en problemas más simples.

Búsqueda directa de árbol (de ramificación) se basan en una enumeración implícita de todas las soluciones del problema de optimización considerado. El espacio de búsqueda es explorado mediante la construcción dinámica de un árbol cuyo nodo raíz representa el problema que se está resolviendo y el correspondiente espacio de búsqueda explorado. Los nodos hoja representan potenciales soluciones

y los nodos internos posibles subproblemas. La poda del árbol de búsqueda se realiza a través de funciones de delimitación que podan subárboles que no contengan ninguna solución óptima.

La programación dinámica se basa en la división recursiva del problema en subproblemas más simples. Este método de optimización por etapas es el resultado de una secuencia de decisiones parciales. El proceso evita una enumeración total del espacio de búsqueda mediante la poda de secuencias de decisiones parciales que no conduzcan a soluciones óptimas.

La Programación Lineal (PL) (con restricciones) consiste en modelar un problema de optimización utilizando, únicamente, inecuaciones lineales. Es un lenguaje construido alrededor de conceptos como los árboles de búsqueda y las implicaciones lógicas. Un problema de PL consta de una función objetivo (lineal) por maximizar o minimizar, sujeta a ciertas restricciones en la forma de igualdades o desigualdades. La optimización de problemas mediante la programación con restricciones se realiza a través de la definición de un conjunto de variables que, a su vez, están ligadas a un conjunto de restricciones. Las variables toman sus valores de un dominio de valores finito y las restricciones pueden venir representadas por símbolos o expresiones matemáticas.

Michael Gendreau afirma que los métodos exactos son eficientes en problemas hasta de 50 clientes debido a restricciones de tiempo computacional (Nabila Azi, 2010).

2.2.2 Heurísticas

En la década de los setentas, se empezaron a difundir diferentes métodos de resolución de problemas a los cuales les llamaron técnicas heurísticas. Estas técnicas son de especial interés en el caso de variables enteras, aunque también se aplican a problemas con variables reales.

Se ha definido a la heurística 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.

Desde un punto de vista práctico, para aquellos problemas donde la búsqueda exhaustiva sea ineficiente o para aquellos problemas donde la cardinalidad del espacio de búsqueda, aumenta exponencialmente con el tamaño del problema, es necesaria la utilización de algoritmos heurísticos.

2.2.3 Metaheurísticas

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.

Algunas características de las metaheurísticas son las siguientes: (Sadiq M. Sait, 2000)

- Se les debe guiar a la solución óptima porque son “ciegas”, hay que indicarles cuándo deben terminar.
- Son algoritmos aproximados, no garantizan la obtención de una solución óptima.
- Aceptan malos movimientos para acceder a nuevas regiones no exploradas.
- Solo necesitan una solución inicial (o un conjunto de ellas), un espacio de soluciones y un mecanismo para explotar el campo de soluciones.
- Se pueden aplicar en la resolución de cualquier problema de optimización de tipo combinatorio.

Existen muchas posibles clasificaciones para organizar las distintas metaheurísticas. En esta sección se centrará en la que las organiza teniendo en cuenta el número de soluciones que optimiza: metaheurísticas basadas en una única solución.

Cuando se resuelven problemas de optimización, las metaheurísticas basadas en una única solución (metaheurísticas basadas en trayectoria o S- metaheurísticas) mejoran la calidad de una única solución. Este tipo de técnicas pueden ser vistas como “caminatas” a través de vecindades o trayectorias a través del espacio de búsqueda del problema en cuestión (Gendreau T. C., 2002). Los caminos (o trayectorias) se generan mediante procesos iterativos que mueven la solución actual hacia otra del espacio de búsqueda. Las S-metaheurísticas han demostrado su efectividad en numerosos problemas de diferentes dominios.

Las S-metaheurísticas aplican procedimientos de generación y reemplazo sobre la solución actual. En la fase de generación se generan un conjunto de soluciones a partir de la solución actual ***s***. Este conjunto C(s) se obtiene normalmente a través de transformaciones locales de la solución. En la fase de reemplazo se realiza una selección del conjunto de soluciones candidatas C(s) para reemplazar la solución actual, es decir, se toma una solución $s' \in C(s)$ como nueva solución actual.

Este proceso se repite hasta alcanzar la condición de terminación. Las fases de generación y reemplazo pueden no utilizar memoria. En este caso, las dos funciones solo utilizarían la solución actual. En caso contrario, puede almacenarse información relativa a las búsquedas y después utilizarla para generar nuevas listas de soluciones candidatas o para seleccionar nuevas soluciones actuales. Algunos ejemplos de esta clase de metaheurísticas son los algoritmos Recocido Simulado (SA por sus siglas en inglés), Búsqueda Tabú (TS siglas en inglés) o Búsqueda de vecinos variables (VNS siglas en inglés).

Al contrario que con los métodos exactos, las metaheurísticas permiten abordar instancias de problemas complicadas obteniéndose soluciones satisfactorias en un tiempo razonable, pero sin garantizar el descubrimiento de soluciones óptimas globales ni extremas. Las metaheurísticas han recibido más y más popularidad en los últimos 20 años. Su uso en diferentes aplicaciones demuestra su eficiencia y efectividad en la resolución de problemas complejos. La optimización está en todas partes y, como además, los problemas de optimización son normalmente complejos, las metaheurísticas también están presentes en todas partes.

A la hora de diseñar metaheurísticas se pueden tener en cuenta dos criterios contradictorios: la exploración del espacio de búsqueda (diversificación) y la explotación de las mejores soluciones encontradas (intensificación). Las regiones prometedoras se determinan a través de las “buenas” soluciones encontradas. En la intensificación, las regiones prometedoras son exploradas más profundamente con la esperanza de encontrar mejores soluciones. En la diversificación, las regiones sin explorar deben visitarse para asegurarnos de abordar todas las posibles regiones del espacio de búsqueda y así, asegurarnos de que la búsqueda no está limitada solamente a una porción del mismo.

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.

Los algoritmos basados en una única solución (por ejemplo, SA y Variable Neigborhood Search (VNS)) manipulan y transforman una única solución durante el proceso.

Estas dos familias poseen características complementarias: las metaheurísticas basadas en una única solución están orientadas a la explotación ya que presentan la capacidad de intensificar las búsquedas en determinadas regiones y las

metaheurísticas basadas en población están orientadas a la exploración al permitir una mayor diversificación por el espacio de búsqueda.

Los algoritmos de ambas familias comparten muchos mecanismos de búsqueda: Iterativos y voraces En los algoritmos iterativos se inicia con una solución completa (o una población de soluciones) y la transforma en cada iteración utilizando varios operadores de búsqueda. Los algoritmos voraces comienzan en una solución vacía y en cada etapa se asigna el valor a una variable de decisión hasta completarla. La mayoría de las metaheurísticas son algoritmos iterativos.

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. Las metaheurísticas constructivas establecen estrategias para seleccionar las componentes con las que se construye una buena solución del problema. 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. Son procedimientos intermedios entre las metaheurísticas de relajación y las constructivas.
- **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. La mejora de una solución se obtiene con base al análisis de soluciones similares o cercanas, denominadas soluciones vecinas. Tales métodos se conocen también como búsquedas monótonas (descendentes o ascendentes) y algoritmos escaladores (hillclimbing). Las metaheurísticas de **búsqueda global** incorporan pautas para escapar de los óptimos locales de baja calidad. Estas pautas consideran tres formas básicas: volver a iniciar la búsqueda desde otra solución de arranque; modificar en forma sistemática la estructura de vecindad que se está aplicando (la manera como se define una solución vecina) y permitir movimientos o transformaciones de la solución de búsqueda que no sean de mejora. Siendo 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. La característica fundamental de estas metaheurísticas es la interacción entre los elementos del conjunto de soluciones que evolucionan, la cual consiste en la combinación entre las soluciones que conforman una población para crear una población nueva que permita evolucionar a la población anterior. Están los algoritmos genéticos, los algoritmos meméticos y los algoritmos de estimación de distribuciones (EDA).

2.3 Metaheurísticas aplicadas al VRP

El VRP es uno de los problemas más estudiados en el área de la programación entera (como ya se mencionó en apartados anteriores). Ha sido abordado por una gran cantidad de investigadores que han aplicado algoritmos exactos (que solo resuelven problemas con pequeñas instancias) y diversas metaheurísticas para problemas de mayor escala. En el apartado 1.1.1, se describieron los principales modelos que han surgido del problema del VRP.

(Eksioglu, Vural, & Reisman, 2009), (Gendreau, Potvin, Braumlaysy, Hasle, & Lokketangen, 2008) y (Braekers, Ramaekers, & Van Nieuwenhuyse, 2016) realizaron una revisión del problema del VRP, hicieron una clasificación basada con 277 artículos revisados. Encontraron que la mayoría de estos artículos solo trabajan con variantes específicas.

En la Tabla 2 se presenta una revisión de los algoritmos aplicados a algunas de las variantes más representativas, no es tan extensiva por no ser parte de esta investigación.

Los artículos fueron tomados de (Braekers, Ramaekers, & Van Nieuwenhuyse, 2016) entre los años 2009 y 2015. Solo consideraron literaruta en inglés y revistas con un factor de impacto de al menos 1.5; las revistas consideradas fueron: Operations Research & Management Science, Transportation or Transportation Science & Technology y Computers & Industrial Engineering (por su importancia).

Tabla 2. Metaheurísticas aplicadas al problema de VRP

Variante	Investigador(es)
dynamic VRP	*(Pillac, Gendreau & Laporte, 2007)*
VRPTW	*(Braysy & Gendreau, 2005ª, 2005b; (Gendreau & Tarantilis, 2014)*
pickup and delivery problems	(Berbeglia, Cordeau, Gribkovskaia, & Laporte, 2007)
vehicle routing with multiple depots	(Montoya-Torres, Lopez France, Nieto Isaza, Felizzola Jimenez, & Herazo-Padilla, 2015
vehicle routing with split deliveries	(Archetti & Speranza, 2012)
VRP periodic	*(Campbell & Wilson, 2014)*
green vehicle routing	(C. Lin, Choy, Ho, Chung, & Lam, 2014)
synchronization aspects in vehicle routing	(Drexl, 2012b)

Como puede observare en la Tabla 2, existen gran variedad de investigaciones hacia el VRP, en esta investigación se va a trabajar con el problema del modelo del CVRP. (Braekers, Ramaekers, & Van Nieuwenhuyse, 2016), reporta en su artículo que éste es más estudiado con 296 modelos propuestos para su solución. Junto con (Laporte G. , 2009). (Gendreau, Laporte, & Potvin, 2002), enumeran 6 principales tipos de metaheurísticas que han sido aplicadas al CVRP:

1. Recocido Simulado (S.A.)
2. Búsqueda Tabú (TS)
3. Algoritmos Genéticos (GA)
4. Colonia de Hormigas (AS)
5. Enjambre de abejas
6. Búsqueda local guiada

Cabe mencionar que existen más metaheurístiacas, resultado de hibridaciones y combinaciones entre las arriba mencionadas y otros mecanismos de apoyo, pero solo se revisaron los artículos con metaheurísticas puras.

En la Tabla 3 se presentan algunos artículos de estas técnicas metaheurísticas utilizadas para dar solución al problema del CVRP y los resultados obtenidos por sus autores.

Tabla 3. Metaheurísticas aplicadas al problema de CVRP

Autor y año	Metaheurística	Trabajo realizado
(Mazzeo & Loiseau, 2004)	**Colonia de Hormigas**	Algoritmo colonia de hormigas (ACO). El algoritmo es competitivo para resolver las instancias de menos de 50 clientes CVRP.
(Park & Imai, 2000)	**Búsqueda Local Guiada**	Se propone un algoritmo de búsqueda local de tipo de cambio vía rápida que utiliza una nueva estrategia de búsqueda, para restringir la búsqueda a los vecinos apropiados.
(Gendreau, Hertz, & Laporte, 1994)	**Búsqueda Tabú**	Se realizaron pruebas en un conjunto de instancias de referencia las cuales indican que la búsqueda tabú supera a las mejores heurísticas existentes y con frecuencia produce las soluciones más conocidas.
(Gomez, Imran, & Salhi, 2013)	**Enjambre de abeja**	Analiza la aplicación meta-heurística de enjambre de abeja por medio de una variación que han denominado Colonia aritifcial de abejas para la solución del problema de CVRP. Aplicaron su algoritmo a las 14 instancias de Christófides.
(Berger & Barkaoui, 2003)	**Algoritmos Genéticos**	El esquema básico consiste en evolucionar al mismo tiempo dos poblaciones de soluciones para minimizar la distancia total recorrida usando los operadores genéticos que combinan las variaciones de los conceptos clave de inspiración a partir de técnicas de enrutamiento y estrategias de búsqueda utilizadas para una variante en el tiempo del problema de proporcionar orientación adicional de búsqueda y así equilibrar la intensificación y diversificación.

Continuación Tabla 3. Metaheurísticas aplicadas al problema de CVRP

(Marinakis, Marinaki, & Dounias, 2010)	**Enjambre de partículas**	Algoritmo de optimización de enjambre de partículas híbridas (HybPSO), en el cual combina el algoritmo de optimización de enjambre de partículas (PSO), el algoritmo de múltiples fases de búsqueda en un vecindario y un procedimiento de búsqueda voraz adaptativa (MPN-GRASP), la estrategia de expansión de búsqueda local (ENS) y un camino volver a vincular la estrategia (PR). Está probado en un conjunto de instancias de referencia y produjo resultados muy satisfactorios.
(Yurtkuran & Emel, 2010)	**Algoritmo electro-magnetismo**	Es un algoritmo basado en la población sobre la base de mecanismos de atracción-repulsión entre partículas cargadas. Los resultados computacionales muestran tiempos de cálculo aceptables cuando se comparan con otras metaheurísticas. Aplicaron su algoritmo a las 14 instancias de Christófides.
(Berger & Barkaoui, 2003)	**Algoritmo Genético Híbrido**	La HGA propuesto tiene tres etapas: Un método de adición más cercano (NAM) incorporado al algoritmo de barrido (SA) que tiene en cuenta al mismo tiempo las relaciones axiales y radio entre los puntos de distribución con el depósito para generar una población inicial cromosomal bien estructurada. Segundo, se empleó la metodología de superficie de respuesta (RSM) para optimizar la probabilidad de cruce y la probabilidad de mutación a través de experimentos sistemáticos. Por último, se incorporó un algoritmo mejorado de barrido en el GA, produciendo un gran revuelo sobre permutaciones de genes en los cromosomas que mejoran la diversidad exploración del GA, evitando con ello la convergencia en una región limitada, y la mejora de la capacidad de búsqueda del GA en acercarse a una solución próxima al óptimo. Por otra parte también se llevó a cabo una estrategia de conservación elitismo celebración de cromosomas superiores para reemplazar los cromosomas inferiores.

Continuación Tabla 3. Metaheurísticas aplicadas al problema de CVRP

(Kirkpatrick , Gelatt, & Vecchi, 1983) (Zeng, Ong, & Ng, 2005) (Osman I. , 1993)	**Recocido Simulado**	Realizaron una prueba con un problema de TSP con 400 clientes. Aplicadado a las instancias de Christhófides. Aplica la metaheurística con interambios entre los nodos de los clientes.
(Shih-Wei, Zne-Jung, Kuo-Ching, & Chou-Yuan, 2009)	**Algoritmo híbrido de recocido simulado y búsqueda tabú**	Toma las ventajas de recocido simulado y búsqueda tabú para resolver CVRP.
(Szeto, Yongzhong, & Ho, 2011)	**Algoritmo de Colonia de abejas artificial para el problema de ruteo vehicular.**	Aplica una heurística de colonia de abejas al modelo de CVRP.

De los trabajos anteriores, se puede concluir que el CVRP es uno de los modelos más estudiados por los investigadores, los autores arriba mencionados han observado que todos los modelos tienen bastantes variantes y metodologías de solución y que solo resuelven un problema específico.

Dentro de las metodologías presentadas en la Tabla 3, las más usadas son los de colonia de hormigas, lista tabú, enjambre de abejas y recientemente algoritmos genéticos. El recocido simulado se ha aplicado de manera híbrida con la lista tabú y como un metaheurística generadora de soluciones iniciales. No se encontró en una revista como un metaheurística pura para la solución del CVRP, solo el trabajo de (Harmanani, Azar, Helal, & Keirouz, 2011) que es un documento de Congreso.

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.

De la revisión de la literatura de estos artículos (y de otros más), permite tomar las consideraciones siguientes:

Del *algoritmo secuencial de recocido simulado*, este algoritmo ha sido aplicado muy poco, generalmente sus aplicaciones son de manera híbrida; en su mayoría con el algoritmo de búsqueda Tabú. Lo anterior porque está considerado como de lenta convergencia, misma que crece de acuerdo al tamaño de la entrada de datos. Su uso como generadora de instancias iniciales es muy apreciada porque los resultados que se obtienen de su ejecución son muy buenos y factibles. De manera “pura”, los resultados obtenidos solo son aceptables, no todas las instancias de prueba utilizadas por los autores logran igualar la mejor cota reportada en la literatura. Su mecanismo de perturbación contempla una o dos heurísticas de búsqueda local.

De esta parte se propone trabajar con una estructura híbrida de vecindad que contenga al menos 3 heurísticas de búsqueda local y una restricción de término de ejecución, esto es que logrado el objetivo de alcanzar un valor menor al 0.10% se termina la ejecución del algoritmo.

Del algoritmo paralelo se observa que predominan 2 formas de paralelizar: Movimientos Paralelos y Búsquedas simultáneas con interacciones periódicas, con ambas reportan soluciones aceptables, el método de interacciones periódicas reporta tiempos más largos de ejecución debido a la comunicación entre nodos que debe realizarse por su forma de trabajo. Además de estas técnicas se tienen otras dos reportadas por Alba, Luque, & Nesmachnow que no se consideraran por su alto costo de latencia.

De esta revisión se trabajará con los movimientos paralelos, pero se considerará otra estrategia con los datos de entrada del problema.

Todo esto será comenado en el capítulo 4 de Metodología de solución.

CAPÍTULO 3 DESCRIPCIÓN DEL PROBLEMA

En este capítulo, se presenta una descripción del problema de ruteo vehicular, se describen las instancias de prueba que validan el algoritmo desarrollado. Se presenta el modelo de optimización, su representación a través de un grafo y los benchmarks utilizados, también se presenta el método de solución que se usó.

3.1 Problema del Transporte Vehicular con Capacidades Homogéneas (CVRP)

El problema de ruteo vehicular con capacidades homogéneas, consiste en obtener un conjunto de rutas lo más cortas posibles utilizando la menor cantidad de vehículos, que partiendo de un depósito y regresando a él, abastezcan a una serie de clientes distribuidos geográficamente, teniendo en cuenta que cada vehículo tiene una capacidad máxima, la demanda de los clientes es diferente y que además no se exceda la capacidad del vehículo asignado a la ruta.

En este capítulo se presenta un panorama general del CVRP, incluye, su definición, su formulación matemática, variantes del problema, instancias del problema y los métodos aplicados que le han dado solución a dicho problema. Una solución, representada de forma gráfica se puede visualizar en la Figura 8.

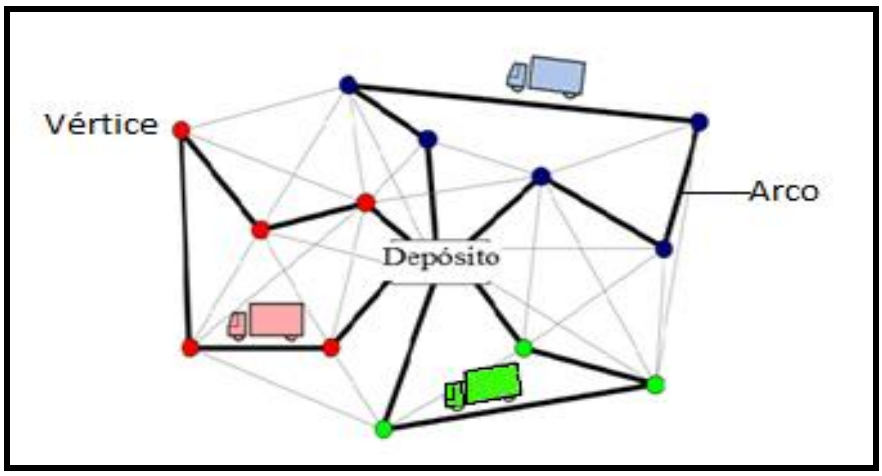


Figura 8. Representación gráfica de la definición del CVRP

3.2 Modelo de optimización

El modelo CVRP se describe como un modelo de programación lineal, entera y binaria; consiste en diseñar un conjunto de rutas donde el objetivo es minimizar el costo total del recorrido de todas las rutas. Como *costo*, puede considerarse el tiempo o la distancia necesaria para realizar el recorrido.

Se considera ruta a un camino que permite transitar de un lugar a otro con un itinerario planeado para realizar un recorrido. Se asigna un vehículo a cada ruta generada.

La asignación de vehículos se realiza tomando en cuenta las características siguientes: (Achuthan, Caccetta, & Hill, 1997) (Laporte G. , 2009) (Hasle & Kloster, 2007)

1. Cada cliente/ciudad es visitada exactamente una vez por un vehículo.
2. Todas las rutas inician y terminan en el depósito.
3. Algunas de estas restricciones son satisfechas:
    3.1 *Restricciones de capacidad*, la demanda no debe exceder la capacidad de los vehículos.
    3.2 No hay una demanda individual que exceda la capacidad del vehículo

3.3 El número de clientes/ciudades a visitar, está acotado por la capacidad de los vehículos.

3.4 *Restricciones de tiempo* total, la longitud de recorrido de cualquier ruta no puede exceder un tiempo definido.

En este modelo las demandas de los clientes son determinísticas ya que se conocen de antemano y no pueden ser divididas, los vehículos son idénticos, tienen una restricción de capacidad y un único almacén; se conocen las distancias que existen entre cada cliente al depósito así como las distancias entre los clientes.

Cuando se ha logrado una asignación de todos los clientes a los vehículos y sus recorridos cumplen con las restricciones del problema, se dice que se ha encontrado una solución. El objetivo central de este problema es minimizar el costo total del recorrido.

En esta investigación se trabajará con las variables, índices, parámetros y modelo siguientes (Restrepo & Medina, 2008):

Variables

$x_{i,j}^{k}$ { = 1 si se asigna el vehículo k para recorrer el arco del nodo i al nodo j.
= 0 (cero) en caso contrario.

$y_{i,j}$ { = 1 si se realiza el recorrido desde i hasta j.
= 0 (cero) en caso contrario

K = número de vehículos a utilizar.

Índices

i = nodo de partida i (1,2,..., n)

j = nodo de llegada j (1,2,..., n)

n = nodos totales

k = vehículo k (1,2,..., K)

Parámetros

c_{ij} = costo de transporte del nodo i al nodo j

d_i = demanda en el nodo j

u = capacidad del recurso k

n = cantidad de clientes

Lo cual genera el modelo siguiente: (González & González, 2006).

Minimizar

$$\sum_{(i,j)\in A} C_{ij}y_{ij} \quad (1)$$

Sujeto a:

$$\sum_{1\le k\le K} x_{ij}^{k} = y_{ij}\,; \quad \forall \quad i,j \quad (2)$$

$$\sum_{1\le j\le n} y_{ij=}1\,; \quad \forall \quad j \quad (3)$$

$$\sum_{1\le i\le n} y_{ij=}1\,; \quad \forall \quad j \quad (4)$$

$$\sum_{1\le j\le n} y_{0j=}k\,; \quad (5)$$

$$\sum_{1\le j\le n} y_{i0=}k\,; \quad (6)$$

$$\sum_{1\le i\le n}\sum_{1\le j\le n} d_i x_{ij}^{k} \le u\,; \quad \forall \quad k \quad (7)$$

$$\sum_{i\in Q}\sum_{j\in Q} y_{ij} \le |Q| - 1 \quad (8)$$

$$\forall \ subconjunto\ Q\ de\ (1,2,\dots n) \quad (9)$$

$$k \le K$$

$$y_{ij} \in \{0,1\};\ \forall\,(i,j)\in A \quad (10)$$

$$x_{ij}^{k} \in \{0,1\};\ \forall\,(i,j)\in A\,, \quad \forall \quad k \quad (11)$$

El conjunto *A* se define como:
$A = \{(i,j): y_{ij=}1\}$.

Las ecuaciones del modelo se describen a continuación:

La ecuación (1) es la función objetivo del modelo del CVRP.
La ecuación (2), hace obligatoria la asignación de un vehículo a la ruta (i,j).
$x_{ij}^{k} = 1$, si se hace el recorrido y $x_{ij}^{k} = 0$ si no se hace el recorrido. Se utiliza el vehículo k en el arco (i,j).
En las ecuaciones (3) y (4), la variable y_{ij}, indica la activación del arco (i,j) y determina un recorrido entre los nodos i,j. Se asegura que todo cliente es un nodo intermedio de alguna ruta.
Las ecuaciones (5) y (6), indican que k es la cantidad de vehículos utilizados en la solución y que todos los que parten del depósito deben regresar al mismo.
Ecuación (7), garantiza que cada vehículo no sobrepase su capacidad.

En la ecuación (8), se vigila que la solución no contenga ciclos usando los nodos 1,2,..n. Q es el conjunto de nodos.
La ecuación (9), limita el número de vehículos a usar.
Ecuaciones (10) y (11) indican que tanto la variable x_{ij}^{k} como la variable y_{ij} son binarias.

Con este modelo se buscará la secuencia en la que todos los puntos deben visitarse a manera de reducir al máximo el tiempo o distancia total del recorrido.

3.3 Modelo de grafos

Un grafo es un conjunto de vértices unidos por aristas (Sedgewick & Wayne, 2011). Es considerado una representación de tipo gráfico, que permite modelar problemas de conectividad. Los grafos son una estructura de datos no lineales que pueden usarse para modelar y poder dar solución a problemas reales como redes de transporte, hidráulicas entre otras. En un grafo se indican trayectorias o caminos.

El problema de Ruteo Vehicular es definido como un grafo en donde sea $G= (V, A)$ donde $V= \{v_1,..., v_n\}$ es un conjunto de vértices representando ciudades o clientes, cada vértice con una demanda asociada q y con un depósito localizado en el vértice v_0 donde tienen su base una flotilla de K vehículos de igual capacidad Q.

$A = \{(v_i, v_j) \mid v_i, v_j \in V \; e \; i \neq j\}$ es un conjunto de arcos, cada arco es una distancia (i, j) donde $i \neq j$, y su valor es positivo.
Este recorrido tiene un costo C asociado al recorrido de i a j, $C= (c_{ij})$, este costo puede ser por tiempo recorrido o por distancia recorrida (Laporte G.) (Duarte & Pantrigo, 2007) (Lenstra & Rinnooy Kan, 1981).

La definición anterior se puede explicar de forma gráfica en la Fig. 9 en donde hay un costo de recorrer el arco conformado por cliente i al j.

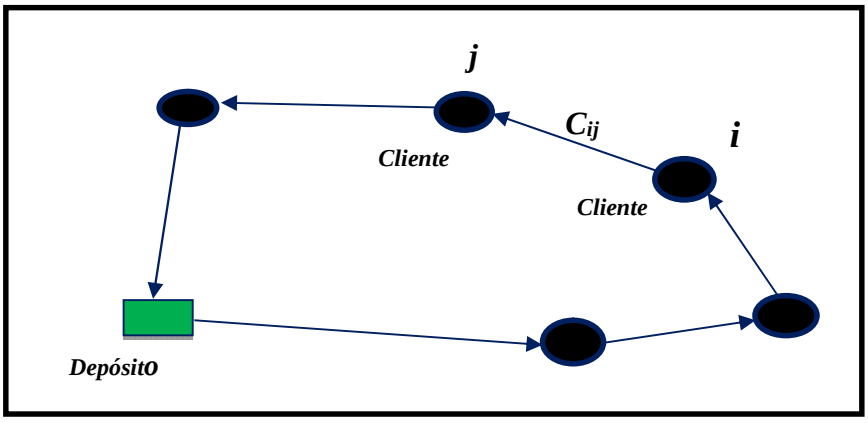


Figura 9. Grafo de una ruta con 5 clientes.

3.4 Benchmarks para el problema de CVRP

Un benchmark consiste en un conjunto de pruebas compuesto por instancias, éstas son un conjunto de datos ya evaluados, cuyo resultado permiten realizar un análisis comparativo de los resultados obtenidos en la ejecución de una heurística o metaheurística (con ciertos parámetros), se le considera un comparador de rendimiento.

En informática, una instancia es un archivo de datos que contiene la información necesaria y validada que permite medir la eficicacia de ejecución de una serie de instrucciones (heurística o metaheurística), se consideran archivos de prueba.

Una instancia generalmente no está dada explícitamente, es una lista que contiene elementos que varían de acuerdo al tipo de problema a resolver, algunas instancias tienen coordenadas, capacidades, demandas de clientes o costos respectivos.

Una instancia de un problema de optimización combinatoria tiende a formalizarse como una pareja (S, f), donde S denota el conjunto finito de todas las soluciones posibles y f es $f:S\rightarrow\mathbb{R}$ la función de costo .

Los elementos de la instancia conforman su tamaño y contribuyen al costo computacional requerido por el algoritmo para generar una solución.

Para el problema del CVRP hay benchmarks desarrollados por diferentes autores: Augerat et al.; Christofides and Eilon; Fisher; Christofides, Mingozzi y Toth; Rinaldi y Yarrow, Taillard; Golden et al, Wasil, Krlly y Chao (Alba E. , 2013).

En esta investigación, se trabajó con las instancias de Augerat, Christofides, Mingozzi y Toth and Taillard por la forma en que están distribuidos los clientes que es de forma aleatoria, por ser los que más se asemejan a distribuciones de la vida cotidiana. Las características de los benchmarks aplicados son los siguientes:

3.4.1 Benchmark de Augerat et Al

Este benchmarks fue propuesto en 1995 y está conformado por tres casos (Set A, B y P), cada uno de ellos están definidos por características diferentes como la distribución y localización de los clientes.

Conjunto A.- Está conformado de instancias en el que ambos, los lugares y las demandas del cliente son generados aleatoriamente por una distribución uniforme. El tamaño de las instancias tiene un rango de 31 a 79 clientes. Este caso fue el que se trabajó para la investigación.

Conjunto B. Las instancias de este conjunto, se caracterizan principalmente porque los clientes se encuentran agrupados.

Conjunto P.- Las instancias de esta clase, son versiones modificadas de otras instancias tomadas de la literatura. (Ajith, Grosan, & Pedrycz, 2010).

En la Figura 10, se muestra la forma en que están diseñados los archivos de las instancias para el CVRP, es una instancia de Augerat et al.

```
NAME: P-n16-k8
COMMENT: (Augerat et al, No of trucks: 8, Best value 435)
TYPE: CVRP
DIMENSION: 16
EDGE_WEIGHT_TYPE: EUC_2D
CAPACITY: 35
NODE_COORD_SECTION
1  30  40
2  37  52
3  49  49
4  52  64
5  31  62
6  52  33
7  42  41
8  52  33
9  57  58
10  62  42
11  42  57
12  27  65
13  43  67
14  55  45
15  55  27
16  37  69
DEMAND_SECTION
1  0
2  19
3  30
4  16
5  22
6  11
7  31
8  15
9  20
10  8
11  8
12  7
13  14
14  6
15  19
16  11
DEPOT_SECTION
1
-1
EOF
```

Figura 10. Estructura de la instancia Pn-16-k8 (16 clientes) por Augerat et al.

3.4.2 Benchmark de Christofides, Mingozzi y Toth

El benchmark está compuesto de 8 instancias (C1-C10) en las cuales los clientes son localizados de forma aleatoria en el plano, más 4 de instancias agrupadas (C11-C14). (Ajith, Grosan, & Pedrycz, 2010)

3.4.3 Benchmark de Taillard

Su benchmark está compuesto de 15 problemas no uniformes. Los clientes son localizados en un plano desplegados en varias grupos y tanto el número de los grupos y su densidad son bastante variables. Las cantidades ordenadas por los clientes son exponencialmente distribuidas, así que la demanda de un cliente puede requerir todo un vehículo. Hay 4 instancias de 75 clientes (9 o 10 vehículos), de 100 clientes (10 u 11 vehículos) y 150 clientes (14 o 15 vehículos). (Ajith, Grosan, & Pedrycz, 2010).

3.5 Recocido Simulado

El recocido simulado es una de las metaheurísticas más utilizadas en optimización combinatoria (Aarts & Lenstra, 2003). Está basado en el "principio de empeoramiento", es decir, se permiten movimientos que empeoran la función objetivo para escapar de óptimos locales.

Se seleccionó esta metaheurística para la resolución del problema del CVRP porque es un procedimiento de búsqueda local apoyado en trayectorias; este tipo de metaheurísticas realizan una búsqueda que parte de un punto de inicio (solución inicial) y van generando un camino o trayectoria (soluciones nuevas) en el espacio de soluciones que explora a través de movimientos (perturbaciones).

Los problemas idóneos para el recocido simulado, por lo general tienen las siguientes características:

(1) Se puede construir una solución inicial, o "estado de energía" (energía de referencia) S_0, donde S_0 es el conjunto de todos las soluciones factibles y se evalúa la función de costo $f(S_0)$.

(2) Se puede construir un mapeo de bajo costo, a través de una relación de vecindad g, a partir de una solución factible del estado s en un conjunto de estados posibles g [{s}].

En este apartado, se explicarán las características del Recocido Simulado que es la metaheurística que se aplica en esta investigación.

La técnica metaheurística del recocido simulado (RS), en inglés Simulated Anneling (SA), está basada en un algoritmo propuesto por Metrópolis et al. (1953) en el marco de la termodinámica estadística, para simular el proceso de enfriamiento de un material. Formulada por Kirkpatrick, Gelatt y Vecchi (1983) e independientemente Cerni (1985) como una forma análoga entre este proceso de recocido y el reto de resolver problemas de optimización combinatoria.

El proceso de recocido es una técnica para modificar el estado de un material y alcanzar un estado óptimo mediante el control de la temperatura. Inicia con el calentamiento del material a una temperatura alta, para luego enfriarlo lentamente, manteniendo en cada etapa una temperatura por cierto tiempo.

Si la disminución de la temperatura es demasiado rápida, pueden originarse defectos en el material y no lograrse el recocido. El disminuir de forma controlada la temperatura conduce a un estado sólido cristalizado, el cual es un estado estable y que corresponde a un mínimo absoluto de energía (Dréo , Pétrowski, & Taillard, 2006).

En este proceso, a medida que baja la temperatura, el sólido va modificando su configuración (distribución de sus elementos). Cada una de las configuraciones tiene asociada una energía determinada, a medida que la temperatura decrece, el conjunto de configuraciones que puede adoptar el sólido se va restringiendo. Con una velocidad de enfriamiento muy rápida se obtienen sólidos no cristalinos, porque

la temperatura desciende muy rápido y las moléculas no pueden reconfigurarse para adoptar configuraciones cada vez más estables de energía menor debido a que las posibles configuraciones se reducen conforme disminuye la temperatura. Lo anterior se ilustra en la Figura 11.

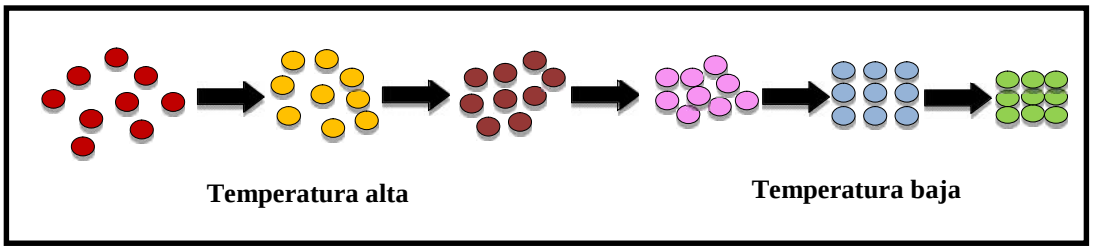


Figura 11. Configuraciones de un sólido aplicando Recocido Simulado.

Metrópolis et al. (1953) modelaron el proceso de recocido, simulando los cambios energéticos en un sistema de partículas conforme decrece la temperatura hasta que converge a un estado estable.

El uso del recocido simulado en optimización combinatoria se basa en establecer analogías entre el sistema físico y el problema de optimización.

En la Tabla 4 se presentan estas analogías.

Tabla 4. Analogía de un sistema físico

Sistema físico (termodinámica)	***Problema de optimización***
Estados del sistema	***Soluciones factibles***
Energía	***Función de costo (función objetivo)***
Cambio de estado	***Solución vecina***
Temperatura	***Parámetro de control T***
Estado estable	***Solución óptima***

El método recocido simulado traslada el proceso de recocido a la solución de un problema de optimización combinatoria, la función objetivo del problema, similar a la

energía del material, es minimizada con la ayuda de una temperatura, la cual es un parámetro de control del algoritmo. Este parámetro debe tener el mismo efecto que la temperatura del sistema físico: conducir hacia el estado óptimo. Si la temperatura es disminuida gradualmente y de manera controlada se puede alcanzar el mínimo global, si es disminuida abruptamente se puede llegar a un mínimo local.

El recocido simulado trata de hacer mínima o máxima una función, que hace de nivel de energía del sólido. A lo largo del proceso van explorando diferentes soluciones, que en términos del proceso de solidificación equivalen a las diferentes configuraciones que puede adoptar el sólido. Igual que en el proceso de solidificación, en el recocido simulado se define una variable temperatura (T).

3.6 Descripción del proceso de Recocido Simulado

El núcleo de recocido simulado es el algoritmo de Metrópolis y la forma de trabajar de la metaheurística es la siguiente:

Algoritmo de Metrópolis

En el algoritmo de Metrópolis se genera una modificación en el sistema (generar un vecino) a partir de un estado de energía actual (solución actual), se genera un nuevo estado (modificación), muy parecido al actual y se calculan los cambios de energía resultantes (función de costo); si esta modificación (vecino), origina una disminución energética, se acepta la modificación, y si ocurre un incremento de energía, el cambio será aceptado con una probabilidad dada.

A continuación se presentan los pasos anteriores.

1) El algoritmo inicia su ejecución con una temperatura (T) cuyo valor sea alto.
2) Parte de una solución factible $\boldsymbol{S}_{\mathbf{actual}}$, que puede ser generada en forma aleatoria, se evalúa la función de costo de esta solución (función objetivo), la cual se identificará como $\boldsymbol{f}\,(\boldsymbol{S}_{\mathbf{actual}})$.

3) Mediante el mecanismo de perturbación, genera una nueva solución ***S*** que corresponde a una solución vecina de S_{actual}, ésta solución vecina es (S_{vecina}).
4) Se calcula el valor asociado de la función de costo *f* (S_{vecina}).
5) Se raliza el cálculo de la diferencia de la función de costo entre ambas soluciones (**ΔE**) con ecuación (12), que muestra la diferencia de energía:

$$\Delta E = f(S_{vecina}) - f(S_{actual}) \quad (12)$$

6) Si **ΔE** < 0, el costo de la nueva solución S_{vecina} es menor al costo de la solución actual S_{actual}, la nueva solución es aceptada, es decir, una solución de menor costo siempre se acepta.
7) En caso contrario, si **ΔE** > 0, se aplica el criterio de aceptación de Boltzmann. La nueva solución es aceptada con una probabilidad P(ΔE), que muestra la ecuación (13), llamada probabilidad de aceptación de Boltzmann:

$$P(\Delta E) = e^{-(\Delta E/t)} \quad (13)$$

A la ecuación (13), se le ha quitado la constante de Boltzman (*k*), ésta no se considera en los problemas de optimización combinatoria por no tener un significado dentro de la formulación del problema (Díaz & Dowsland, 2003).

A una temperatura alta la probabilidad **P(ΔE)** es cercana a 1, por lo que la mayoría de los cambios generados en el sistema son aceptados, esto es, se aceptan soluciones de menor calidad. El aceptar una solución de menor calidad permite salir de un posible mínimo local y explorar otras áreas del espacio de soluciones. Como la simulación comienza con una temperatura alta, **P(ΔE)** es cercana a 1 y por lo tanto, una nueva solución con un costo mayor tiene una alta probabilidad de ser aceptada. Cuando la temperatura baja, **P(ΔE)** es cercana a 0 y la mayoría de los cambios son rechazados (Dréo , Pétrowski, & Taillard, 2006) y (Díaz & Dowsland, 2003).

La probabilidad de aceptar una mala solución (que empeore el costo) va disminuyendo a medida que la temperatura decrece. Para cada nivel de temperatura, el sistema debe alcanzar un equilibrio, es decir, un número de nuevas soluciones debe ser ensayado antes de que la temperatura sea reducida. Se puede mostrar que el algoritmo encontrará, bajo ciertas condiciones el mínimo global y no se estancará en un mínimo local (Moins, 2002).

Este proceso se repite *nrep* veces. Cuando este ciclo iterativo se completa, la temperatura se disminuye y comienza nuevamente el bucle interno (algoritmo de Metrópolis), repitiendo el proceso de creación, evaluación y posible aceptación de soluciones vecinas. Cuando la temperatura es lo suficientemente baja (T_f = *criterio_de_parada*) el algoritmo finaliza y debe indicar cuál fue la mejor solución obtenida.

El pseudocódigo del algoritmo básico de recocido simulado para minimización se presenta en la Figura 12.

El algoritmo para el caso de maximización, se elimina el signo menos en la ecuación (13).

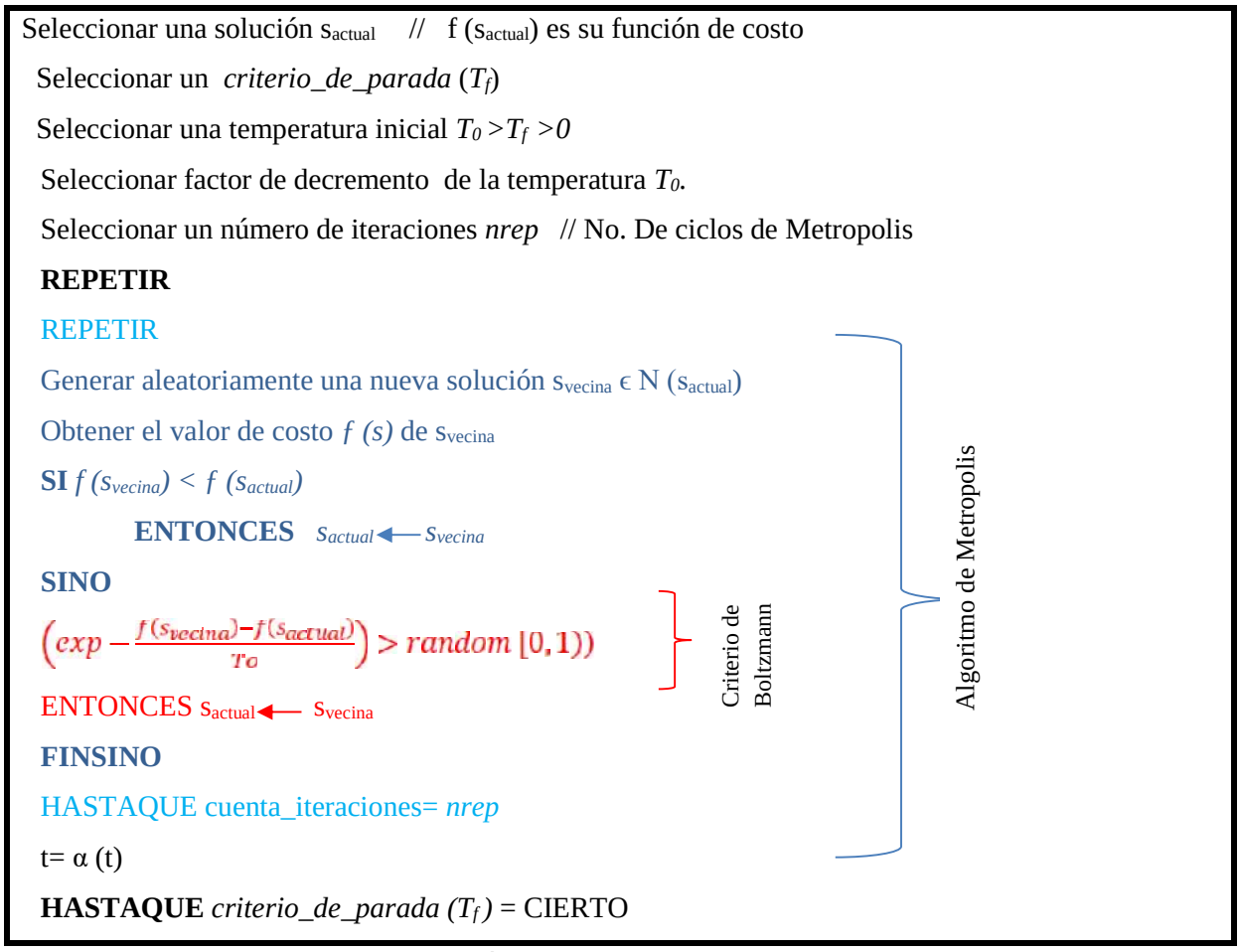


Figura 12. Pseudocódigo del Algoritmo de Recocido Simulado

Para poder implementar el algoritmo de recocido simulado para resolver un problema concreto es necesario tomar ciertas decisiones referentes a la definición de parámetros (temperatura inicial T_0, número de iteraciones en cada nivel de temperatura *nrep*, temperatura mínima T_f), la tasa de decremento (α) de la temperatura, el mecanismo para generar soluciones vecinas, entre otras. Estas decisiones se pueden clasificar en decisiones genéricas y específicas. Una transición consiste en la aplicación del mecanismo de generación de configuraciones vecinas y la aplicación del mecanismo de aceptación.

El algoritmo de recocido simulado se puede ver como varias iteraciones de algoritmos de Metrópolis. Si se baja la temperatura lo suficientemente lento, se puede alcanzar el equilibrio térmico en cada temperatura; esto se logra mediante la generación de varias transiciones en cada temperatura logrando así alcanzar la configuración óptima. Se ha probado que si se realiza este proceso durante muchas transiciones, se puede llegar al equilibrio térmico, caracterizado por la distribución de Boltzmann.

3.6.1 Implementación del algoritmo de RS

Para aplicar el algoritmo se requieren especificar 3 componentes:

- La representación del problema
  - Representar el espacio de solución
  - Expresar la función de costo que represente adecuadamente el costo de soluciones.
- El mecanismo de transición, presentado en la ecuación (14)
  - Generar una nueva solución
  - Calcular diferencia de costo
  - Tomar decisión de aceptación

$$P(aceptación) = \begin{cases} 1 & si\ \Delta E\ \Delta < 0 \\ \exp\left(-\frac{\Delta E}{t}\right) & si\ \Delta E\ \geq 0 \end{cases} \quad (14)$$

- El mecanismo de enfriamiento

En el pseudocódigo de la Figura 12, se pueden diferenciar dos tipos de elementos que permiten afinar el algoritmo para que la ejecución del mismo sea la más eficaz de acuerdo con el problema que se desea resolver (Moreno Díaz, Huecas Fernández-Toribio, Sánchez Allende, & García Manso, 2007):

- **Elementos genéricos.** Son elementos que no tienen una dependencia directa del problema, pero hay que afinarlos para el problema concreto que se desea solucionar.
  - Temperatura inicial: T_0
  - Proceso de enfriamiento: α(t)
  - Número de repeticiones: *nrep*
  - Condición de parada: *criterio_de_parada*

Temperatura inicial. El algoritmo debe comenzar con una temperatura alta que permita que muchos movimientos (o soluciones vecinas) sean aceptados. En la práctica, se requiere conocer el valor de la función de costo para las soluciones de vecinas, por ejemplo, si el mayor incremento en la función objetivo (ΔE) entre soluciones vecinas es conocido, sería posible calcular un valor T_0 que aceptase un movimiento con cierta probabilidad usando la ecuación 14. En la ausencia de tal conocimiento, se puede seleccionar una temperatura que parezca ser un valor alto, ejecutar el algoritmo para un tiempo corto y observar la tasa de aceptación. Si esta tasa es "convenientemente alta", la temperatura con la cual se experimentó puede ser usada como el valor inicial T_0. El significado de "convenientemente alta" varía de una situación a otra, pero en muchos casos una tasa de aceptación ente 40% y 60% parece dar buenos resultados (Reeves, 1996) (Díaz & Dowsland, 2003).

Proceso de enfriamiento. Se refiere a la forma de la curva de enfriamiento que se utiliza en el recocido simulado, la cual determina la velocidad de disminución de la temperatura a medida que avanzan las iteraciones del algoritmo (Gendreau, Laporte, & Potvin, 2002). Existen diferentes maneras de abordar el decrecimiento de la temperatura, una de las más utilizadas debido a su simplicidad y a los buenos

resultados que ha dado en numerosas aplicaciones es la forma exponencial o geométrica, la ecuación de decremento de temperatura geométrica se muestra en la ecuación (15).

$$t_{k+1} = \alpha t_k \tag{15}$$

Donde t_{k+1} es la temperatura en la iteración k+1, t_k es la temperatura en la iteración k y α es una constante cercana a 1, escogida en el rango de 0.85 a 0.99. (Kirkpatrick , Gelatt, & Vecchi, 1983).

Otro método bastante utilizado es el propuesto por Lundy y Mess (1986), la ecuación de decremento de Lundy se muestra en la ecuación (16):

$$t_{k+1} = \frac{t_k}{1+\beta * t_k} \tag{16}$$

Siendo β una constante cuyo valor está cerca de 0. Existen otras formas funcionales en el programa de enfriamiento que se pueden utilizar, sin embargo, no hay en la literatura recomendaciones concisas acerca de cuál es la mejor, esto depende del problema y en ese caso debe decidirse por experimentación.

Número de repeticiones (iteraciones). El recocido simulado ejecuta cierto número de iteraciones en cada nivel de temperatura para alcanzar el equilibrio, una vez alcanzado este estado la temperatura se reduce y el proceso se repite. Un esquema obvio es mantener un número de iteraciones constante en cada temperatura o alternativamente, se puede variar según desciende la temperatura, dedicando suficiente tiempo de búsqueda a temperaturas bajas para garantizar que se visita el óptimo local, es decir que conforme disminuye la temperatura se aumenta el número de repeticiones (Díaz & Dowsland, 2003).

Temperatura final. Teóricamente la temperatura debería reducirse hasta 0, pero en la práctica la búsqueda converge por lo general a su óptimo local final antes de llegar a este valor temperatura. Se corre el riesgo, de que el algoritmo puede hacerse demasiado largo y los tiempos computacionales pueden ser muy grandes, especialmente si se utiliza una tasa de decrecimiento geométrica. Hasta cierto grado la temperatura final puede depender del problema específico, y como en el caso de

la temperatura inicial, se puede hacer una estimación a partir de establecer la probabilidad de aceptación que se desea en la etapa final del algoritmo. Un criterio diferente es detener la búsqueda cuando se haya producido un número determinado de iteraciones sin alguna aceptación. Así por ejemplo, Dréo et al. (2006) sugieren que se debe terminar el algoritmo después de tres etapas sucesivas de temperatura sin que haya habido alguna aceptación.

- **Elementos dependientes de problema**. Son los elementos que definen de forma directa el problema que se está resolviendo. Definen un modelo del problema y la estructura del espacio de soluciones para el mismo.
  - Espacio de soluciones: S
  - Estructura de vecindad: S'
  - Función de costo: f(s)
  - Solución inicial: s_0

Espacio de soluciones / búsqueda. Cuando se presenta un problema concreto, existe una serie de algoritmos que se pueden aplicar para resolverlo. Al hacerlo, se busca la mejor solución entre un conjunto de posibles soluciones.

Al conjunto de todas las posibles soluciones que pueden ser consideradas durante la ejecución del algoritmo para un problema concreto se llama espacio de búsqueda. El buscar una solución, se reduce a buscar un valor extremo (mínimo o máximo) en el espacio de búsqueda. A veces, este espacio de búsqueda puede ser bien definido, pero en la mayoría de las ocasiones sólo se conocen algunos puntos.

Existe un costo asociado a cada solución, de tal forma que el espacio de soluciones está caracterizado por un "paisaje de costo". La dificultad de un problema de optimización radica en que este paisaje comprende un gran número de valles de profundidad variable y que corresponden a mínimos locales y de crestas de altura también variable y que se consideran óptimos locales (ver Figura 13). La forma de este paisaje depende bastante de la función de costo y de la estructura de vecindad.

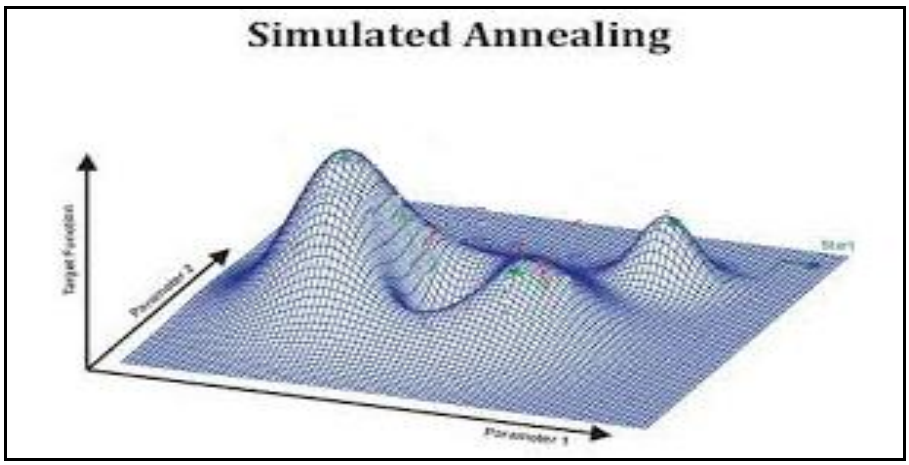


Figura 13. Representación de un espacio de soluciones aplicando el algoritmo de Recocido Simulado.

Estructura de vecindad. En cada iteración, las configuraciones generadas de una solución inicial (s_0), definen un conjunto de soluciones vecinas de la solución s_0, a esto se le llama vecindario $N(s_0)$ de s_0 Una decisión importante en un problema de recocido simulado es la definición de la estructura de vecindad, o lo que es igual, establecer cómo generar una solución vecina. Algunos de los primeros desarrollos teóricos se basaban en vecindades, es decir, se pasa de una solución i a una solución j. Los resultados teóricos demuestran que es suficiente con exigir que cualquier solución pueda alcanzarse desde cualquier otra a través de una serie de movimientos válidos denominados perturbaciones o movimientos (Reeves, 1996) y (Díaz & Dowsland, 2003).

Función de costo. Esta función debe medir la calidad de una solución y se define de tal manera que represente el problema que se está tratando de resolver. Un aspecto que es importante tomar en cuenta relacionado con la función de costo es el cálculo de su valor para una solución determinada. En cada iteración del algoritmo cuando se halla una nueva solución es necesario calcular el cambio de la función de costo con respecto a la solución anterior. Es posible disminuir el tiempo de computación si este cambio se calcula usando información relativa a qué parte de la solución ha cambiado, sin proceder a evaluarla totalmente sin tener en cuenta la información previamente disponible (Díaz & Dowsland, 2003).

Solución inicial s_0. Es una instancia de prueba (datos de entrada de un problema) que puede ser factible o no, esta factibilidad depende del tipo de problema a resolver porque algunas metaheurísticas cuentan con mecanismos que las pueden hacer factibles. Se considera una solución factible porque cumple las restricciones requeridas por el problema a tratar La solución inicial aplicada al algoritmo propuesto, se presentará de manera amplia en el apartado 3.3.

CAPÍTULO 4 METODOLOGÍA DE SOLUCIÓN

En este capítulo se presentan la metodología experimental aplicada, los formatos que permiten la introducción de los datos e instancias para la resolución del problema del CVRP, el algoritmo de recocido simulado de forma secuencial y distribuido y el cálculo de la complejidad del algoritmo. Se detallando las métricas empleadas para evaluar la calidad de los resultados obtenidos en el algoritmo distribuido.

4.1 Representación simbólica de la solución inicial

La representación simbólica es un valor o referente que recibe la computadora por diferentes medios. Los datos representan la información que se manipula en la construcción de una solución o en el desarrollo de un algoritmo.

La instancia usada para el CVRP en esta tesis, se modificó para ser usada por el algoritmo y está conformada por los datos siguientes en un archivo *.txt:

1) La capacidad del vehículo
2) Las coordenadas (x, y) ubicadas en las dos primeras columnas, la primeras coordenadas corresponden a la ubicación del depósito, por eso su valor de demanda de cero.
3) La demanda de cada cliente indicada en una tercera columna.

Lo anterior se representa en la Figura 14.

200 →		**Capacidad de vehículo**
Coordenadas del depósito		
35	**35**	**0**
41	**49**	**10**
35	**17**	**7**
55	**45**	**13**
55	**20**	**19**
15	**30**	**26**
25	**30**	**3**
20	**50**	**5**
10	**43**	**9**
55	**60**	**16**
30	**60**	**16**
20	**65**	**12**
50	**35**	**19**
30	**25**	**23**
15	**10**	**20**
30	**5**	**8**
10	**20**	**19**
5	**30**	**2**
Coordenada X	**Coordenada Y**	**Demanda Cliente**

Figura 14. Representación de la instancia de prueba para el CVRP.

Para el funcionamiento del algoritmo se usaron estructuras de datos del tipo *struct*, como una representación simbólica del problema.

El algoritmo tiene dos estructuras:

La primera estructura, guarda los datos de la instancia de prueba: las coordenadas y la demanda, la Figura 15 muestra a la estructura.

```
typedef struct
{
   int C;
   int coordX[LIMITE_MAX+2];
   int coordY[LIMITE_MAX+2];
   int dema[LIMITE_MAX+2];
}instancia;
```

Figura 15. Tipo de dato “struct” para el almacenamiento de datos de la instancia

Los campos que utiliza son los siguientes:

a) La variable ***C***, que almacena la capacidad de los vehículos.
b) Arreglo ***coordX*** almacena la posición de la coordenada X del cliente. La longitud del arreglo está determinada por una constante global.
c) El arreglo ***coordY*** almacena la posición de la coordenada Y del cliente.
d) Arreglo ***dema*** guarda la demanda de cada cliente.

La segunda estructura usada guarda la solución inicial con la que trabaja el algoritmo de optimización. Esta estructura se presenta en la Figura 16.

```
typedef struct
{
   int vh;
   int numClientes;
   int ruta[LIMITE_MAX+2];
   int limiteSuperior[LM];
   int limiteInferior[LM];
   double costoVh[LM];
   double costoSolucion
}solucion;
```

Figura 16. Estructura que almacena la solución inicial del algoritmo.

Los campos que utiliza son los siguientes:

a) La variable ***vh***, almacena el número de vehículos o rutas creadas.
b) La variable ***numClientes*** almacena la cantidad de clientes de la instancia.
c) El arreglo ***ruta*** almacena los clientes que conforman una ruta.
d) Arreglo ***limiteSupeior*** guarda las posiciones de los segmentos superiores de la ruta.
e) Arreglo ***limiteInfeior*** guarda las posiciones de los segmentos inferiores de la ruta.
f) Arreglo ***costoVh*** guarda el costo de los recorridos de cada ruta.

g) La variable ***costoSolucion*** almacena la suma del costo total de la solución de los recorridos de todas las rutas.

4.2 Generación de la solución inicial

La solución inicial con la que se trabaja es generada por una función dentro del algoritmo de recocido simulado el cual es tomado y adaptado del algoritmo UKCVRP (D'Granda-Trejo, 2013) . El algoritmo de agrupamiento es parte de una estrategia de dos fases "agrupar primero, rutear después". La primera fase, en donde se genera una solución inicial, sirve como parámetro de entrada para la segunda fase de rutear (algoritmo de recocido simulado).

Este algoritmo cumple con las restricciones indicadas por el modelo del CVRP (sección 3.1 y 3.2) y en la fase de agrupamiento da una solución con el mínimo de rutas posibles pero sin optimizar. El algoritmo CVRPUK trabaja con instancias desde 5 hasta 1000 clientes.

Una representación gráfica de una solución inicial generada por el algoritmo de agrupamiento se muestra en la Figura 17. En esta figura se representa la instancia CH3 de Cristhófides cn un costo inicial de 1971 unidades de distancia con 8 rutas.

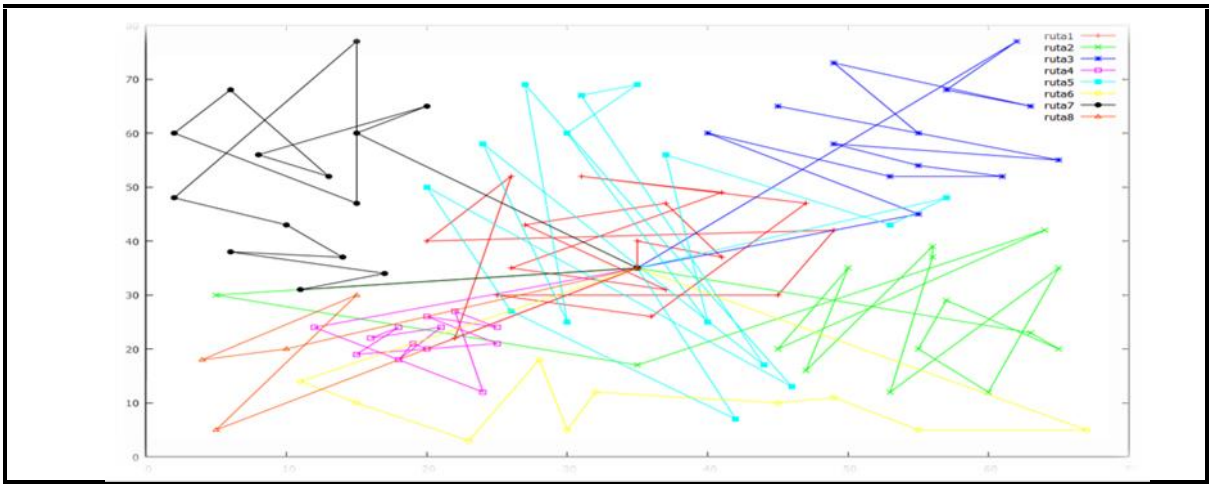


Figura 17. Representación de una solución inicial de la instancia CH3.

La representación de esta solución inicial factible es a través de un archivo *.txt con las características siguientes:

a) Primer número es el número de rutas óptimo generado por el algoritmo UKCVRP,
b) Segundo renglón indica el vector solución con los clientes de la instancia agrupados en cada ruta, éstos se encuentran delimitados por límites,
c) El renglón 3 marca los límites inferiores de cada ruta y el renglón 4 indica los límites superiores.

Lo anterior se muestra en la Figura 18:

8 Número de rutas óptimo

27 28 69 52 53 89 1 31 50 6 26 76 18 88 92 55 25 54 4 39 24 56 80 68 72 12 21 Vector de clientes asignados a cada ruta

0 16 31 44 57 72 82 96 0 Vector de Límites Inferiores

15 30 43 56 71 81 95 99 0 Vector de Límites Superiores

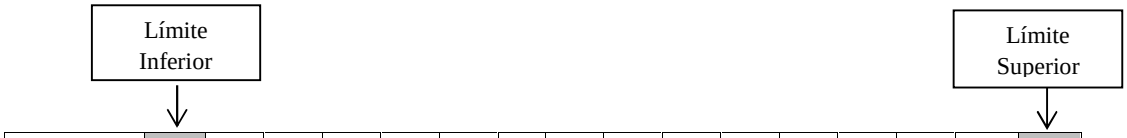
Figura 18. Representación de una solución inicial.

4.2.1 Representación del vector de rutas

Una ruta de la Figura 18 se representa en la Tabla 5. La ruta mostrada es la que tiene los límites de 0 a 15. Su explicación es la siguiente:

1. La primera fila es la posición de cada cliente.
2. La segunda, muestra al vector de clientes de la solución.
3. Los cuadros sombreados representan las posiciones de los límites inferior y superior de la ruta.

Tabla 5. Representación de una ruta de la soluión inicial de la Figura 18.



Posición	**0**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	**15**
#cliente	27	28	69	52	53	89	1	31	50	58	6	26	76	18	88	92

4.2.2 Representación de las soluciones obtenidas

Una vez aplicado el algoritmo de RS, las soluciones obtenidas se representan de la forma siguiente en un archivo *.txt.

Los campos que se ven son:

1) El número de rutas generado.
2) El vector en donde están ubicados todos los clientes de la instancia por rutas.
3) Un vector en donde se indican los límites inferiores, en donde se inicia de la posición 0.
4) Un vector donde se indican los límites superiores, finalizando con la última posición del cliente de la última ruta.
5) Adicionalmente se puede agregar la carga de cada vehículo asignada a cada ruta, la cual no debe exceder de la capacidad del vehículo.

Lo anterior se representa mejor en la Figura 19.

8 **Número de rutas generadas**

1) 13 2 57 15 41 22 75 74 72 73 21 40 58 53
2) 28 76 68 80 24 29 79 33 81 9 51 1 69
3) 50 77 3 78 34 35 71 65 66 20 30 70 27
4) 97 87 42 43 14 44 38 86 16 61 96
5) 89 18 7 82 48 19 63 90 32 10 31 52
6) 26 4 56 23 67 39 25 55 54 12
7) 94 95 92 98 37 100 91 85 93 59 99 6
8) 88 62 11 64 49 36 47 46 8 45 17 84 5 83 60

Clientes asignados a cada ruta

0 14 27 40 51 63 73 85 **Límites Inferiores**
13 26 39 50 62 72 84 99 **Límites Superiores**

Figura 19. Representación de una solución a la instancia CH3.

Su representación gráfica es la siguiente (Figura 20):

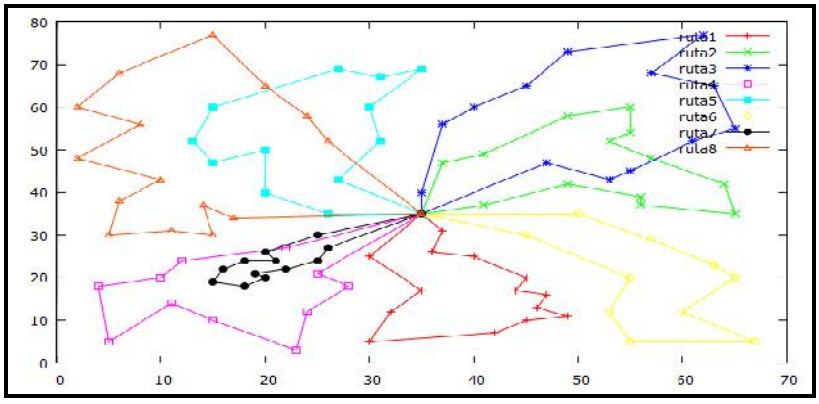


Figura 20. Representación gráfica de una solución de la instancia CH3.

4.3 Estructura híbrida de vecindad

Las heurísticas de mejora para el VRP están dirigidas a optimizar una solución factible inicial a través de un mecanismo de búsqueda.

Las heurísticas de mejora pueden clasificarse siguiendo dos criterios: por el tipo y la estrategia de búsqueda.

El tipo de heurística se refiere al número de rutas involucradas en la mejora: procedimientos intra-ruta si sólo una ruta está implicada y un procedimiento inter-rutas si dos rutas son involucradas. Las intra-rutas son procedimientos de mejora que se enfocan a mejorar la secuencia de una ruta cambiando el orden de los elementos de una misma ruta. Los intercambios inter-rutas mejoran la solución inicial moviendo sus elementos entre un par de rutas.

La estrategia de búsqueda es un procedimiento el cual indica el orden en el cual nuevas soluciones son buscadas, se conocen dos categorías: las estrategias de búsqueda local y global.

La estrategia local, es un procedimiento tradicional "descendente" en el cual se busca un mínimo local realizando movimientos de los elementos que mejoren el valor de la función objetivo. Lo anterior se logra con varias iteraciones, aunque generalmente se obtendrá un óptimo local, por lo que a esta forma de mejoramiento se conoce como heurística de búsqueda local.

La heurística es detenida si ya no encuentra una mejora en la función objetivo, este es el principal inconveniente de estos movimientos: que se pueden quedar "atrapados" en un óptimo local.

Como mejoramiento de las rutas se presentan heurísticas que buscan el paso de una solución viable a otra que genere menor costo en la función objetivo. (Osman I. , 1993) Sugiere en su artículo que es mejor iniciar el algoritmo con una solución inicial factible cuyo costo no sea muy alto.

Los diferentes tipos de movimientos que se presentan en este apartado, son considerados para la heurística de mejora, permiten generar un "vecindario de soluciones" de una solución existente. Un movimiento puede ser definido como un intercambio entre los clientes, ya sea de la misma ruta o diferentes (Laporte & Semet, 2001).

La definición de un vecindario es un paso requerido para el diseño de cualquier metaheurística ya que juega un papel crucial en el desarrollo de ésta. Si la estructura del vecindario no es la adecuada para el problema, la metaheurística podría tener dificultades para resolver el problema.

Una solución s' en el vecindario de s_0 $(s' \in N(S))$ es llamado "*vecino*" de s_0. Un vecino es generado por la aplicación de un movimiento por un operador que desarrolla una pequeña perturbación a la solución inicial s_0.

Una vecindad o vecindario se define como un conjunto de soluciones vecinas $\mathbf{N}(s_0)$, derivadas de una solución inicial s_0. La estructura de vecindad generada se usa para realizar una exploración del espacio de soluciones en el vecindario creado (Papadimitriou & Steiglitz, 1998).

El vecindario de $\mathbf{N}(s_0)$ de una configuración s es definida por la ecuación (17):

$$N(S_0) = \{s' | s' \in S, gs_0s'(T_n) > 0\} \quad (17)$$

en la cual $s_0 \not\in N(s_0)$ y $s_0 \in N(s')$ si $s' N(s_0)$

De donde

$N(s_0)$, es el vecindario generado.

s_0, es la solución inicial.

s', es la solución vecina.

$gs_0s'(T_n)$, es la probabilidad de generar una configuración s' (solución vecina) de una configuración s_0 (configuración inicial) a temperatura T_n (Benoit & Jin-Kao, 2007).

La ecuación (17) se explica a continuación:

El vecindario ($N(s_0)$), está conformado por todas las soluciones generadas (soluciones vecinas (s')) que forman parte del conjunto solución **S**; éstas soluciones vecinas, son generadas de una configuración inicial (s_0) con cierta probabilidad a una temperatura dada T_n.

En las Figura 21 se muestran las representaciones de un vecindario generado a partir de la solución inicial (s_0), los puntos en colores de la segunda figura representan soluciones vecinas (s') que pertenecen al vecindario de la solución inicial $(N(s_0))$.

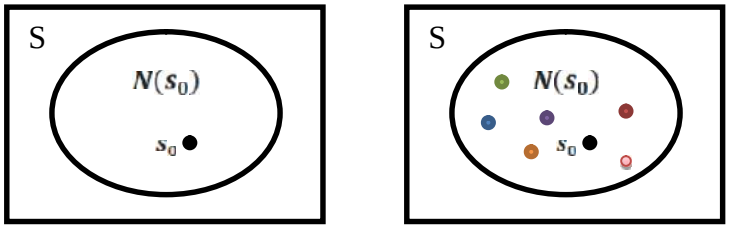


Figura 21. Estructura de Vecindad

Una de las aportaciones de esta investigación es la de integrar una estructura de vecindad para realizar los movimientos de perturbación para encontrar soluciones de buena calidad, también se presenta una heurística de búsqueda local dirigida llamada cambioVecino, la cual se explicará más adelante. La estructura de vecindad está integrada por los movimientos que se presenta en los apartados siguientes.

4.3.1 Heurísticas de mejora de rutas

Una heurística de mejora (movimiento de perturbación) puede ser definido como la "mutación" (intercambio) entre los clientes, ya sean de la misma ruta o de rutas diferentes (Laporte & Semet, 2002). Una solución vecina (s') es generada por la aplicación de un operador de movimiento, el cual realiza una pequeña perturbación o movimiento en la solución inicial (s_0).

Los tres movimientos aplicados en el algoritmo de trabajo de esta tesis doctoral son:

1) “Movimiento 1-swap” basada en intercambiar dos nodos de la misma ruta (Figura 22). (Subraminan, Uchoa, & Satoru, 2013) (Habibeh & Lai, 2012) (Braysy & Gendreau, 2001).

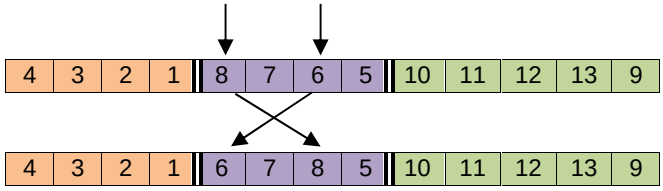


Figura 22. Movimiento 1-swap en la misma ruta.

(Christofides & Beasley, 1984) Mencionan intercambios entre dos rutas, donde los nodos de diferentes rutas son intercambiados. A este movimiento se le conoce como Movimento *l-Exchange*. Ver Figura 23.

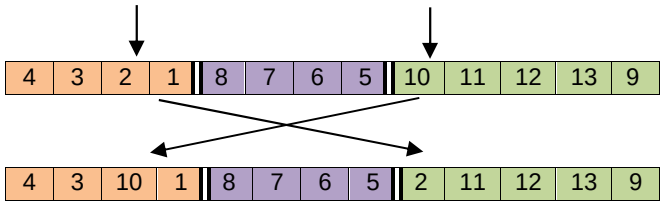


Figura 23. Movimiento l-Exchange.

2) Movimiento 1-Relocate (también conocido como inserción), se genera el salto de un nodo a otra posición dentro de su misma ruta (Ling, Ying, Lee, & Lee, 2009) (Laporte & Semet, 2001) (Mole & Jameson, 1976) (Figura 24).

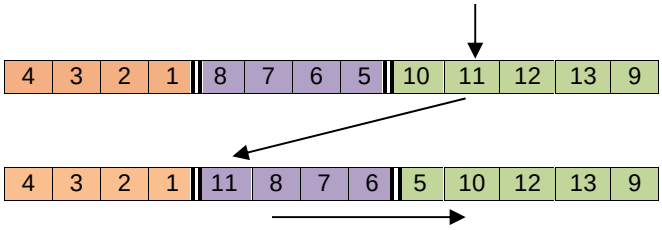


Figura 24. Movimiento 1-relocate.

Movimiento cambiaVecino, todos los movimientos presentados anteriormente generan un vecindario de soluciones de manera aleatoria. Esta heurística, trata de hacer la búsqueda local de una manera más dirigida, reduce la aleatoriedad mejorando la solución obtenida de acuerdo a las pruebas experimentales realizadas.

3) La perturbación cambiaVecino trabaja de la forma siguiente:
   a) Recibe una solución inicial (vector solución).
   b) Escoge un cliente (*num1*) de manera aleatoria de alguna ruta del vector solución
   c) El algoritmo revisa cuál es su vecino más cercano en la matriz de distancias.
   d) Una vez ya localizado el vecino más cercano (*num2*), hace el intercambio escogiendo de forma aleatoria uno de sus vecinos adyacentes ((*num1-1*) o (*num1+1*)) colocando en su lugar el vecino cercano elegido (*num2*) (Figura 25).

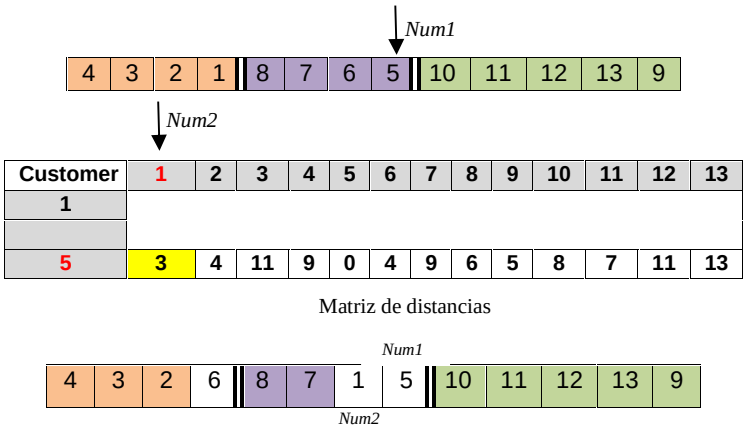


Figura 25. Movimiento cambiaVecino.

4.3.2 Generación de vecinos

La heurística propuesta para la generación de vecinos está basada en los trabajos de (Martínez-Bahena, y otros, 2012) y (Juárez-Chávez, y otros, 2012); está conformada

por una estructura híbrida de vecindad cuyas heurísticas o movimientos anteriores, permiten en cada iteración una configuración diferente que permite hacer la exploración del espacio de soluciones al ir cambiando la configuración como puede observarse en las figuras siguientes (Figura 26 y 27).

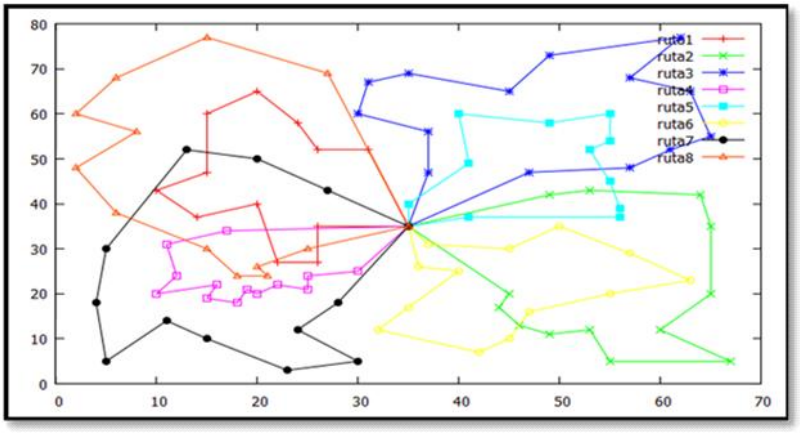


Figura 26. Configuración de la solución de la instancia CH3 con un costo de 933.

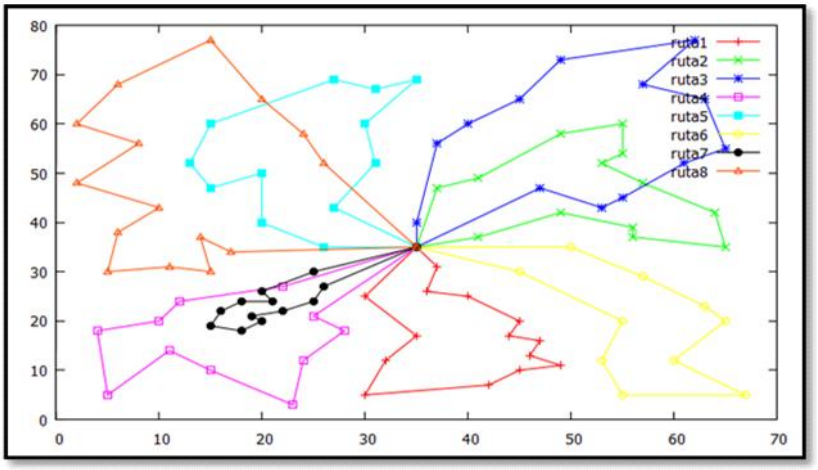


Figura 27. Configuración de la instancia CH3 con un costo de 852.

En la Figura 28 se presenta el pseudocódigo del algoritmo de recocido simulado con reinicio que se desarrolló para esta investigación. Se le ha llamado con reinicio porque cada vez que termine un ciclo de ejecución, el mejor valor obtenido es el que ingresa como nueva solución inicial hasta que termine su número de repeticiones.

```
1.  Recibe solución_actual, tempInicial, tempFin, alfa, LongM
2.  s_actual=calculaCostoVh(s_actual);
3.  s_mejorRS= s_actual ;
4.  Para k=0; k<N; k++  hacer // se establece el número de iteraciones que se ejecuta con reinicio
5.      s_actual = s_mejorRS;
6.    Para  tempIni = tempInicial ; tempInicial >= tempFin  ;  tempInicial *= alfa  hacer
            Para i = 0; i <= LongM; i++ hacer
                cont++;
                s_vecino = s_actual;
                opción = 1 + (rand ( ) %4);
                según opción hacer
                        caso 1: unoSwapB (vecino );
                        caso 2: insercionB (vecino );
                        caso 3: insercionB ( vecino );
                        caso 4: cambiaVecino ( vecino );
                FinSegun
                s_vecino = calculaCostoVh (s_vecino);
                Si s_vecino < s_actual entonces
                        s_actual = s_vecino;
                        Si s_vecino < s_mejorRS entonces
                                s_mejorRS = s_vecino;
                FinSI
                Sino hacer
                        probAcepta = exp ((-1) * ((doublé) (s_vecino – s_actual) / tempInicial));
                        u = ((doublé) rand ( ) / RAND_MAX);
                        Si u < probAcepta entonces
                                s_actual = s_vecino;
                        FinSi
                FinSi
            FinPara
      FinPara
7.  FinPara // si cumple el número de iteraciones ir a 8
8.  retornar s_mejorRS; //es la mejor solución obtenida en varios reinicios
```

Figura 28. Algoritmo Recocido Simulado con reinicio.

4.4 Complejidad del algoritmo secuencial

En este apartado se muestra el análisis de la complejidad del algoritmo secuencial, la fundamentación teórica se mostró en el apartado 2.2.

El primer problema que se tiene al estimar el tiempo de ejecución de un algoritmo es la duración, que es el tiempo que tarda en ejecutarse un programa el cuál varía de acuerdo a la entrada que se le ingrese. En ocasiones, para entradas del mismo tamaño, el algoritmo se ejecuta en tiempos distintos. No obstante, es común que entre más crezca la entrada, más se tarde la ejecución. Es por esto que el tiempo de ejecución se define en función del tamaño de la entrada.

Por esto, el tiempo de ejecución de un algoritmo se define por el número de instrucciones u operaciones ejecutadas. Una línea de código puede correr en diferente tiempo que otra (realizar una multiplicación normalmente tarda más que una suma), pero en general se supone que la ejecución de cualquier instrucción básica tarda lo mismo

Las cotas de complejidad ayudan a clasificar los algoritmos de acuerdo a su tiempo de ejecución. Si se tiene un algoritmo que para una entrada *n* tarda *f*(*n*) en ejecutarse, lo que nos interesa es encontrar una cota que crezca de manera similar a *f*(*n*).

La complejidad asintótica del algoritmo de recocido simulado con reinicio es presentada en la ecuación (18).

$$F(n) = n(69 + 11n + 8w) + 9v + 100 \quad (18)$$

El cálculo de su complejidad se muestra en el Anexo 1.

4.5 Paralelización de algoritmos

En los últimos años proliferan cada vez más problemas que requieren un alto grado de cómputo para su resolución. La existencia de estos problemas complejos computacionalmente, exige la fabricación de computadoras potentes para solucionarlos en un tiempo adecuado. Las computadoras secuenciales utilizan técnicas como segmentación o computación vectorial para incrementar su productividad; también se aprovechan de los avances tecnológicos, especialmente en la integración de circuitos, que les permite acelerar su velocidad de cómputo. Actualmente se han alcanzado límites físicos en el crecimiento de los beneficios, de modo que sobrepasarlos requerirá costes económicos no tolerables por los fabricantes y mucho menos por los usuarios. Es por ello que la programación paralela ha pasado a ser la forma más barata de trabajar estos problemas.

La paralelización de algoritmos es un procedimiento que aborda problemas de gran tamaño y complejidad en donde encontrar una solución, es una tarea muy costosa. Un programa es paralelo si en cualquier momento de su ejecución se puede ejecutar más de un proceso.

4.5.1 Técnicas de paralelización

El paralelismo es la ejecución de varias actividades al mismo tiempo y que tienen una interrelación. El cómputo paralelo es la ejecución de más de un cálculo al mismo tiempo usando más de un procesador (García Regis & Cruz Martínez, 2003).

La técnica para la programación en paralelo, permite fragmentar un problema en subproblemas a resolver, su objetivo es dividir en forma equitativa los cálculos asociados al problema y los datos sobre los cuales opera el algoritmo y así reducir al mínimo el tiempo total de cómputo al distribuir la carga de trabajo entre varios procesadores. Estas técnicas son:

1. **Descomposición** o **Particionamiento** que a su vez se divide en:
    a) ***Descomposición de dominio*****,** se intenta dividir los datos en partes pequeñas, de aproximadamente el mismo tamaño para realizarles cálculos asociando las operaciones a datos específicos. Se asocia con la estrategia “divide y vencerás” y los modelos SIMD (Single Instruction Multi Data, memoria compartida) y *SPMD (Single Program Multi Data, memoria distribuida)*. Un mismo programa puede ejecutarse sobre diferentes conjuntos de datos, pueden no realizar las mismas instrucciones. Es el modelo de la biblioteca MPI (Message Passing Interface).
    b) ***Descomposición funcional*****,** se concentra en el particionamiento de las operaciones del problema, replica los datos entre los procesos asociados a las diferentes tareas (realizan diferentes operaciones).

Cuando se comparte la carga de trabajo entre N procesadores, se espera que trabaje N veces más rápido que con un solo procesador pero la situación no es tan sencilla, existen varios factores de sobrecarga que pueden disminuir el rendimiento. Algunos de estos factores son: las estructuras pocos regulares de los problemas, los mismos algoritmos o las técnicas utilizadas.

Sin importar que los parámetros para la ejecución del recocido simulado sean seleccionados de forma adecuada, éste converge a menudo muy lentamente. Una implementación en un sistema distribuido podría reducir el tiempo de cómputo dependiendo del número de procesadores usados en la paralelización para obtener un speedup mayor manteniendo la calidad del algoritmo secuencial. Un punto importante es que no existen teoremas que prueben la convergencia del recocido simulado distribuido como los hay para recocido simulado secuencial. (Rochat & Taillard, 1995), han escrito que este método es intrínsecamente secuencial y su paralelización no resulta fácil.

Entre las propuestas realizadas por diversos autores como estrategias de paralelización presentan las siguientes:

2. ***Búsquedas simultáneas independientes (paralelización independiente)***, se disponen de ***N*** procesadores idénticos, cada uno con suficiente memoria y capacidad computacional, los cuales inician con un espacio de soluciones *S*, una función de costo *f(s)* y un esquema de enfriamiento.

Cada procesador tiene asignada una configuración de solución inicial y ejecutan el algoritmo de recocido simulado, en cada uno se obtendrán ***k*** soluciones independientes S_1^N, $S_2^N \ldots S_k^N$, entre las cuales se selecciona la mejor de todas como solución final, la probabilidad de encontrar la solución óptima aumenta con el número de procesadores. Variantes de esta estrategia sería:

a) Modificar todos parámetros de control para cada procesador o solamente modificar uno solo de los parámetros: la temperatura inicial, la cadena de Markov o la función de decremento. De esta forma utilizando distintos valores para cada procesador se puede conseguir que cada uno de ellos explore un espacio de soluciones de distintas dimensiones.

b) El maestro duplica la solución actual entre los procesadores. Cada uno desarrolla por separado su solución y los resultados se devuelven al maestro.

El esquema correspondiente a una paralelización de este tipo se muestra en la Figura 29. El modelo de trabajo aplicado para paralelizar en esta investigación fue la de “maestro-esclavo”; cada nodo desarrolla el mismo proceso pero con ruteos diferentes de la misma instancia.

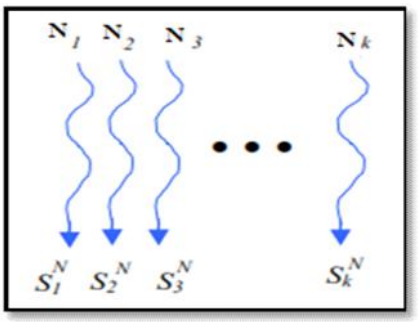


Figura 29. Implementación paralela de RS, búsquedas simultáneas independientes.

3. ***Búsquedas simultáneas con interacciones periódicas***. Esta estrategia permite comunicaciones en puntos intermedios, por lo general se usan cadenas de Markov diferentes a las iniciales en cierto momento de la ejecución. Otra variante, es el paso de la mejor solución encontrada en cierto tiempo de ejecución (*Nk* soluciones encontradas) del algoritmo por cada procesador y pasarla entre procesadores contiguos, si la solución recibida es mejor de la que disponía en ese momento seguirá la búsqueda, en caso contrario seguirá con la que tiene. Se obtendrán ***k*** soluciones de cada procesador y se selecciona la mejor. (Ram, Sreenivas, & Subramaniam, 1996) usan este modelo y lo aplican para obtener una buena solución inicial.

En la Figura 30 se presenta el esquema de la estrategia arriba mencionada.

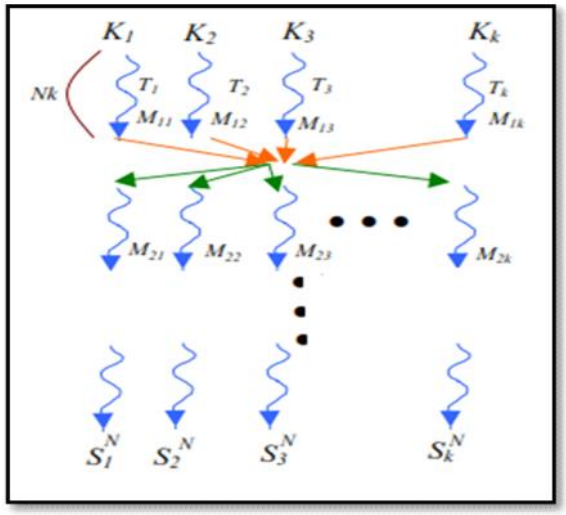


Figura 30. Estrategia de búsquedas simultáneas con interacciones periódicas

Una preocupación al paralelizar el algoritmo, es la no existencia de un teorema que pruebe la convergencia del Recocido Simulado distribuido con reinicio, a diferencia del recocido simulado secuencial, entonces el comportamiento global del distribuido debe ser lo más semejante posible al del secuencial. Para lograr esto se usó el criterio propuesto en el capítulo 7 por (Azencott, 1992), el cual propone una estrategia de paralelización independiente que mantiene las propiedades de convergencia del algoritmo secuencial.

4.5.2 Modelos de comunicación entre procesos

Estos modelos se utilizan para comunicar y/o sincronizar procesos paralelos. Los modelos de comunicación son:

I. **Modelo maestro-esclavo.** En las redes de computadoras, este es un modelo para un protocolo de comunicación en el que un proceso (proceso maestro) impone la dirección del control del programa a uno o más procesos (esclavos) que son los que procesan la información, la única comunicación que tienen con el maestro es la entrega de resultados de la tarea asignada. En la descomposición de dominio los esclavos son idénticos. En la descomposición funcional los esclavos son diferentes. En la Fig. 31 se muestra este modelo.

   Las variantes que tiene es que pueden ser síncronos o asíncronos, el maestro puede procesar o no (activo o pasivo); el mecanismo de asignación de datos por parte del maestro puede ser dinámico o estático.

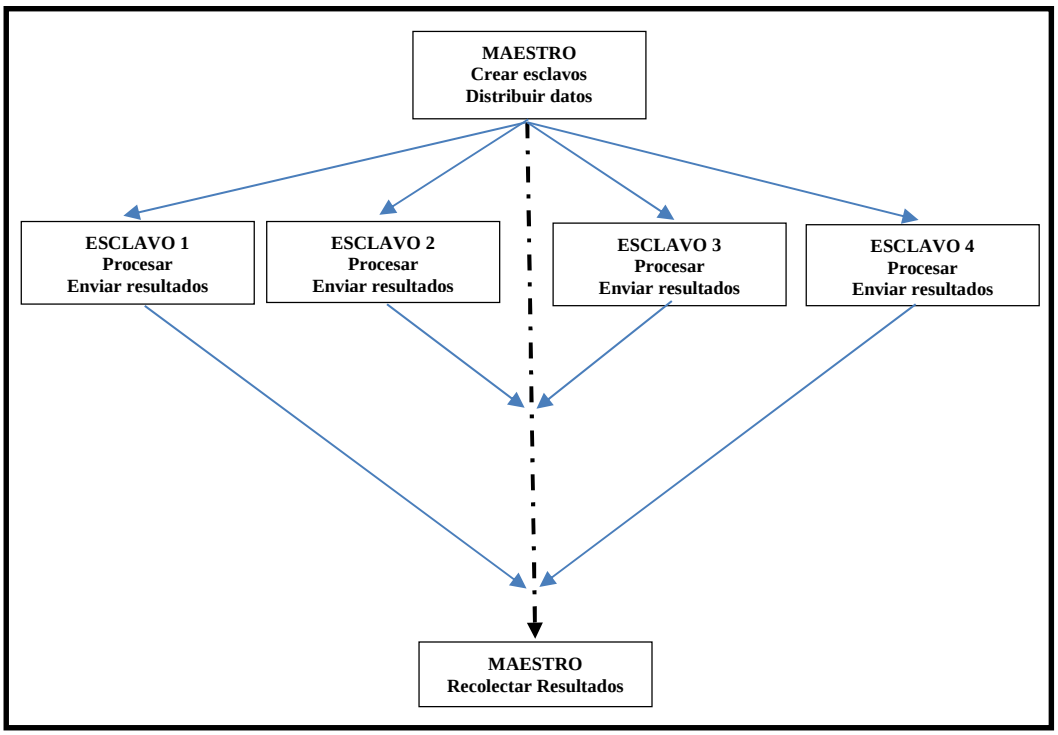


Figura 31. Modelo Maestro-Esclavo.

II. **Modelo cliente servidor.** Identifica dos clases de procesos: terminales (los clientes) que se les provee de un recurso (los servidores) que atienden los pedidos. Los clientes son los elementos que necesitan servicios del recurso y el servidor es la entidad que posee el recurso. El proceso servidor está permanentemente activo aguardando solicitudes de parte del cliente. La Figura 32 ejemplifica este modelo.

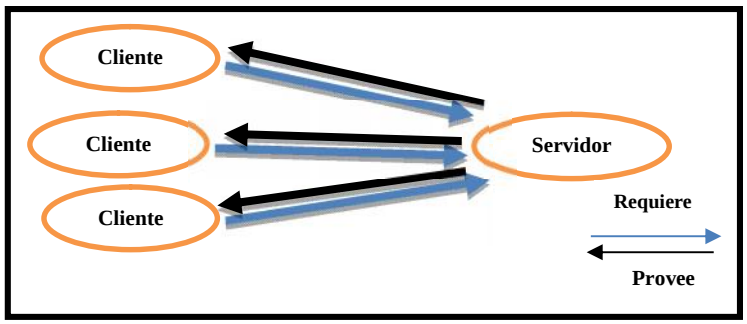


Figura 32. Modelo cliente-Servidor.

III. **Modelo peer to peer.** Modelo de comunicación en el cual cada componente tiene las mismas capacidades para establecer la comunicación.

4.5.3 Formas básicas para paralelizar un programa

Para poder paralelizar un algoritmo, es necesario contar con un lenguaje o biblioteca que permita hacerlo; dependiendo de esta herramienta y las restricciones del equipo de cómputo que se tenga, se "descompondrá" el cómputo secuencial en tareas para su ejecución en paralelo en varios procesadores, esto es conocido como "*granularidad*" (García Regis & Cruz Martínez, 2003).

Se le llama *granularidad* al tamaño de las tareas en que se divide un algoritmo, éstas pueden ser desde una sentencia de código, una función o un proceso que será ejecutado en paralelo.

Paralelización de grano fino, cuando el código es dividido en tareas a nivel sentencia o cuando divide un ciclo en varios subciclos (se le conoce como Paralelismo de Instrucción).

Paralelización de grano medio, el código se descompone a nivel subrutinas o segmentos de código (poca división del código).

Paralelización de grano grueso, se le conoce como Paralelismo de Tareas. Es importante identificar las tareas del código que se pueden ejecutar en paralelo, esto es distribuir las operaciones en partes iguales entre la cantidad de procesadores a utilizar, sin olvidar que hay partes del código que deben ejecutarse de forma secuencial.

Ésta última es la que más se utiliza debido a su portabilidad y que se adapta a multiprocesadores de memoria compartida y distribuida. Se conocen tres estilos distintos de programación; paralelismo en datos, ***programación por paso de mensajes*** (**MPI**) y programación por paso de datos.

4.5.4 Message Passing Interface (MPI)

Las siglas MPI significan 'Interfaz de Paso de Mensajes' y es una implementación de un estándar que define la sintaxis y la semántica de paso de mensajes diseñados para ser utilizados en programas que exploten el paralelismo en arquitecturas de múltiples procesadores (Blaise Barney, 2014).

En la actualidad, el uso de bibliotecas de paso de mensajes (como es MPI) y los clusters, han proporcionado una buena alternativa para ejecutar aplicaciones paralelas que requieren de suficiente poder de cómputo.

Una arquitectura de memoria distribuída está formada por un conjunto de procesadores, cada uno con su propia memoria local y espacio de direcciones. También es conocido como clúster y se define como un conjunto de computadoras, también llamadas nodos, que están interconectadas mediante una red de alta

velocidad que trabajan de manera conjunta para la solución de un problema, a través de configuraciones especiales que le habilitan la programación paralela a través del uso de bibliotecas de paso de mensajes. Cuando se ejecuta un algoritmo distribuido en un cluster, sus instrucciones y datos son distribuidos entre los procesadores con el objetivo de obtener un buen desempeño.

MPI puede definirse como: Una colección de rutinas que facilitan la comunicación entre procesadores en programas paralelos. Esta propuesta de ejecución del paralelismo mediante el envío de mensajes es a nivel de grano grueso y su principal esquema es la comunicación y generación de los procesos con el modelo Maestro-Esclavo.

En este modelo, el maestro divide y distribuye el trabajo en pequeñas tareas que va asignando a cada nodo denominado como “esclavo”. Al terminar de ejecutar su tarea le envía los resultados de su trabajo al maestro para que éste recopile la información y se pueda mostrar al usuario. En esta investigación, el mismo programa se ejecuta en los diferentes procesadores (un solo programa con múltiples datos SPMD por sus siglas en inglés).

MPI ofrece varias características encaminadas a mejorar el rendimiento de los programas. Entre sus principales características destacan:

- Un conjunto de rutinas de comunicación punto a punto.
- Un grupo de rutinas de comunicación entre grupos de procesos.
- Un contexto de comunicación para el diseño de bibliotecas paralelas.
- La posibilidad de especificar diferentes topologías de comunicación.
- La posibilidad de crear tipos de datos derivados para enviar mensajes que contengan datos no contiguos en memoria.
- En MPI se puede hacer uso de comunicación asíncrona.
- MPI maneja más eficientemente el paso de mensajes.
- Usando MPI se pueden desarrollar aplicaciones más eficientes en MPP y clústeres.
- MPI es totalmente portable.

- Existen varias implementaciones de calidad de MPI disponibles (LAM, MPICH, CHIMP).
- Existe una implementación de MPI para C.
- Hay especificación formal de MPI.
- MPI es un estándar

Cada programa en MPI debe incluir la librería mpi.h que contiene las definiciones y las declaraciones necesarias para compilar un programa que use la librería.

Todos los identificadores de MPI contienen el prefijo 'MPI' en sus nombres de funciones. Antes de llamar a cualquier función MPI, es necesario invocar además MPI Init.

Cuando el programa termine de ejecutarse debe invocarse MPI Finalize.

Se puede resumir la estructura en 3 pasos básicos:

1) *Inicializar la comunicación*

MPI_Init ()	Función que inicializa el ambiente MPI
MPI_comm_size ()	Retorna el número de procesos
MPI_Comm_rank ()	Indica el número de cada procesador

2) *Comunicar para compartir datos entre procesos (punto a punto)*

MPI_Send ()	Envía un mensaje
MPI_Recv ()	Recibe un mensaje

3) *Finalizar el ambiente paralelo.*

MPI_Finalize ()	Finaliza el ambiente paralelo de MPI

La estructura básica de cualquier programa MPI se incluye en el Algoritmo de la Figura 33, (González-Álvarez, 2013).

Estructura básica de cualquier programa MPI

```
1.  # include <mpi.h>
2.  int main() {
3.  int nproc;   /*Número de procesos*/
4.  int rank;    /*Mi dirección; 0<=rank<=(nproc-1)*/
5.  MPI_Init(&argc, &argv);
6.  MPI_Comm_size(MPI_COMM_WORL,&nproc);
7.  MPI_COMM_rank(MPI_COMM_WORL,&rank);
8.  /*CUERPO DEL PROGRAMA*/
9.  MPI_Finalize();
10. Return 0;
11. }
```

Figura 33. Estructura Básica de un Algoritmo en MPI.

4.5.5 Tipos de comunicación

Un punto importante en MPI es el tipo de comunicaciones, se determinan con base en el número de procesos que intervienen en la comunicación. Se conocen dos tipos de comunicación:

Punto a punto, es el mecanismo básico de transferencia de mensajes entre un par de procesos, uno enviando y el otro recibiendo. MPI provee de un conjunto de funciones de envío y recepción de mensajes que permiten la comunicación de datos de cierto tipo con una etiqueta asociada (Hidrobo, 2005). La comunicación punto a punto consiste en tener exactamente un proceso emisor que ejecuta una operación de envío y un proceso receptor que ejecuta la operación de recepción, esto puede observarse en la Figura 34.

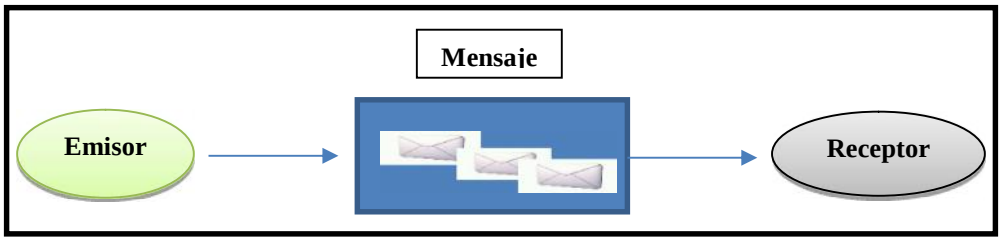


Figura 34. Comunicación punto a punto.

Existen ciertas condiciones que deben tomarse en cuenta para poder trabajar con la comunicación punto a punto:

- Cada mensaje enviado dentro de la red de comunicación será recibido por el receptor procesador de forma secuencial.
- Debe de existir siempre un proceso receptor de los mensajes enviados, de no ser así, ocurrirán errores.
- Se debe cuidar que las variables enviadas sean del mismo tipo: enteros, flotantes, etc.

Dentro de la comunicación punto a punto se tienen dos tipos de comunicaciones:

a) Comunicación bloqueante, en este tipo de comunicación, el procesador emisor tiene que "esperar" hasta que el procesador receptor complete las etapas de su programa para que pueda continuar su ejecución. Tiene la desventaja de producir tiempos muertos que afectan el aprovechamiento y utilización de los procesadores.
b) Comunicación no bloqueante, su característica es que no interrumpe la ejecución del procesador cuando se realizan operaciones de envío y recepción.

Colectiva, se usan cuando se requiere enviar un conjunto de datos a múltiples procesos o cuando se necesita recibir en un único proceso datos de varios remitentes. Este tipo de comunicaciones permiten la transferencia de datos entre todos los procesos que pertenecen a un grupo específico, se usan comunicadores en lugar de etiquetas. En este tipo de comunicación todos los procesos que pertenecen a un grupo están involucrados en el intercambio de datos de manera simultánea.

MPI proporciona las funciones siguientes para la distribución de datos:

- Difusión de datos *Broadcast*
- Dispersión de datos *Scatter*
- Reunión de datos *Gather*
- Reunión en todos *Allgather*
- Todos a todos *all to all*
- Reducción *Reduce y Allreduce.*

En la figura 35 se muestra un ejemplo de comunicación colectiva que es la distribución de un dato de un emisor a varios receptores.

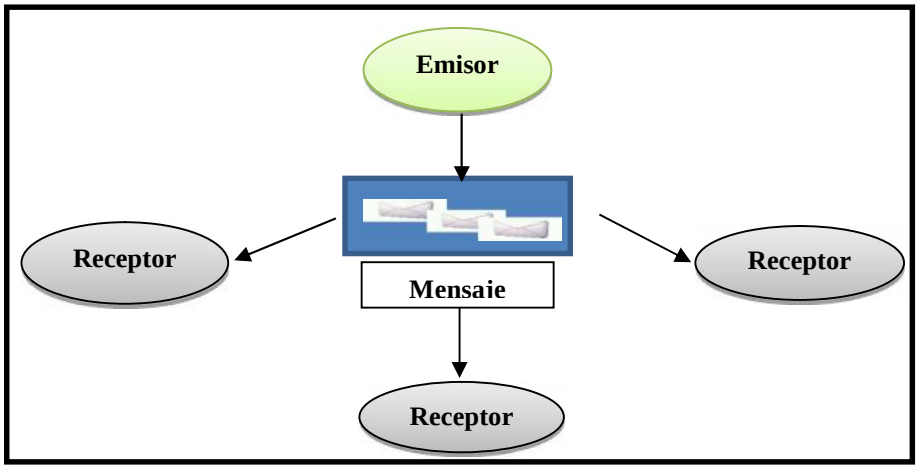


Figura 35. Comunicación colectiva

4.6 Análisis de rendimiento de algoritmos distribuidos

En este apartado se presentan algunas métricas para medir el rendimiento de un programa paralelo. En este tipo de análisis, influyen varios factores: almacenamiento de datos en dispositivos, comunicación de datos, generación de la información, entre otros.

Para medir el rendimiento se puede hacer uso de métricas diversas como lo son: tiempo de ejecución, escalabilidad, eficiencia, portabilidad, escalabilidad, etc.

4.6.1. Tiempo de ejecución

El tiempo de ejecución es el índice más intuitivo, es la medida utilizada para evaluar la eficiencia computacional de un programa. Es un parámetro absoluto pues permite medir la rapidez del algoritmo sin compararlo con otro. En el caso de un programa secuencial, consiste en el tiempo transcurrido desde que se lanza su ejecución hasta que finaliza. En el caso de un programa paralelo, el tiempo de ejecución es el tiempo que transcurre desde el comienzo de la ejecución del programa en el sistema paralelo hasta que el último procesador culmine su ejecución (A. Grama, 2003). Se le considera una medida del desempeño.

Para sistemas paralelos con memoria distribuida el tiempo paralelo (**TP**) con *p* procesadores, se determina de modo aproximado mediante la fórmula de tiempo paralelo de la ecuación (19).

$$T_P \approx T_A + T_C - T_{SOL} \tag{19}$$

Donde:

TA es el tiempo aritmético, es decir, el tiempo que tarda el sistema multiprocesador en hacer las operaciones aritméticas. El tiempo aritmético se expresa en cantidad de FLOPs, donde el FLOP es una medida que indica la velocidad que tarda el procesador en realizar una operación aritmética en punto flotante. El tiempo aritmético depende de dos parámetros de la red, uno es el tiempo de envío de una palabra, que se denota con el símbolo τ, y el otro es el tiempo que transcurre desde que el procesador decide enviar el mensaje hasta que el mensaje circula por la red, que se denota con el símbolo β.

TC es el tiempo de comunicación, o sea, el tiempo que tarda el sistema multiprocesador en ejecutar transferencias de datos;

TSOL es el tiempo de solapamiento, que es el tiempo que transcurre cuando las operaciones aritméticas y de comunicaciones se realizan simultáneamente. El tiempo de solapamiento suele ser muchas veces imposible de calcular tanto teórica como experimentalmente, en cambio puede influir bastante en el tiempo total de ejecución

del algoritmo paralelo. No obstante, la dificultad de su cálculo condiciona que se realice la aproximación, la fórmula para tiempo paralelo simplificada se presenta en la ecuación (20):

$$T_P \approx T_A - T_C \tag{20}$$

4.6.2. Velocidad (Speed-Up)

El Speed-Up es una medida de la mejora del rendimiento de una aplicación, al aumentar la cantidad de procesadores. Para p procesadores, Sp (también denominado como $S_{(n)}$), es el cociente entre el tiempo de ejecución de un programa secuencial, T_S, y el tiempo de ejecución de la versión paralela de dicho programa en p procesadores, T_P. Dado que pueden existir distintas versiones secuenciales, se elige el TS de la versión secuencial más rápida. Este índice indica la ganancia de velocidad que se ha obtenido con la ejecución en paralelo; la [ecuación 21] es la fórmula para calcular el Speed up.

$$S_P = \frac{T_S}{T_P} \tag{21}$$

Por ejemplo, un Speed-Up igual a 2 indica que se ha reducido el tiempo a la mitad al ejecutar el programa con varios procesadores. En el mejor de los casos el tiempo de ejecución de un programa en paralelo con *p* procesadores será p veces inferior al de su ejecución en un sólo procesador, teniendo todos los procesadores igual potencia de cálculo. Es por ello que el valor máximo que puede alcanzar el Speed-Up de un algoritmo paralelo es *p*. Generalmente el tiempo nunca se verá reducido en un orden igual a *p*, ya que hay que contar con la sobrecarga extra que aparece al resolver el problema en varios procesadores, debido a sincronizaciones y dependencias entre ellos.

4.6.3. Eficiencia

Un algoritmo es **eficiente** cuando logra llegar a sus objetivos planteados utilizando la menor cantidad de recursos posibles, es decir, minimizando el uso memoria, de pasos y de esfuerzo humano.

Es la medida del uso de los recursos computacionales requeridos por la ejecución de un algoritmo en función del tamaño de las entradas, esto es en menor tiempo y con menores recursos

Eficiencia es el cociente entre el Speed-Up y el número de procesadores. Significa el grado de aprovechamiento de los procesadores para la resolución del problema. El valor máximo que puede alcanzar es 1, que significa un 100 % de aprovechamiento. La E significa la fracción de tiempo que los procesadores se mantienen haciendo trabajo útil. P número de procesadores, S_p, es el speed-up. La ecuación (22) representa la fórmula para el cálculo de la eficiencia.

$$E = \frac{S_p}{P} \tag{22}$$

4.6.4. Eficacia

La palabra eficacia, se usa para definir que el algoritmo entrega el resultado esperado o realiza la labor esperada. Un algoritmo es **eficaz** cuando alcanza el objetivo primordial.

4.6.5. Escalabilidad

La escalabilidad es la capacidad de un determinado algoritmo de mantener sus características cuando aumenta el número de procesadores y el tamaño del problema en la misma proporción. En definitiva la escalabilidad suele indicar la capacidad del algoritmo de utilizar de forma efectiva un incremento en los recursos

computacionales. Un algoritmo paralelo escalable suele ser capaz de mantener su eficiencia constante cuando se aumenta el número de procesadores incluso a base de aumentar el tamaño del problema. Si un algoritmo no es escalable, aunque se aumente el número de procesadores, no se conseguirá incrementar la eficiencia aunque se aumente a la vez el tamaño del problema, con lo que cada vez se irá aprovechando menos la potencia de los procesadores.

La escalabilidad puede evaluarse mediante diferentes métricas (A. Grama, 2003). Es conveniente tener en cuenta las características de los problemas con los que se está tratando para elegir la métrica adecuada de escalabilidad.

4.7 Algoritmo Secuencial

El algoritmo de recocido simulado secuencial propuesto en este trabajo, cuenta con un reinicio, para reiniciar, toma la mejor solución obtenida resultante de cada iteración y la reingresa como solución inicial. El algoritmo se ejecuta un número determinado (n veces), cumplido el ciclo, se obtiene el mejor resultado obtenido, después de varias iteraciones, el cual es el resultado final.

Para el algoritmo secuencial se realizó un análisis de sensibilidad que permitió identificar los valores de los parámetros de control que proporcionan mejores resultados; se trabajó con las instancias ya comentadas en la sección 3.3. Para el secuencial se usaron las instancias de 35 a 238 clientes para el problema del CVRP (Media, 2016), considerados como instancias pequeñas y medianas.

4.7.1 Sintonización de los parámetros de Recocido Simulado (Análisis de Sensibilidad)

Análisis de Sensibilidad es una investigación experimental que se realiza a los parámetros de control que se usan en la ejecución del algoritmo con el objetivo de

conocer su influencia en el comportamiento estadístico en la eficacia y eficiencia del algoritmo al ejecutarlo.

El conocer estos valores permite identificar a aquellos parámetros que benefician un desempeño óptimo sobre el algoritmo. Esta actividad también es conocida como Sintonización y permite conocer cuál es la sensibilidad al cambiar (incrementando o decrementando) sus valores. Esa sintonización es importante porque permite observar la eficiencia del algoritmo a través de su funcionamiento con la configuración de los parámetros de control.

La realización de este análisis se hará aplicando el método de Análisis de Sensibilidad distribuido propuesto por Romeo y Sangiovanni-Vincentelli [(Romeo & Sangiovannai-Vincentelli, 1991)].

El objetivo de la metodología es reducir los largos tiempos que toma el realizar la sintonización por la cantidad de parámetros y repeticiones que hay que hacer para probar cada configuración propuesta.

Es necesario comentar que el algoritmo trabaja de acuerdo a parámetros de control cuya sintonización debe ser hecha para cada conjunto de instancias de trabajo (Gendreau, Laporte, & Potvin, 2002).

La sintonización propuesta se presenta en la sección siguiente.

4.7.2 Metodología del Análisis de Sensibilidad

Esta metodología consiste en considerar un conjunto de parámetros de control $P\forall = \{p_1, p_2, p_3, \dots, p_n\}/p \in P$ que van a ser sintonizados por un conjunto de valores

C = {c1, c2, c3,… cc} que serán analizados para reducir el tiempo de cómputo requerido para obtener su sintonización.

El análisis de sensibilidad se realiza de acuerdo a las consideraciones siguientes:

1. ***Identificar los parámetros de control*** de la metaheurística que se está trabajando.

Para el Recocido Simulado (RS). Los parámetros de control se muestran en la Tabla 6.

Tabla 6. Parámetros del R.S.

Parámetros Recocido Simulado			
T_o	T_f	LCM	α

De donde:

T_0 = Parámetro de control inicial, (temperatura inicial).

T_f = Parámetro de control final, (temperatura final).

LCM = Longitud de la Cadena de Markov

α = Factor de decremento del parámetro de control inicial (alfa).

El valor del parámetro de control ***temperatura***, determina en qué medida pueden ser aceptadas soluciones vecinas peores que la actual.

La ***cadena de Markov*** permite la generación de un número definido de soluciones vecinas en cada iteración. La generación de LCM soluciones vecinas es considerada una iteración, una vez finalizada se enfría la temperatura y se pasa a la siguiente iteración. Con el aumento de iteraciones disminuye la probabilidad de aceptar soluciones peores que la actual.

Alfa va reduciendo en cada iteración la temperatura inicial en pequeños decrementos hasta alcanzar la temperatura final.

2. ***Definir rangos numéricos de los parámetros***; el método de Romeo y Sangiovanni-Vincentelli y (Juárez Pérez, 2013), permite encontrar de forma analítica los parámetros del algoritmo, le llaman “sintonización analítica de parámetros”. En el método indican que las temperaturas inicial y final están en función de los incrementos máximo y mínimo del costo obtenido de la

estructura de vecindad usada. Los valores mínimos y máximos se obutuvieron de diferentes papers de la literatura. Ver Tabla 7.

El método considera los siguientes conceptos: El criterio de vecindad, los deterioros máximos y mínimos y la probabilidad de aceptación deseada al principio y al final del algoritmo.

Tabla 7. Valores Mínimos y Máximos para los parámetros del R. S.

Parámetros	Mínimo	Máximo
T_o	5	29
T_f	0.000001	1.68
α	0.88	0.994
LCM	5000	90000

Hay investigaciones en las que apoyan el criterio de selección de sus temperaturas iniciales y finales basándose en el criterio de aceptación de Boltzmann, cada temperatura produce una tasa de aceptación, a mayor temperatura, mayor probabilidad de aceptación de soluciones peores**.** Así, al inicio se permite la exploración del espacio de soluciones**.** Al ir disminuyendo la temperatura la probabilidad de aceptar soluciones malas se reduce permitiendo la explotación de este espacio.

Para el Recocido Simulado los valores de aceptación para la temperatura inicial se definen entre 97% - 95% y para la temperatura final, valores del 5% hasta el 3% se consideran buenos (Alonso-Pecina F. , 2008).

Valores menores del 90% se consideran inicios “tibios” y la probabilidad de aceptación no permite realizar una buena exploración. Por el contrario valores mayores al 5% se consideran como un enfriamiento rápido que no permite alcanzar su equilibrio térmico y el resultado será una mala solución.

3. ***Tamaño de las muestras***, éstas se usarán para evaluar el comportamiento del algoritmo y así sintonizar cada instancia. Se determinan por el número de incrementos que le aplican a cada parámetro. La Tabla 8 presenta los parámetros de recocido simulado, los incrementos que se darán a cada valor inicial y el número de muestras generadas.

Tabla 8. Incremento para los parámetros del R. S.

Parámetros	Incremento	# Muestras
T_o	3	9
T_f	*6	9
α	0.018	9
LCM	7250	9

4. ***Las configuraciones*** correspondientes a cada parámetro se muestran en la Tabla 9.

Tabla 9. Configuraciones de los parámetros de RS

T_o	5	8	11	14	17	20	23	26	29
T_f	0.000001	0.000006	0.00004	0.00022	0.0013	0.008	0.05	0.3	1.68
LCM	10000	17250	24500	31750	39000	46250	53500	60750	68000
α	0.85	0.868	0.886	0.904	0.922	0.94	0.958	0.976	0.994

5. Proceso de sintonización de Parámetros, el proceso se enfoca en la evaluación de los valores que conforman el conjunto de parámetros de control definidos para el algoritmo de recocido simulado, la intención es fijar el valor de la configuración que permite obtener los mejores resultados.

El procedimiento para sintonizar es el siguiente: *Mantener fijos los valores (c_c) de los parámetros de control (p_p) e ir evaluando las diferentes configuraciones generadas al ir variando el primer parámetro con los valores de las muestras generadas.* Un ejemplo de esta actividad se presenta en la tabla 10, el parámetro T_o es el que se va a ir variando y los demás se mantendrán fijos.

Tabla 10. Configuraciones de la sintonización de parámetros

Permutación	A	B	C	D	E	F	G	H	I
T_0	5	8	11	14	17	20	23	26	29
T_f	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001
LCM	10000	10000	10000	10000	10000	10000	10000	10000	10000
A	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93

De cada configuración se realizan 30 ejecuciones (Alba E. , 2005) hasta cubrir el número total de éstas de cada parámetro. El valor de la configuración que genere el mejor resultado (con respecto a la función objetivo) como media de las 30 ejecuciones queda fijo para repetir el proceso para el parámetro siguiente a evaluar.

Este procedimiento permitió fijar los valores de los parámetros que corresponderían a la configuración que estadísticamente produce los mejores resultados.

4.7.3 Convergencia del Algoritmo

Si un algoritmo ha sido implementado correctamente, las soluciones generadas a través de las iteraciones mejorarán permitiendo que se alcance el óptimo global, al realizar un número de repeticiones (iteraciones), las aproximaciones obtenidas terminan por acercarse cada vez más al verdadero valor buscado, a este proceso se le denomina convergencia.

La convergencia de un algoritmo es una característica que permite identificar en qué punto un algoritmo alcanza la estabilidad. Se dice que el algoritmo converge cuando el 95% de la soluciones sufren variaciones de no más del 5% en sus resultados. Para garantizar la convergencia de un algoritmo en recocido simulado, el número de iteraciones en cada decremento de temperatura puede ser muy grande para que la cadena de Markov subyacente alcance un estado constante (Robusté Anton & Galván, 2005).

Para determinar la convergencia se realizaron pruebas con los valores de sintonización que se obtuvieron en la sección 4.7.2; se ejecutó cada instancia 30 veces. Los parámetros considerados para determinar la convergencia fueron el porcentaje de aceptación de Boltzmann y el número de iteraciones que se consideró como criterio de paro del algoritmo por ser el parámetro que incrementa el tiempo de ejecución del algoritmo. Se le consideró a Boltzmann como parámetro debido a que el recocido simulado trabaja con base en el criterio de soluciones malas aceptadas como se explicó en el apartado 4.7.2.

En las figuras siguientes (36, 37, 38 y 39), se muestran las gráficas de las instancias sintonizadas (CH1, CH2, Mn151 y CH5); los resultados de la temperatura final son menores al 5% que se consideran buenos para obtener buenas soluciones. Éstas se presentarán en la sección 6.2. En las gráficas se presenta el número de iteraciones y su porcentaje de aceptación de Boltzmann para la temperatura final.

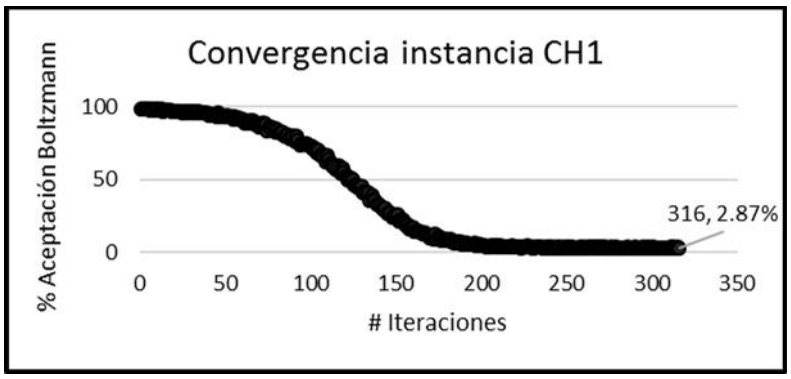


Figura 36. Gráfica de convergencia instancia CH1 (50 clientes).

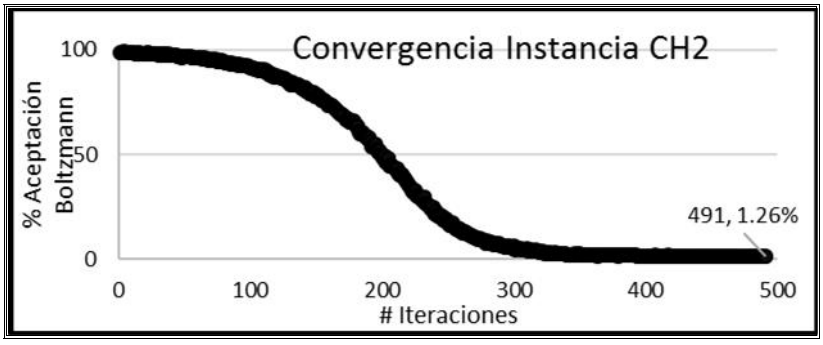


Figura 37. Gráfica de convergencia instancia CH2 (75 clientes).

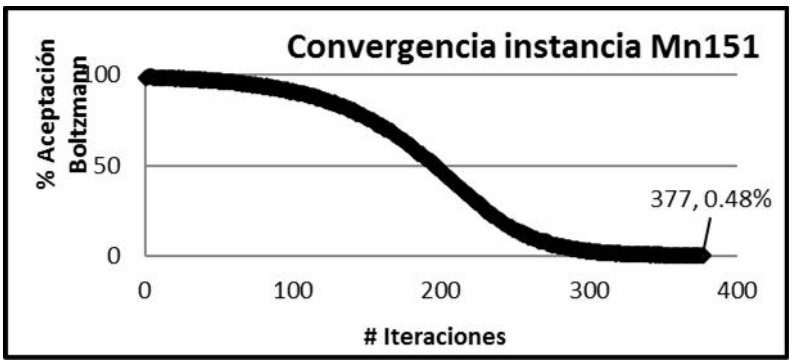


Figura 38. Convergencia de la instancia Mn151 (150 clientes).

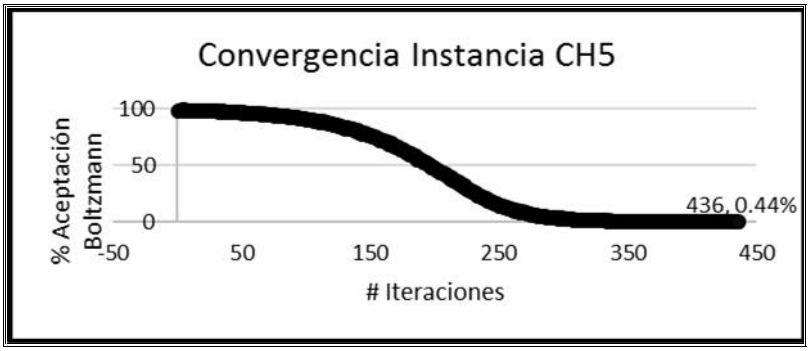


Figura 39. Convergencia instancia CH5 (199 clientes).

En las gráficas puede observarse que el número de iteraciones no es el mismo para las instancias presentadas. El número de iteraciones depende de la instancia a evaluar, los parámetros de temperatura (inicial y final) y del parámetro de control, aplicados en la ejección del algoritmo.

4.8 Algoritmo Paralelo con Comunicación Colectiva

En esta sección, se muestran las características del algoritmo distribuido desarrollado en este trabajo para el problema del CVRP. El modelo que se trabaja es el de maestro-esclavos, fue desarrollado en lenguaje de programación C y con la biblioteca de paso de mensajes MPI que es compatible con las versiones de Intel [MPI, 2013], OpenMPI [OpenMPI, 2015] y MPICH [MPICH, 2013].

Un aspecto importante en el cómputo paralelo, es la organización lógica de los procesos que se incluirán en la ejecución del algoritmo. Esta elección junto con el algoritmo desarrollado para resolver el proceso es muy importante para obtener un rendimiento óptimo de la paralelización.

La organización empleada para este trabajo fue la del esquema “maestro-esclavo”. En la cual, el proceso “maestro” administra el trabajo de los demás y puede o no participar en el cálculo. Los otros procesos se denominan “esclavos”.

El modelo implementado permite mantener la secuencia del algoritmo secuencial pero es organizado, sincronizado y coordinado por el nodo maestro que manda a cada procesador su carga de trabajo.

Esta carga consiste en entregar a cada nodo esclavo: un ruteo inicial de una misma instancia para darle solución, el valor de los parámetros del esquema de enfriamiento del recocido simulado. Ellos resuelven el problema y entregan resultados.

El nodo “maestro” recibe las soluciones de los esclavos y selecciona la mejor.

La ejecución del algoritmo paralelo inicia en el nodo maestro con la lectura de la instancia de prueba y el ruteo inicial. En esta parte, el maestro se encarga de realizar la lectura de la instancia de prueba y de las diferentes soluciones iniciales (ruteos iniciales) que serán asignadas a cada nodo esclavo, estos ruteos se obtuvieron del algoritmo CVRPUK de (D’Granda-Trejo, 2013), no las genera el algoritmo de recocido simulado.

Cada esclavo se recibe una solución inicial (ruteo inicial), y es responsable de resolver el problema asignado, realiza el cálculo de la matriz de distancias y aplica el algoritmo de recocido simulado.

El modelo aplica un reinicio, este reinicio trabaja de la forma siguiente:

- El nodo esclavo inicia la ejecución del algoritmo con una solución inicial S_0, que es diferente a los demás nodos involucrados.

- En cada iteración del algoritmo de recocido simulado, cada nodo esclavo va obteniendo una solución que es llamada Mejor Solución Recocido (MRS), el algoritmo toma el valor obtenido de esta solución y lo reingresa nuevamente como solución inicial.
- El reinicio se ejecuta un número determinado (n veces), cumplido el ciclo, cada proceso esclavo entrega su solución al nodo maestro el cual evalúa los resultados enviados y se queda con el mejor el cual es el resultado final.

En la Figura 40 se muestra el diagrama del algoritmo paralelo.

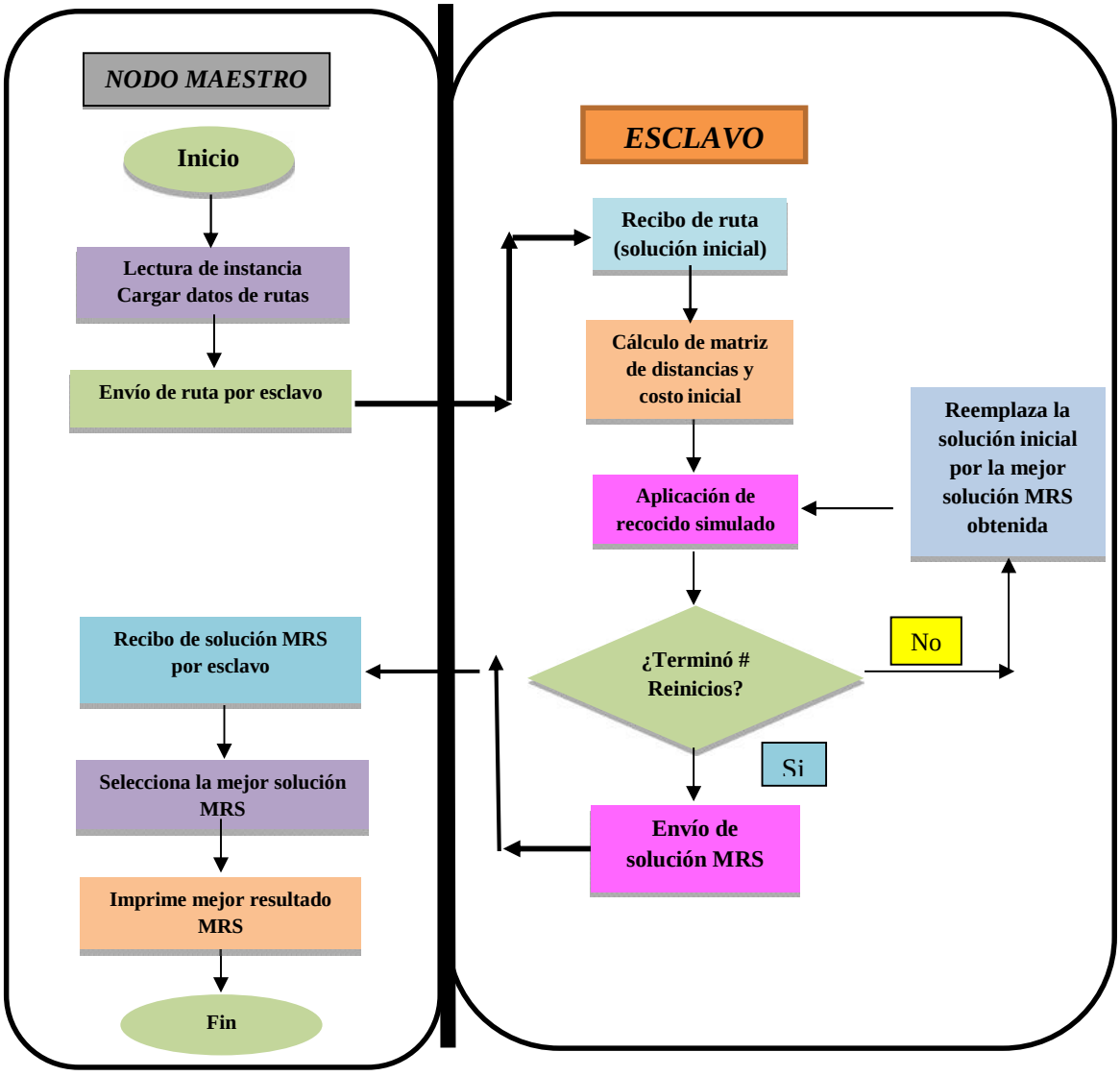


Figura 40. Diagrama del Algoritmo de Recocido Simulado distribuido con reinicio.

Se puede decir que solo se dan dos comunicaciones entre el nodo maestro y los esclavos; al inicio cuando el nodo maestro entrega la información y la segunda vez son los esclavos quienes se comunican con el maestro para entregarle los resultados. En la Figura 41 se esquematiza esta comunicación. El tipo de comunicación es colectiva (ver 4.5.5).

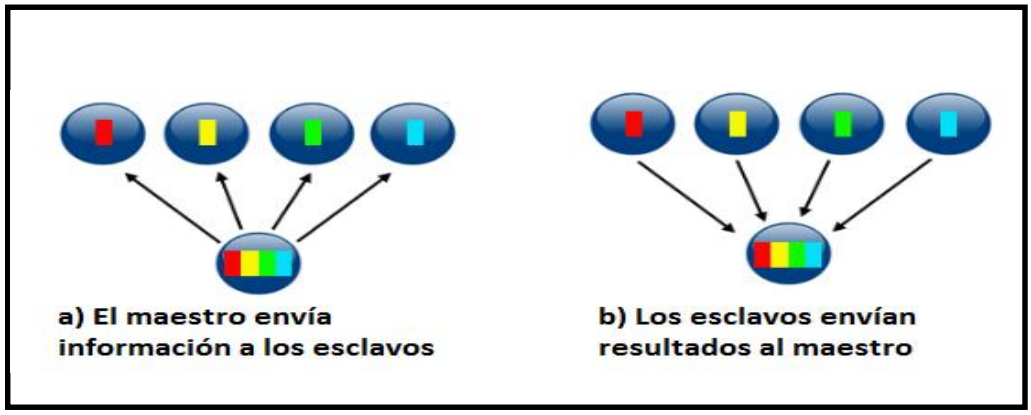


Figura 41. Momentos de comunicación entre maestro y esclavos.

Para implementar el modelo de maestro-esclavos, es necesario diseñar los grupos que van a comunicarse. El algoritmo necesita conocer las diferentes soluciones iniciales, esto se hace a través de la estructura instancia en su estructura ***s_actual***, la cual contiene los diferentes ruteos iniciales, la estructura ***s_vecino*** será a la que se le aplicará el mecanismo de perturbación que generará las soluciones vecinas que ingresarán al algoritmo para su optimización, la estructura ***s_mejorRS*** guardará el mejor valor obtenido de la aplicación del algoritmo (línea 1). En línea 2 se inicializa el ambiente para la ejecución de MPI para poder usar las rutinas definidas en la biblioteca. En las líneas 3 y 4 se identifican los procesos y el número de ellos. En la línea 5 se identifican las variables que se usan para la ejecución del algoritmo. La línea 6 asigna las funciones que realizará el maestro como son la lectura de la instancia y sus características (7, 8, 9 y 10); la demanda de cada cliente y su ubicación en el plano cartesiano (líneas 11 a 20); sus diferentes soluciones iniciales (línea 21) son enviadas desde un proceso “maestro” a todos los nodos “esclavos” en el comunicador por la instrucción MPI_Bcast.

En la línea 22 se asignan las diferentes ruteos iniciales a cada esclavo, a partir de esta línea, cada esclavo realiza el cálculo: de la matriz de distancias (línea 23); del costo de la solución inicial entregada (línea 24); la función de mejora (línea 25); el cálculo de la solución inicial después de la mejora (línea 26). La (línea 27) hace la entrega del valor de la solución inicial a la variable s_mejorRS como el mejor valor hasta ese momento obtenido para iniciar la ejecución del algoritmo de recocido simulado con reinicio (líneas 28 a 54). La línea 55 hace entrega de la mejor solución encontrada para que vuelva a entrar el algoritmo, es la parte del reinicio. La instrucción MPI_Gather (línea 56) reúne los valores del grupo de procesos. De la línea 57 a la 63 se hace una evaluación de los resultados obtenidos y se selecciona al mejor que se le asigna a s_mejorRS (línea 64). En la línea 66, el nodo maestro se encarga de las impresiones de los resultados y termina en la línea 68. Las líneas 69 y 70 dan las instrucciones de MPI para liberar memoria del sistema y finaliza el ambiente MPI.

En la Figura 42 se muestra el pseudocódigo del algoritmo de Recocido Simulado distribuido con reinicio descrito en los párrafos anteriores. En el anexo 3 se muestra el código del algoritmo distribuido del recocido simulado de este proyecto.

```
1. Entrada: s_actual, s_vecino, s_mejorRS
2. MPI_init
3. rank ← MPI_Comm_rank()
4. nprocs ← MPI_Comm_size()
5. datosC, datosdema[LIMITE_MAX+2], datoscoordX[LIMITE_MAX+2],
     datoscoordY[LIMITE_MAX+2], limite, LongM, u=0.0, probAcepta = 0.0,
6.  si rank = = 0 entonces
7.          leeArchivos()
8.          limite ← LIMITE_MAX
9.          banderaCapa ← 2
10.           datosC ← datos.C
11.           para  i ← 0 hasta i<= s_rutaIni[0].numClientes; i++  hacer
12.                  datosdema[i] ← datos.dema[i]
13.           fin para
```

```
14.          para i ← 0 hasta i<= s_rutaIni[0].numClientes; i++   hacer
15.                datoscoordX[i] ← datos.coordX[i]
16.          fin para
17.          para i ← 0 hasta i<= s_rutaIni[0].numClientes;  i++  hacer
18.                datoscoordY[i] ← datos.coordY[i]
19.          fin para
20.    fin si
21.    MPI_Bcast(&s_rutaIni, 1, ruteo, &s_actualVEC1, 1, ruteo, 0, MPI_COMM_WORLD);
22.    s_actual ← s_actualVEC
23.    funcion distancias(s_actual, mDistancias, datoscoordX, datoscoordY)
24.    s_actual ← funcion calculaCostoVh( s_actual, mDistancias, datosdema)
25.    s_actual ← funcion mejora(s_actual, mDistancias, rank)
26.    s_actual ← funcion calculaCostoVh( s_actual, mDistancias, datosdema)
27.    s_mejorRS ← s_actual
28.    para reinicio ← 0 hasta reinicio<300; reinicio++ hacer
29.       s_actual ← s_mejorRS
30.       para tempIni ← tempInicial hasta tempIni >= tempFin;   tempIni ← tempIni * alfa  hacer
31.            para metro ← 0 hasta metro <= LongM; metro++ hacer
32.                s_vecino ← s_actual
33.                s_vecino ← funcion perturba(s_vecino, mDistancias)
34.                s_vecino ← funcion calculaCostoVh(s_vecino, mDistancias, datosdema)
35.                capa ← funcion evaluaCapacidad(s_vecino, datosC)
36.                     si capa entonces
37.                          si s_vecino.costoSolucion < s_actual.costoSolucion entonces
38.                               s_actual ← s_vecino
39.                               si s_vecino.costoSolucion < s_mejorRS.costoSolucion entonces
40.                                    s_mejorRS ← s_vecino
41.                               fin si
42.                          fin si
43.                          si no
44.                                 probAcepta <- exp ((-1)*((double)(s_vecino.costoSolucion
45.                                 s_actual.costoSolucion)/100/tempIni))
46.                                 u ← ((double) rand () / RAND_MAX)
47.                                 si u < probAcepta entonces
```

```
48.                                     s_actual <- s_vecino
49.                                 fin si
50.                             fin si no
51.                         fin si
52.             fin para
53.         fin para
54.     fin para
55.     s_actualVEC ← s_mejorRS
56.     MPI_Gather(&s_actualVEC, 1, ruteo, &s_rutaINI[rank], 1, ruteo, 0,
    MPI_COMM_WORLD);
57.     s_mejorRS.costoSolucion ← s_rutaIni[0].costoSolucion
58.     para j ← 0 hasta j < nprocs; j++  hacer
59.         si s_rutaIni[j].costoSolucion < s_mejorRS.costoSolucion entonces
60.             s_mejorRS.costoSolucion <- s_solucionfinal[j].costoSolucion
61.             pos ← j
62.         fin si
63.     fin para
64.     s_mejorRS <- s_rutaIni[pos]
65.  MPI_Barrier (MPI_COMM_WORLD);
66.     si rank = = 0 entones
67.   funcion imprimeRutaOptimizada(s_mejorRS, tempInicial, tempFin, alfa, LongM, tiempo,
    rank)
68.     Salida rutas optimizadas generadas a partir de las diferentes soluciones iniciales
69.  MPI_Type_free(&ruteo);
70.  MPI_Finalize();
```

Figura 42. Pseudocódigo del Algoritmo de Recocido Simulado distribuido con reinicio

4.9 Herramientas hardware

Las aplicaciones para sistemas de procesamiento paralelo, son programas que necesitan para su ejecución equipos dotados de N procesadores, de modo que en cada uno de ellos pueda correr simultáneamente un proceso creado en tiempo de ejecución compartiendo entre todos alguna zona de la memoria para que el tiempo total que le tome ejecutarse a la aplicación completa se reduzca con relación al que tomaría ejecutarla en un equipo secuencial monoprocesador.

Un programa escrito para ser procesado en máquinas paralelas, incluirá sentencias especiales (que lo transforman en paralelo): desde las encargadas de la creación de procesos y el lanzamiento de los mismos en distintas unidades de procesamiento, hasta los permisos de acceso a zonas comunes, los pedidos de bloqueo y la liberación de las zonas comunes, etc. Insertadas dentro del código de programación secuencial.

En cuestiones de equipamiento, el proceso de paralelizar un programa involucra conocer más de la arquitectura de un cluster sobre el cual pretende paralelizar su código, conocer el número de procesadores con los que se cuenta, la cantidad de memoria, espacio en disco, los niveles de memoria disponible, el medio de interconexión, entre otros. También en cuestiones de software, se debe conocer qué sistema operativo se está usando, si los compiladores instalados permiten realizar aplicaciones con paralelismo, además hay que tener herramientas como MPI ("Message Passing Interface", Interfaz de Paso de Mensajes) en sistemas distribuidos.

4.10 Herramientas Software

Las herramientas software se componen por el ambiente paralelo de programación, el lenguaje de programación, y las librerías secuenciales y paralelas de álgebra lineal numérica. En este caso se ha empleado el lenguaje de programación ANSI C, la librería de paso de mensajes MPI (Message Passing Interface) para las arquitecturas de memoria distribuida, el API de multihilado directo explícito para memoria compartida OpenMP.

CAPÍTULO 5 PRUEBAS EXPERIMENTALES, ANÁLISIS DE RESULTADOS

En este capítulo, se valida de manera experimental el desempeño del algoritmo de recocido simulado desarrollado en este trabajo y descrito en capítulos anteriores. Se muestra el desempeño en función de la eficiencia y la eficacia en el algoritmo secuencial y distribuido, además de la infraestructura utilizada en el desarrollo de esta investigación.

En el capítulo se incluyen los resultados del desempeño del algoritmo secuencial y paralelo. El capítulo se organiza de la forma siguiente: El primer apartado describe las especificaciones de hardware y software utilizados para la experimentación. En la segunda sección se muestran los resultados experimentales del algoritmo secuencial, para la tercera sección se muestran los resultados obtenidos para el algoritmo distribuido. En la última sección se realizan las comparaciones de la ejecución del algoritmo secuencial y paralelo, evaluando con esto el grado de cumplimiento de los objetivos planteados en esta investigación.

5.1 Infraestructura Clúster Cuexcomate

Un clúster es un grupo de computadoras interconectadas y que cooperan entre sí para realizar una tarea común, uno de los objetivos de forma un clúster es el de obtener alto rendimiento en la resolución de un problema a través de la cooperación entre los equipos que lo componen en un menor tiempo posible.

Una vez teniendo una idea de las mejoras de desempeño que representa el uso del *clúster* sobre un equipo uniprocesador, es factible comenzar a utilizarlo en la

resolución de una variedad de problemas reales, algunas de las características que permiten clasificar los clústeres son:

a) **Alto rendimiento**; en donde se ejecutan tareas que requieren gran capacidad computacional, gran cantidad de memoria o ambas.
b) **Alta disponibilidad**; su objetivo es proveer disponibilidad y confiabilidad.
c) **Alta eficiencia**; ejecutan una gran cantidad de tareas en el menor tiempo posible, se permite la independencia de datos entre las tareas individuales.

Para el funcionamiento del clúster no es necesario que todos los nodos dispongan del mismo hardware y sistema operativo, debe disponer de una interfaz que se encargue de interactuar con el usuario y los procesos, repartiendo la carga entre los diferentes nodos del equipo.

El clúster en el cual se realizó este proyecto fue el Clúster Cuexcomate el cual forma parte de la minigrid Tarántula; el clúster se puede ver en la Figura 43, está ubicado en el Laboratorio de Optimización y Software en el Centro de Investigaciones en Ingeniería y Ciencias Aplicadas (CIICAp) que pertenece al Instituto de Investigación en Ciencias Básicas y Aplicadas.

Figura 43. Clúster Cuexcomate.

El clúster cuenta con el equipo y las características siguientes presentadas en la Tabla 11:

Tabla 11. Infraestructura Hardware y Software del Clúster de producción Cuexcomate.

Nodos de procesamiento	
CPU 01 al 04 Total 48 cores Total 96 GB RAM Total 2 TB HD	1 Motherboard • 2 procesadores Intel Xeon Six Core a 3.06 GHz, 12 MB cache • 1 HD Enterprise, 7200 RPM de 500GB • 6 módulos RAM de 4GB 1333 MHZ DDR3. Total de 24GB RAM 1 tarjeta Infiniband 400Gb/s
Comunicaciones	Switch 3 COM 24/10/100/1000 Switch Infiniband Mellanox de 18 puertos de 40 Gb/s 1 Motherboard • 2 procesadores Intel Xeon Six Core a 3.06 GHz. 12 MB cache. • 2 HD Enterprise, 7200 RPM de 500GB (para S.O.). • 6 HD Enterprise, 7200 RPM, 12 TB en total. • 6 módulos RAM de 4 GB 1333 MHZ DDR3. Total de 24 GB RAM. • 1 tarjeta Infiniband 40 Gb/s.
Nodo de procesamiento	
GPU 05 Total 896 cores Total 36 GB RAM Total 1 TB HD	1 Motherboard • 1 procesador Intel Xeon Six Core a 3.06 GHz, 12 MB cache. • 2 HD Enterprise, 7200 RPM de 500 GB. • 9 módulos RAM de 4GB 1333MHZ DDR3. Total de 36 GB RAM. • 2 tarjetas NVIDIA TESLA C2070, arquitectura Fermi, con 6 GB RAM DDR5 c/u, 448 cores c/u. • DVD/RW • Lector de memoria 1 tarjeta Infiniband 40 Gb/s

Los recursos del clúster Cuexcomate se encuentran distribuidos como se indica en la Tabla 12.

Tabla 12. Distribución de los recursos del clúster

CUEXCOMATE

Nodo	Cores
Cuexcomate	12
Ciicap01	12
Ciicap02	12
Ciicap03	12
Ciicap04	12
Ciicap-gpu01	6
Total	66

5.2 Algoritmo Secuencial

En este apartado se muestran los resultados obtenidos para el algoritmo secuencial de recocido simulado en el cual se utilizaron los benchmarks y sus respectivas instancias de Augerat, Christofides, Mingozzi y Toth and Taillard; sus características y estructuras se presentaron en el punto 3.3.

Se muestran las pruebas de eficacia al comparar los resultados contra las mejores cotas reportadas en la literatura y el análisis de las pruebas de eficiencia considerando el tiempo requerido para obtener la solución. También se muestra un análisis comparativo del efecto que tuvo la aplicación de una estructura híbrida de vecindad contra la aplicación de un solo movimiento de perturbación.

5.2.1 Análisis de Eficacia

En esta sección se describen los resultados de los experimentos realizados para probar el algoritmo de recocido simulado en el CVRP. Ya que se han sintonizado los parámetros a través del proceso de sintonización mostrado en 5.1.2, los valores obtenidos son utilizados para realizar las pruebas de eficacia del algoritmo de

recocido simulado. Este paso es importante porque al realizarles un análisis estadístico se puede observar el comportamiento del algoritmo.

Las pruebas se realizaron en el clúster Cuexcomate. Para el análisis estadístico se realizaron 30 ejecuciones de cada instancia que es la cantidad mínima requerida para poder realizar una interpretación estadística del benchmark evaluado (Alba E. , 2005).

La solución inicial es contruída por medio del algoritmo de agrupamiento llamado UKCVRP (D'Granda-Trejo, 2013). El algoritmo de recocido simulado fue codificado en lenguaje C y ejecutado en el nodo de procesamiento GPU con 36 GB RAM, 1 Six Core Intel Xeon processor a 3.06 GHz, 12 MB de memoria en cache.

El algoritmo fue probado en un conjunto de instancias de los benchmarks de: Christófides, Mingozzi and Toth (1979), Augerat and Taillard (1995).

Se trabajó con instancias que tuvieran la característica de tener clientes geográficamente dispersos de manera aleatoria (R) y agrupadas-aleatorias (RC) por ser las que tuvieron mejor desempeño desde la generación de soluciones iniciales al cumplir con los requisitos de rutas mínimas.

Las instancias están conformadas por clientes que van desde 30 a 200. Las características de las instancias que se trabajaron en esta tesis son:

Benchmark de (Christofides, Mingozzi, & Toth, 1979), está compuesta por un conjunto de datos donde los depósitos y los clientes están localizados en un plano cartesiano. Las distancias entre ellos están definidas como una línea recta entre (Euclidean distance). La distancia recorrida se asume que es igual al tiempo de viaje. Hay catorce estancias en el benchmark de Christofides.

El benchmark tiene dos tipos de patrones de distribución: En el primero, los clientes están distribuidos de manera aleatoria (R) alrededor del depósito (instancia 1 a 10). En el segundo, los clientes son agrupados aleatoriamente (RC) (instancias 11 a 14). El número de clientes varía de 50 a 199. En las Figura 44 se ejemplifican los patrones de distribución, a la izquierda el patrón aleatorio y a la derecha el agrupado.

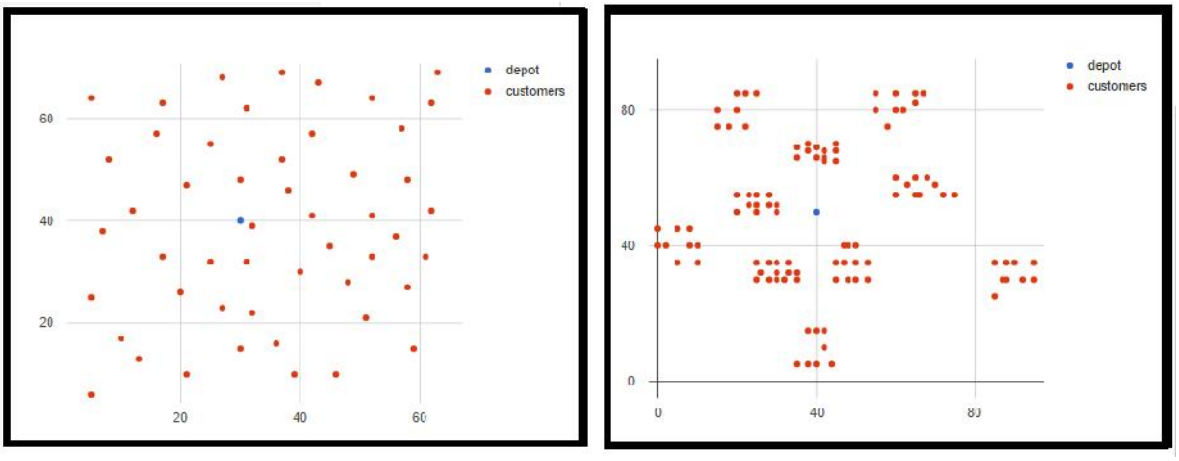


Figura 44. Patrón de distribución aleatoria y agrupada.

El benchmark de Augerat et al. Está compuesto de tres casos; (set A, B y P), en esta investigación se trabajó con el caso A donde la localización de los clientes y su distribución es aleatoria. El tamaño de las instancias varía en el rango de 31 a 79 clientes.

El benchmark de Taillard está compuesto de 15 no uniformes instancias. Los clientes están esparcidos en agrupaciones, sus demandas están exponencialmente distribuidas y es posible que un solo cliente requiera el uso de un vehículo para atender su demanda.

Los resultados son mostrados en las Tablas 12, 13 y 14. La estructura híbrida aplicada quedó conformada como sigue: 25% movimiento swap, 50% movimiento inserción y 25% movimiento cambio vecino. Para conformar esta estructura se hicieron pruebas con diferentes configuraciones y ésta fue la que mostró mejores resultados.

Las métricas usadas para evaluar el desempeño del algoritmo de recocido simulado fueron:

a) El valor de la mejor solución encontrado para cada instancia
b) El promedio de todas las 30 corridas ejecutadas por instancia
c) La desviación estándar

d) La diferencia entre la mejor solución encontrada de la instancia y la mejor cota reportada en la literatura (en porcentaje)
e) El mejor valor encontrado para cada instancia en la revisión de la literatura.

La eficiencia del desempeño del algoritmo de recocido simulado es medido por la calidad de la solución producida. La calidad está dada en términos de la desviación relativa de la mejor solución reportada en la literatura (BKS). La desviación, a la cual se le llamará desviación relativa porcentual (Relative Percent Deviation RPD), es calculada por la ecuación (25):

$$\Delta\% = RPD = \left[\frac{bets-sol}{best}\right] * 100 \quad (25)$$

Los resultados obtenidos del algoritmo de recocido simulado secuencial, para el benchmark propuesto por Christofides, Mingozzi & Toth del cual solo se trabajaron cinco instancias, la calidad de los resultados está entre 0.31% a 3.13%.

En la Tabla 13 se muestran los resultados obtenidos: Instancias de (Christofides, Mingozzi, & Toth, Chapter 11, 1979).

Tabla 13. Resultados de las pruebas para las instancias de Christofides et al.

Instancia	Clientes	Recocido Simulado			Desviacion Std	Diferencia con el óptimo	Valor mejor reportado
		El mejor	El peor	Promedio	Σ	Δ %	
Vrpn1	50	524.61	524.94	524.75	0.15	0.00	524.61
Vrpn2	75	836.14	844.05	841.44	2.08	0.31	835.26
Vrpn3	100	830.64	836.37	832.44	1.49	0.54	826.14
Vrpn4	150	1048.12	1058.87	1054.12	2.71	1.92	1028.42
Vrpn5	199	1331.68	1340.86	1336.42	3.13	3.13	1291.29

En la Tabla 14, solo tres instancias propuestas por Taillard (NEO Research Group , 2016) lograron igualar la mejor cota reportada, para el resto la calidad de las soluciones está entre 0.11% a 1.00%.

Tabla 14. Resultados de las pruebas para las instancias de Taillard.

Instancia	Clientes	Recocido Simulado			Desviacion Std	Diferencia con el óptimo	Valor mejor reportado
		El mejor	El peor	Promedio	Σ	Δ %	
Tai75a	75	1618.36	1620.76	1619.34	0.76	0.00	1618.36
Tai75b	75	1346.43	1346.64	1346.52	0.09	0.13	1344.64
Tai75c	75	1291.01	1291.63	1291.1	0.19	0.00	1291.01
Tai75d	75	1365.42	1367.19	1366.59	0.40	0.00	1365.42
Tai100a	100	2061.73	2067.48	2062.08	5.78	1.00	2041.34
Tai100b	100	1942.38	1948.46	1947.03	1.61	0.13	1939.9
Tai100c	100	1408.64	1420.6	1414.23	3.19	0.20	1406.2
Tai100d	100	1582.2	1585.45	1583.56	0.92	0.11	1580.46
Tai150a	150	3070.42	3082.52	3075.77	4.28	0.50	3055.23
Tai150b	150	2741.47	2752.72	2748.21	3.07	0.53	2727.03
Tai150c	150	2367.4	2376.81	2372.26	2.93	0.37	2358.66

En la Tabla 15, del benchmark propuesto por Augerat (NEO Research Group , 2016) se logró igualar el mejor valor reportado en nueve instancias, para el resto la calidad de las soluciones está entre 0.07% a 0.59%.

Tabla 15. Resultados de las pruebas para instancias del Benchmark de Augerat.

Instancia	Clientes	Recocido Simulado			Desviacion Std	Diferencia con el óptimo	Valor mejor reportado
		El mejor	El peor	Promedio	Σ	Δ %	
A-n32-k5	32	784	787	786	1.5	0.00	784
A-n33-k5	33	661	661	661	0.00	0.00	661
A-n33-k6	33	742	742	742	0.00	0.00	742
A-n34-k5	34	778	778	778	0.00	0.00	778
A-n36-k5	36	799	799	799	0.00	0.00	799
A-n37-k5	37	669	669	669	0.00	0.00	669
A-n37-k6	37	950	950	950	0.00	0.19	949
A-n38-k5	38	734	734	734	0.00	0.59	730

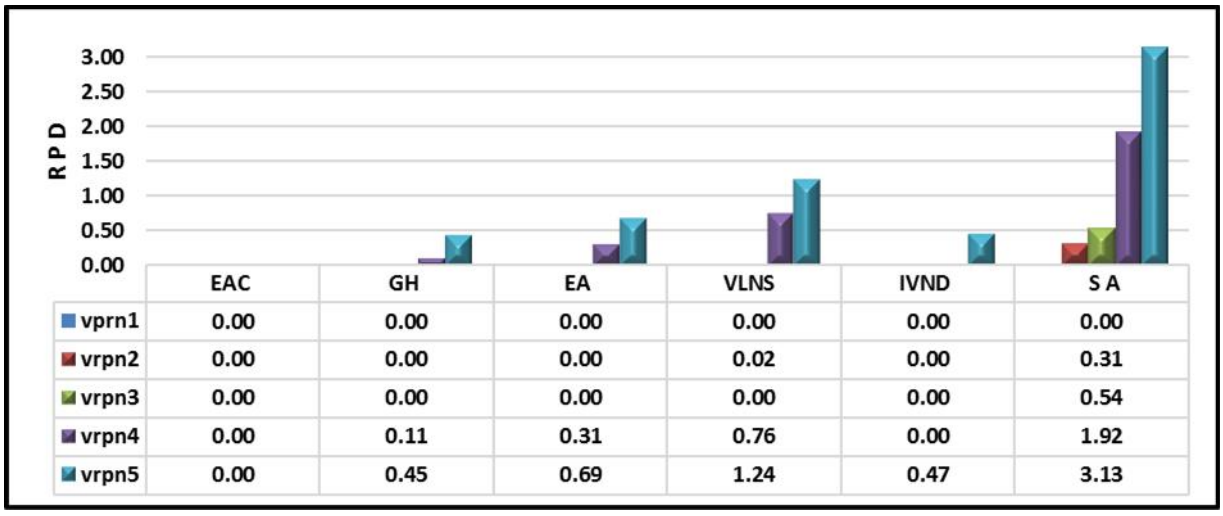
A-n39-k5	**39**	**822**	**822**	**822**	**0.00**	**0.00**	**822**
A-n39-k6	**39**	**831**	**835**	**833**	**0.31**	**0.00**	**831**
A-n44-k6	**44**	**939**	**938**	**938**	**0.00**	**0.10**	**937**
A-n45-k7	**45**	**1146**	**1147**	**1147**	**0.03**	**0.00**	**1146**
A-n53-k7	**53**	**1012**	**1013**	**1013**	**0.35**	**0.19**	**1010**
A-n54-k7	**54**	**1171**	**1171**	**1171**	**0.00**	**0.40**	**1167**
A-n55-k7	**55**	**1074**	**1074**	**1074**	**0.00**	**0.18**	**1073**
A-n60-k9	**60**	**1355**	**1360**	**1357**	**1.93**	**0.07**	**1354**
A-n62-k8	**62**	**1294**	**1297**	**1296**	**1.97**	**0.31**	**1290**
A-n65-k9	**65**	**1182**	**1188**	**1184**	**1.9**	**0.51**	**1174**
A-n80-k10	**80**	**1767**	**1773**	**1769**	**1.23**	**0.24**	**1763**

Tablas 16 y 17, muestran la comparación con los resultados obtenidos por otros investigadores para el CVRP, utilizando diferentes metaheurísticas para las instancias arriba mencionadas con respecto a la eficacia; estas metaheurísticas son the **PSOA** de (Marinakis, Marinaki, & Dounias, 2010); la **HGA** por (Berger & Barkaoui, A new hybrid genetic algorithm for the capacitated vehicle routing problems, 2004); la **GTS** por (Toth & Vigo, 2003); la **HST** por (Lin, Lee, Ying, & Lee, 2009); la **EAC** por (Nagata, 2007); la **GH** por (Pisinger & Ropke, 2007); la **EA** por (Prins, 2004); la **VLNS** por (Ergun, Orlin, & Steele-Feldman, 2006) y **IVND** por (Chen, Huang, & Dong, 2010) .

Los resultados son reportados en términos de RPD comparados con los mejores valores reportados.

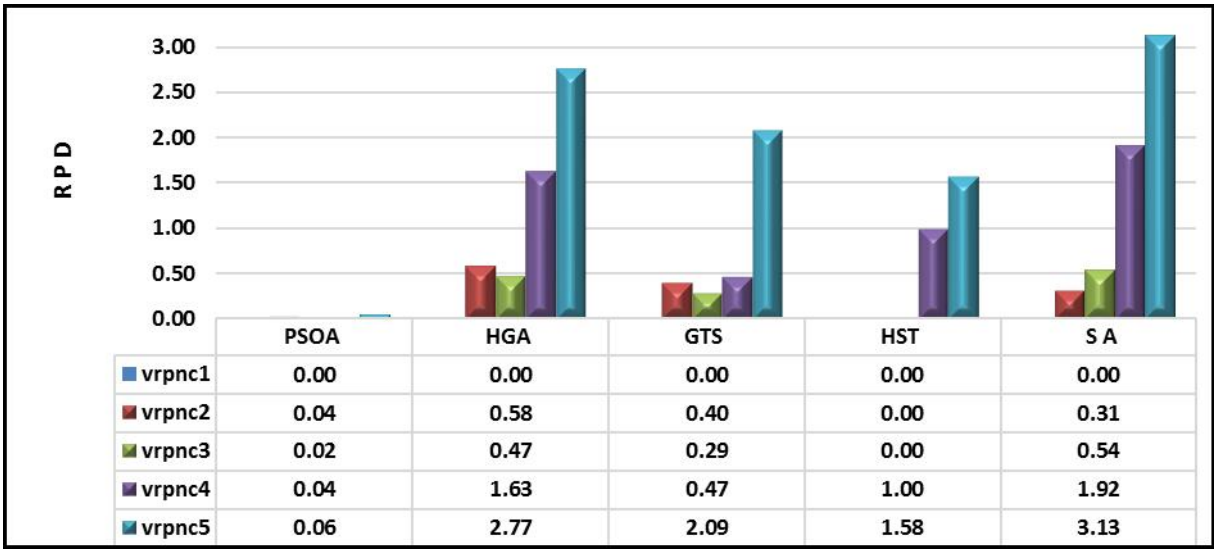
Como puede observarse en las Tablas 16 y 17, el algoritmo de recocido simulado programado para esta investigación presenta un desempeño satisfactorio, cabe mencionar que es una metaheurística pura, las otras metaheurísticas son híbridas y los resultados del recocido simulado no superan el 3.5% del margen de error.

Tabla 16. Algoritmo de recocido simulado comparado contra diferentes metaheurísticas para benchmarks de Christófides.



	EAC	GH	EA	VLNS	IVND	S A
vprn1	0.00	0.00	0.00	0.00	0.00	0.00
vrpn2	0.00	0.00	0.00	0.02	0.00	0.31
vrpn3	0.00	0.00	0.00	0.00	0.00	0.54
vrpn4	0.00	0.11	0.31	0.76	0.00	1.92
vrpn5	0.00	0.45	0.69	1.24	0.47	3.13

Tabla 17. Comparación entre el algoritmo de recocido simulado propuesto y diferentes heurísticas para benchmarks de Christófides.

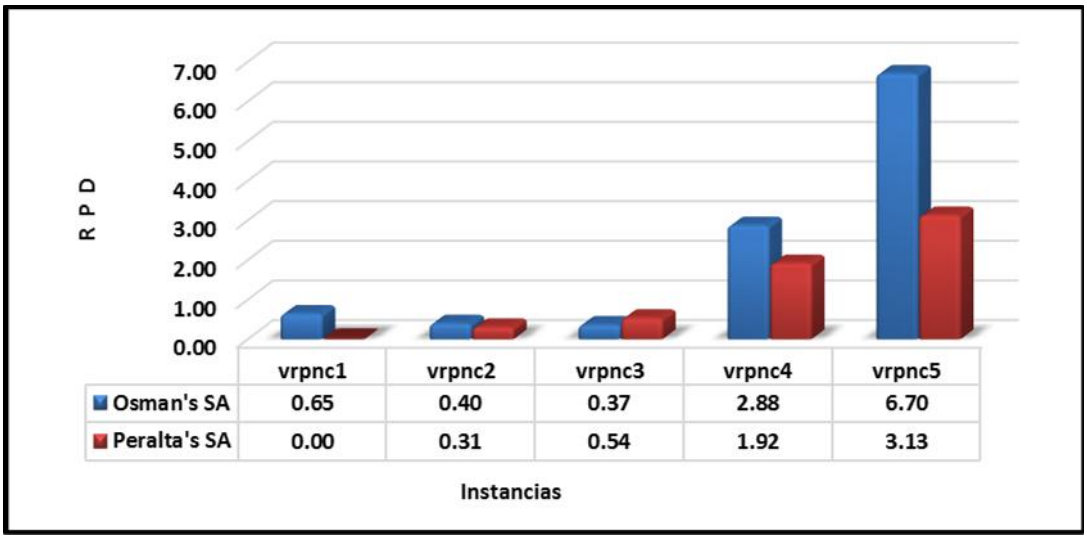


	PSOA	HGA	GTS	HST	S A
vrpnc1	0.00	0.00	0.00	0.00	0.00
vrpnc2	0.04	0.58	0.40	0.00	0.31
vrpnc3	0.02	0.47	0.29	0.00	0.54
vrpnc4	0.04	1.63	0.47	1.00	1.92
vrpnc5	0.06	2.77	2.09	1.58	3.13

Las Tablas 18, 19 y 20, muestran los resultados usando la misma metaheurística de recocido simulado en los benchmarks de Christófides y de Augerat, (Osman I. , 1993) (Harmanani, Azar, Helal, & Keirouz, 2011) .

La Tabla 18 muestra el desempeño de la metaheurística de Osman contra el algoritmo propuesto en esta investigación, en el 84% de las instancias el algoritmo propuesto es mejor.

Tabla 18. Algoritmo de Recocido Simulado comparado contra el de Osman en el benchmark de (Christofides, Mingozzi, & Toth, 1979)



En la Tabla 19 y 20 se muestra la comparación entre el algoritmo de recocido simulado desarrollado por Harmani y el propuesto con las instancias del benchmark de Augerat, puede observarse que también en el 94% de las instancias la metaheurística propuesta es mejor.

Tabla 19. Comparación del algoritmo de recocido simulado propuesto contra el de Harmani.

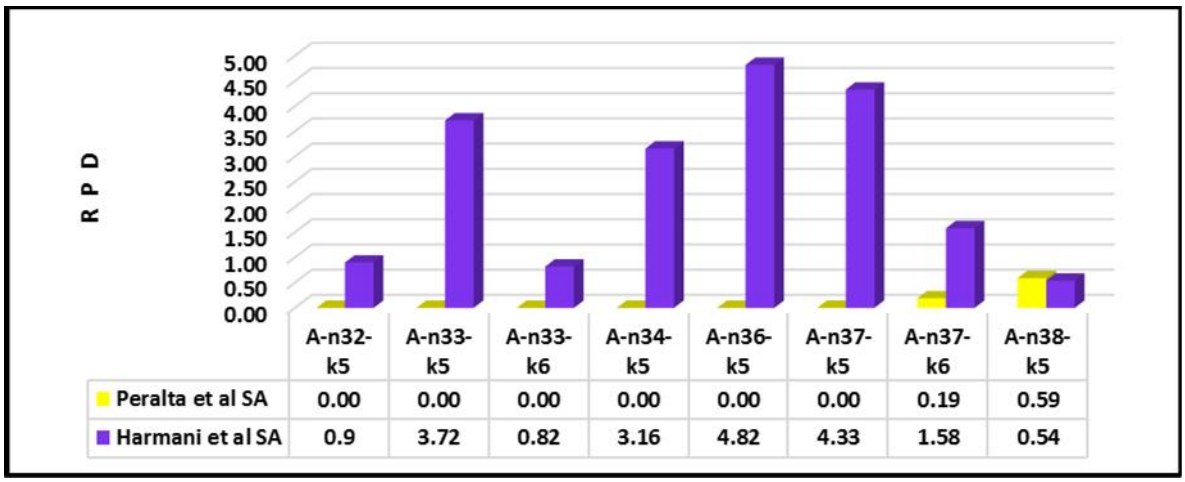
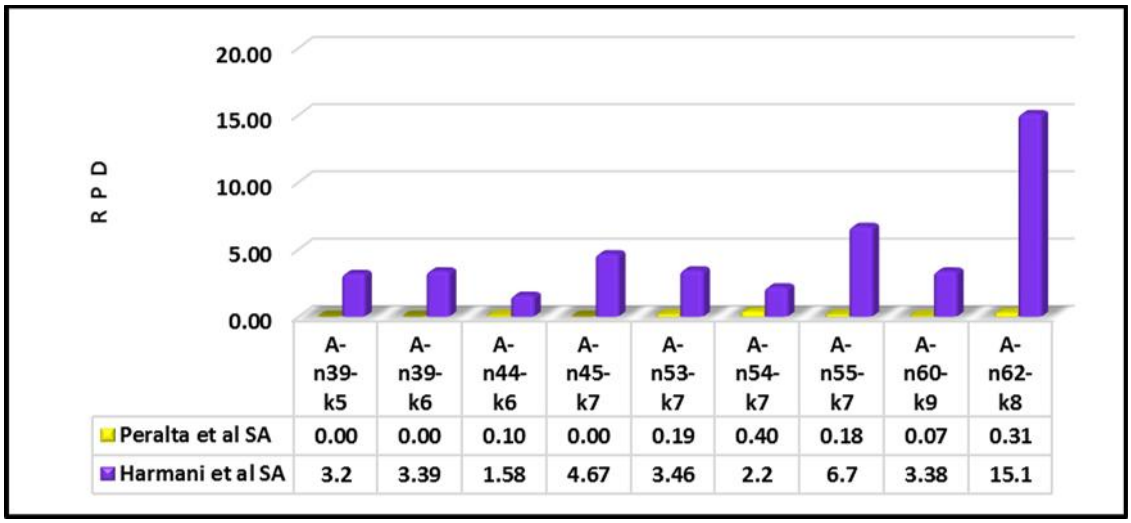


Tabla 20. Comparación del algoritmo de recocido simulado propuesto contra el de Harmani



	A-n39-k5	A-n39-k6	A-n44-k6	A-n45-k7	A-n53-k7	A-n54-k7	A-n55-k7	A-n60-k9	A-n62-k8
Peralta et al SA	0.00	0.00	0.10	0.00	0.19	0.40	0.18	0.07	0.31
Harmani et al SA	3.2	3.39	1.58	4.67	3.46	2.2	6.7	3.38	15.1

5.2.2 Análisis comparativo: estructura híbrida vs estructura simple

En este apartado, se presentan los resultados de varias ejecuciones del algoritmo de RS con reinicio para comparar el desempeño de la estructura de vecindario híbrida propuesta con una estructura de vecindario simple. Lo anterior con el fin de presentar que la estructura híbrida propuesta mejora el desempeño del algoritmo de RS.

Para poder realizar la comparación se trabajó el algoritmo de RS con una estructura de vecindad simple; el movimiento de perturbación que se usó fue el movimiento llamado swap, porque fue el que obtuvo los mejores resultados de acuerdo a la bibliografía consultada.

Las tablas 21, 22 y 23 muestran los mejores resultados obtenidos y su diferencia (RPD) con los mejores reportados en la literatura, al ejecutar el algoritmo SA.

Estos resultados muestran que la estructura híbrida propuesta con "solución dirigida" mejora considerablemente los resultados en RPD, Eq. (25), obtenido por SA con respecto a la cuota más conocida de la instancia.

Tabla 21. Resultados del algoritmo de RS con y sin estructura híbrida, instancias de Christófides

	RS (swap)	**RS (estructura híbrida)**
Instancia	**RPD %**	**RPD %**
Vrpn1	**0.65**	**0.00**
Vrpn2	**3.56**	**0.31**
Vrpn3	**4.23**	**0.54**
Vrpn4	**7.00**	**1.92**
Vrpn5	**8.92**	**3.13**

Tabla 22. Resultados del algoritmo de RS con y sin estructura híbrida, instancias de Taillard

	RS (swap)	**RS (estructura híbrida)**
Instancia	RPD %	RPD %
Tai75a	3.76	0.00
Tai75b	1.33	0.13
Tai75c	5.37	0.00
Tai75d	4.35	0.00
Tai100a	1.36	1.00
Tai100b	4.83	0.13
Tai100c	3.12	0.20
Tai100d	5.62	0.11
Tai150a	5.17	0.50
Tai150b	10.19	0.53
Tai150c	5.75	0.37

Tabla 23. Resultados del algoritmo de RS con y sin estructura híbrida en instancias de Augerat

	RS (swap)	**RS (estructura híbrida)**
Instancia	RPD %	RPD %
A-n32-k5	4.79	0.00
A-n33-k5	4.14	0.00
A-n33-k6	1.68	0.00
A-n34-k5	1.21	0.00
A-n36-k5	1.51	0.00
A-n37-k5	0.87	0.00
A-n37-k6	0.98	0.19
A-n38-k5	3.58	0.59
A-n39-k5	1.01	0.00
A-n39-k6	3.03	0.00
A-n44-k6	1.72	0.10
A-n45-k7	1.78	0.00
A-n53-k7	2.83	0.19
A-n54-k7	4.67	0.40
A-n55-k9	2.53	0.18
A-n60-k9	1.27	0.07
A-n62-k8	2.51	0.31
A-n65-k9	4.29	0.51
A-n80-k10	2.36	0.24

5.2.3 Análisis de Eficiencia

La eficiencia analiza el volumen de recursos utilizados para lograr los objetivos planteados, un algoritmo eficiente hace un uso óptimo de los recursos para que tenga un menor costo posible, también, la eficiencia de un algoritmo es una medida de los recursos que consume a la hora de ejecutarse en función del tamaño de las entradas.

El conocer la eficiencia del algoritmo permite determinar la cantidad de recursos mínimos que requiere para su ejecución. Si requiere de gran cantidad de recursos se considera que no es eficiente y su aplicación no sería la adecuada cuando se tienen limitados los recursos.

Los recursos computacionales se dividen en:

- tiempo de ejecución (temporales), es el tiempo el cual el algoritmo requiere para ejecutar las instrucciones que lo componen.
- memoria ocupada (espaciales), es la cantidad de recursos necesarios para almacenar los datos que requiere el algoritmo.

En esta sección, solo se harán evaluaciones del algoritmo con base al tiempo de ejecución ocupado para cada una de las instancias trabajadas en este proyecto.

Las pruebas experimentales se realizaron en el clúster Cuexcomate (sección 6.1), se realizaron 30 ejecuciones que son lás mínimas sugeridas por (Alba E. , 2005), si la muestra es menor las variaciones no podrán ser observadas para cada una de las instancias. (Calidad, 1992).

El tiempo que tarda en ejecutarse el algoritmo para cada instancia define su eficiencia respecto a los recursos temporales.

Es necesario recalcar que las comparaciones realizadas no pueden ser consideradas en los mismos términos de eficiencia, no hay igualdad ni en el algoritmo, heurística

aplicada o sistema de cómputo, los algoritmos mostrados fueron ejecutados en condiciones muy diferentes al de esta investigación.

En la Tabla 21, se muestra el tiempo de ejecución en segundos para las instancias de (Christofides, Mingozzi, & Toth, 1979) con diferentes heurísticas. No se pudo realizar una comparación con la misma heurística porque no hay investigaciones que hayan trabajado solo con recocido simulado.

Los datos del tiempo fueron obtenidos de las metaheurísticas de **GH** por (Pisinger & Ropke, 2007); **HybPSOA** de (Marinakis, Marinaki, & Dounias, 2010); **AGES-VRP** de (Mester & Bräysy, 2005); **ILS-RVND-SP** de (Subraminan, Uchoa, & Satoru, 2013) y **HGSADC** de (Vidal, Crainic, Gendreau, Lahrichi, & Rei, 2012).

En la Tablas 22 y 23, se presentan los tiempos de ejecución en segundos, para las instancias de Augerat y Taillard. También los datos fueron obtenidos de las metaheurísticas: New branch-and-cut algorithm de (Lysgaard, Letchford, & Eglese, 2004) para la tabla 22 y de (Rochat & Taillard, 1995) para la tabla 23.

Tabla 24. Promedio de los tiempos de ejecución, en segundos, de diferentes heurísticas para el benchmark de Christofides

Instancia	# clientes	GH	AGES VRP	HGSADC	ILS-RVND-SP	HybPSO	SA
vrpnc1	**50**	**21**	**0.20**	**25.80**	**1.48**	**3.00**	**99.33**
vrpnc2	**75**	**36**	**5.50**	**57.60**	**13.52**	**12.60**	**6005.69**
vrpnc3	**100**	**78**	**1.00**	**76.20**	**12.49**	**19.20**	**8896.19**
vrpnc4	**150**	**160**	**10.20**	**172.20**	**53.48**	**60.60**	**10996.74**
vrpnc5	**199**	**219**	**2160.00**	**356.40**	**625.17**	**129.00**	**13822.47**

Tabla 25. Comparación de eficiencia (tiempos de ejecución en segundos) con el benchmark de Augerat.

Instancia	New branch and cut algoritihm	SA
A-n32-k5	12	408.24
A-n33-k5	12	408.11
A-n33-k6	11	455.10
A-n34-k5	11	425.52
A-n36-k5	10	555.09
A-n37-k5	11	16.00
A-n37-k6	12	754.30
A-n38-k5	7	763.16
A-n39-k5	39	767.56
A-n39-k6	13	775.16
A-n44-k6	31	1228.42
A-n45-k7	66	122.92
A-n53-k7	24	2025.20
A-n54-k7	30	1747.78
A-n55-k7	17	1755.84
A-n60-k9	63	2193.54
A-n62-k8	231	2197.34

Tabla 26. Tiempos de ejecución para Benchmark de Taillard y SA.

Instancia	# Clientes	Gendreau et al T S	S A
Tai75a	75	1900	3500
Tai75b	75	1700	3900
Tai75c	75	1600	650
Tai75d	75	1700	1775
Tai100a	100	3600	5900
Tai100b	100	2900	4800
Tai100c	100	2900	3270
Tai100d	100	3300	4800
Tai150a	150	7000	30000
Tai150b	150	8600	17500
Tai150c	150	6400	10500

Analizando las Tablas 21, 22 y 23, se observa que el algoritmo propuesto no tiene un buen desempeño en su eficiencia. Una queja común de esta metaheurística es su lenta convergencia (Ram, Sreenivas, & Subramaniam, 1996) y que necesita mucho tiempo de cálculo cuando los problemas a resolver son grandes, su tiempo de cálculo está directamente relacionado con el tamaño del espacio de soluciones (Díaz & Dowsland, 2003).

Lo anterior es debido a que la metaheurística de recocido simulado para poder llegar a una solución que esté a una distancia próxima a la óptima necesita tiempo de computación de tipo exponencial (Van Laarhoven, Aarts, & Korst, 1997), resultados teóricos basados en la teoría de las cadenas de Markov han demostrado que con un programa de enfriamiento infinitamente lento, el algoritmo de recocido convergería al óptimo global con probabilidad 1, cuando la temperatura tienda a 0 (Aarts & Korst, 1989), aunque no se pueda garantizar esa convergencia con programas de enfriamiento infinitamente lento.

El algoritmo propuesto de recocido simulado, fue probado en tres diferentes benchmarks que contienen una amplia variedad de casos para el problema del CVRP. Los resultados computacionales y estudio comparativo son muy alentadores: el algoritmo propuesto muestra un buen desempeño comparado con los trabajos de Osman y Harmanni y que con otras heurísticas su desempeño no es malo en muchos casos, su requisito de tiempo de cálculo también es razonable para problemas de tamaño real.

Cuando el número de clientes se incrementó, mayor fue el tiempo de ejecución debido a diversos factores como lo son: el tamaño de la instancia, la forma de distribuir los datos en la estructura de vecindad, el espacio de búsqueda generado.

Nuestros tiempos de cálculo son más largos que los de otros enfoques, pueden ser debido al proceso de codificación realizado. Sin embargo, la intención principal de esta comparación es para demostrar que, con respecto a la calidad de la solución, nuestro enfoque puede producir resultados comparables a los de los mejores métodos para el desempeño de CVRP reportados en la literatura.

Como conclusión de este apartado, los resultados comparativos del desempeño de nuestro algoritmo con otros algoritmos, indican que la metaheurística propuesta de recocido simulado, es capaz, resolver con eficacia diversas instancias del CVRP dentro de una cantidad razonable de tiempo. Además el tiempo contabilizado es el total ejecutado por los parámetros de control, la solución se pudo haber conseguido antes de terminar la ejecución.

Aunque el tiempo de CPU requerido es más largo que otros enfoques, sigue siendo razonable para un problema estratégico que no necesita ser resuelto todos los días. Además, para obtener la aplicación de una nueva red logística que será operada por años, vale la pena pasar unos minutos más de tiempo de computadora, siempre y cuando la solución sea competitiva.

5.3 Análisis de desempeño del algoritmo distribuido

En este apartado se muestran los resultados obtenidos en la ejecución del algoritmo distribuido del recocido simulado. Para la experimentación en paralelo se trabajaron las instancias de M-n200-17, X-n209-k16, X.n233-k16 y X-n261-k13; que son instancias cuyos clientes varían de 200 a 260, a las cuales no se les conoce actualmente solución óptima pero la calidad de los resultados obtenido se medirá contra las mejoras cotas reportadas en la literatura. Las instancias X-n pertenecen al grupo del nuevo benchmark propuesto por (Uchoa et al, 2014), este conjunto consta de 100 instancias, las primeras 50 van de 100 a 330 clientes, las otras 50 van de 335 a 1000 clientes. Los depósitos están distribuidos de 3 formas: Centrados (C), En una esquina (E) o Aleatoriamente (R).

5.3.1 Análisis de Eficacia

En el apartado 5.2.1 se menciona que la eficacia de un algoritmo está sujeto a la calidad de las soluciones obtenidas. Recocido simulado tiene el inconveniente de tener muy lenta convergencia y de ser inherentemente secuencial, estas características no le permiten obtener soluciones con la calidad deseada.

En un intento por lograr la calidad en sus resultados se han dado muchos experimentos para desarrollarle una versión paralela que además le permita obtener un speedup (visto en sección 4.6.2) cercano al ideal, punto importante a considerar en la paralelización.

La metodología de trabajo fue la siguiente; el proyecto se desarrolló en el cluster Cuexcomate, el nodo maestro es el Cuexcomate los nodos esclavos utilizados fueron ciicap01, ciicap02, ciicap03 y ciicap04.

Para cada instancia se realizaron 30 pruebas y la ejecución de los procesos fueron distribuidos de manera balanceada como se muestra en la Tabla 24, esto es importante porque si un nodo trabaja con menos procesos la eficiencia del algoritmo

sería mayor por tener que esperar a que terminen los nodos con carga de trabajo mayor. Las instrucciones de trabajo para la distribución de la carga fue la siguiente:

mpiexec.hydra -ppn <**nodo**> -n <**número de procesos**> -hostfile <**archivo nombre nodos**> ./<archivo a ejecutar>

Tabla 27. Distribución de procesos en nodos Cuexcomate

Total de procesos	# de Procesos por nodo
4	1
8	2
16	4
32	8
64	16

En las figuras 45, 46,47, 48 y 49, se muestra la distribución arriba mostrada.

Figura 45. Ejecución de 4 procesos

Figura 46. Ejecución de 8 procesos

Figura 47. Ejecución de 16 procesos

Figura 48. Ejecución de 32 procesos

Figura 49. Ejecución de 64 procesos

Los colores de las figuras 45 a 49 indican el porcentaje de uso del clúster, azul de 0 a 25%, verde de 25 a 50% , amarillo de 50 a 75% y naranja del 75 al 100%.

En la Tabla 25 se muestran los resultados obtenidos por instancia y por proceso obtenidos al ejecutar el algoritmo distribuido de recocido simulado. Los resultados son reportados en términos de RPD (%) comparados con los mejores valores reportados.

Tabla 28. Análisis de eficacia en % RPD.

Mejor valor reportado en literatura	**1275**	**30656**	**19230**	**26558**
Procesos	**M-n200-17**	**X-n209-k16**	**X-n233-k16**	**X-n261-k13**
1	**10.06%**	**8.42%**	**9.89%**	**10.12%**
4	**8.59%**	**7.93%**	**8.87%**	**9.31%**
8	**5.74%**	**7.01%**	**3.59%**	**5.66%**
16	**5.16%**	**4.52%**	**3.35%**	**5.06%**
32	**4.82%**	**3.62%**	**2.68%**	**4.44%**
64	**4.19%**	**2.44%**	**2.46%**	**4.04%**

De la tabla anterior se observa que a mayor número de procesos la solución encontrada mejora, debido a que se amplía el espacio de búsqueda por la cantidad de vecinos que se generan por cada solución inicial con la que arranca cada nodo en la ejecución del algoritmo.

Con un proceso solo se ocupa una solución inicial, con 4, cuatro soluciones iniciales, con 8, ocho soluciones iniciales y así sucesivamente.

El comportamiento de los resultados se puede observar mejor en las gráficas de la Figura 50.

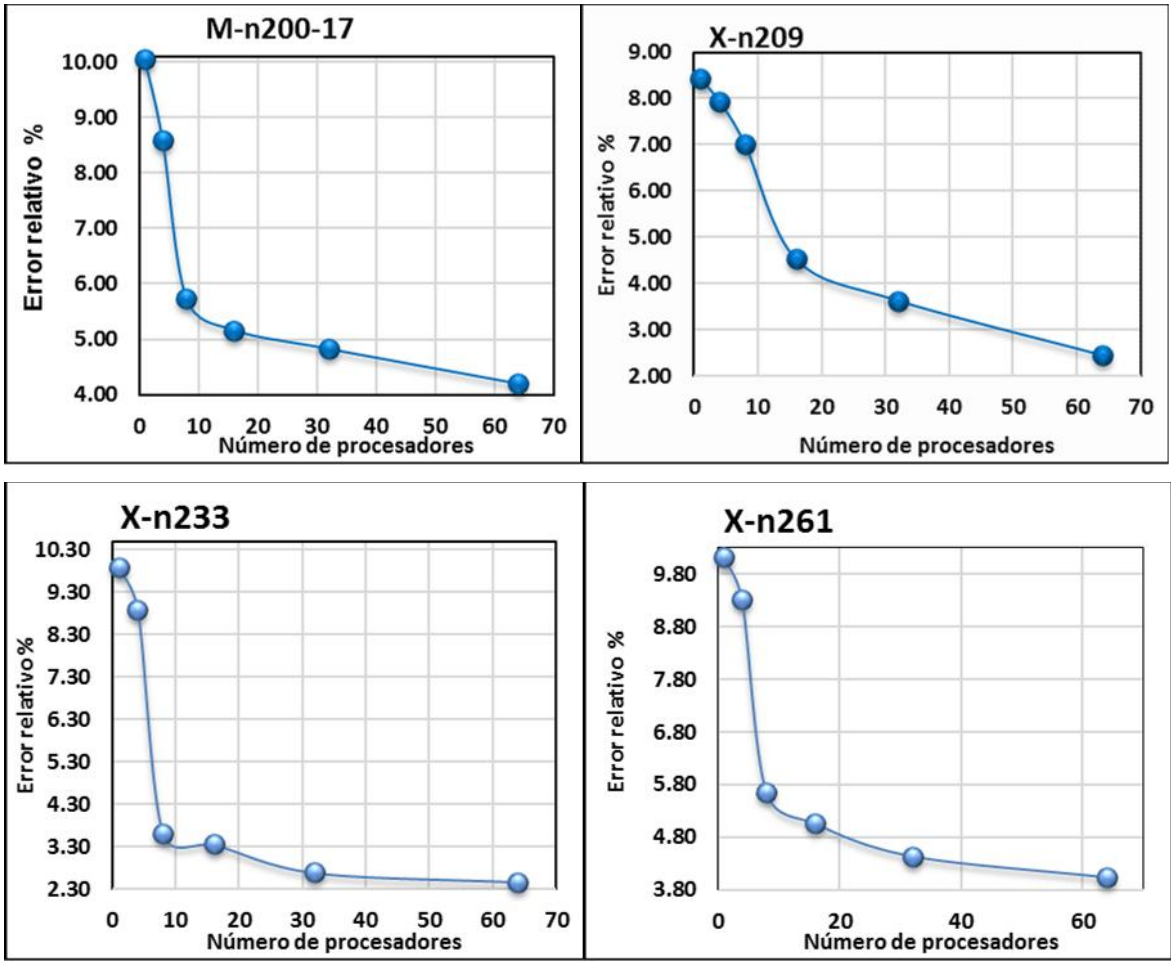


Figura 50. Gráficas de error relativo vs número de procesadores.

De los resultados generados por el algoritmo de recocido simulado distribuido para esta investigación se puede determinar que presenta un desempeño aceptable, el

porcentaje de error relativo varía de 4.19% a 10.06% para M-n200-17, de 2.44% a 7.93% para X-n209-k16, 2.46% a 9.89% para X-n233-k16 y de 4.04% a 10.12% para X-n262-k13; con procesadores de 4 a 64 respectivamente.

En la tabla 29, se muestran los resultados de la ejecución del algoritmo de forma secuencial y paralela. El ejecutar el algoritmo de forma paralela ayudó a diversificar la búsqueda en el espacio de soluciones lo que permite una mejor exploración obteniendo mejores resultados.

Tabla 29. Resultados obtenidos por los algoritmos secuencial y paralelo

Instancia	Resultado Secuencial	Resultado Distribuido
M-n200-17	4.85%	4.19%
X-n209-k16	3.42%	2.44%;
X-n233-k16	2.83%	2.46%
X-n261-k13	5.45%	4.04%

Es necesario indicar que los programas paralelos son más difíciles de escribir que los secuenciales porque la concurrencia introduce nuevos tipos de errores de software. La comunicación y sincronización entre diferentes tareas son algunos de los mayores obstáculos para obtener un buen rendimiento del programa paralelo (Hennessy, Patterson, & Larus, 1999).

La programación en paralelo, utiliza simultáneamente múltiples elementos de procesamiento para resolver un problema. Esto se logra mediante la división del problema en partes independientes de modo que cada elemento de procesamiento pueda ejecutar su parte del algoritmo de manera simultánea con los otros pero el algoritmo de recocido simulado, como ya se mencionó anteriormente es de difícil paralelización porque no se puede dividir por su carácter altamente secuencial y las opciones de paralelización son muy reducidas generando una ejecución secuencial en cada nodo. Aun así se puede observar que, en todas las instancias, los mejores cálculos se logran al incrementar el número de nodos, utilizando únicamente un nodo (versión uniprocesador) el rendimiento no es bueno.

5.3.2 Análisis de eficiencia (*Speed-up*).

La eficiencia de un sistema paralelo se puede realizar a través de medir el tiempo de ejecución.

Existen parámetros que afectan directamente la eficiencia (rendimiento) que se puede esperar del clúster:

1. ***Tamaño del problema***, en este caso las dimensiones de las instancias; y
2. ***Número de elementos de procesamiento***, o nodos, que participan en el cálculo.
3. ***Operaciones secuenciales***, instrucciones que requieren ser ejecutados de forma secuencial.
4. ***Procesos distribuidos***, segmentos de código que pueden ser ejecutados de forma paralela.
5. ***Latencia***, que es el tiempo requerido para realizar la distribución y comunicación ente los procesos.

Para poder observar el comportamiento del *clúster*, se variaron los dos primeros parámetros (dimensión y número), realizando pruebas con diversos tamaños de instancias, variando el número de nodos, y tomando el tiempo empleado en la realización de los cálculos. Estos tiempos se comparan y grafican para permitir un análisis visual de los resultados. Para cada instancia se tomaron tiempos de solución desde un procesador hasta cuatro (ciicap01, ciicap02, ciicap03 y ciicap04).

Una forma de evaluar el desempeño del algoritmo distribuido es a través de su rendimiento, mismo que permite medir su tiempo de ejecución evaluando su comportamiento al incrementar la cantidad de procesadores. Los algoritmos paralelos se diseñan para que su ejecución sea mucho más rápida que el secuencial.

Idealmente, la aceleración a partir de la paralelización es lineal, duplicar el número de nodos de procesamiento debe reducir a la mitad el tiempo de ejecución y duplicar por segunda vez debe nuevamente reducir el tiempo a la mitad, así se puede decidir

si a mayor número de núcleos se podrá obtener una mejora en el rendimiento del algoritmo.

Para la eficiencia se va a comparar que tan cercana se encuentra la medida de aceleración del algoritmo distribuido con el speed-up.

La ficiencia de un sistema para *n* procesadores esta definida por la cuación (24)

$$E_{(n)} = \frac{S_{(n)}}{n} \tag{24}$$

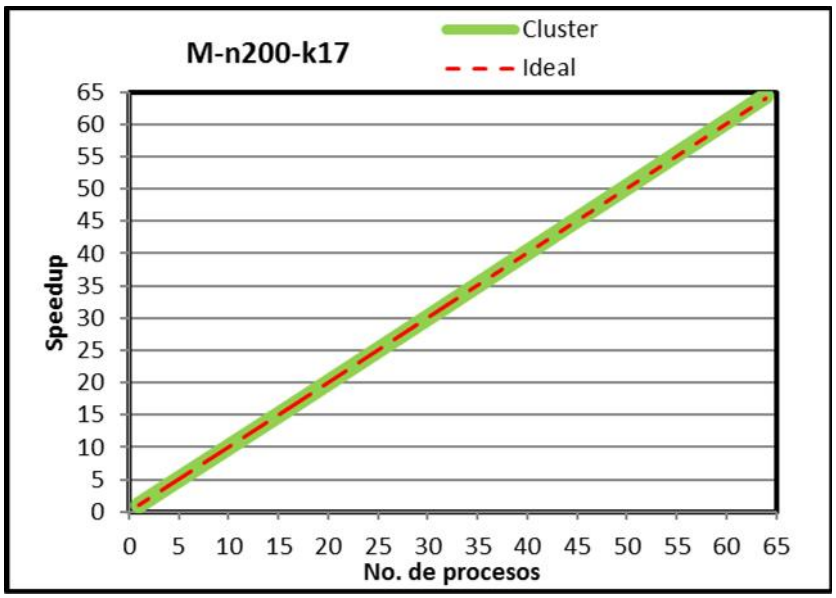
Además la eficiencia es una comparación del grado de speedup conseguido frente al valor máximo. Dado que $\frac{1}{n} \leq E_{(n)} \leq 1$. La eficiencia más baja $E_{(n)} \to 0$ corresponde al caso cuando todo el programa se ejecuta en un solo procesador de forma secuencial, si $E_{(n)} \to 1$ es la eficiencia máxima, se obtiene cuando todos los procesadores están siendo utilizados durante el tiempo de ejecución del algoritmo. Entre mayor sea la parte paralelizable de un algoritmo, el rendimiento a un sistema paralelo será mejor.

La fórmula usada para calcular el speed-up es la siguiente:

$$Speed\ up = \frac{Tiempo\ de\ ejecución\ del\ algoritmo\ secuencial\ más\ eficiente}{Tiempo\ de\ ejecución\ del\ algoritmo\ en\ paralelo}$$

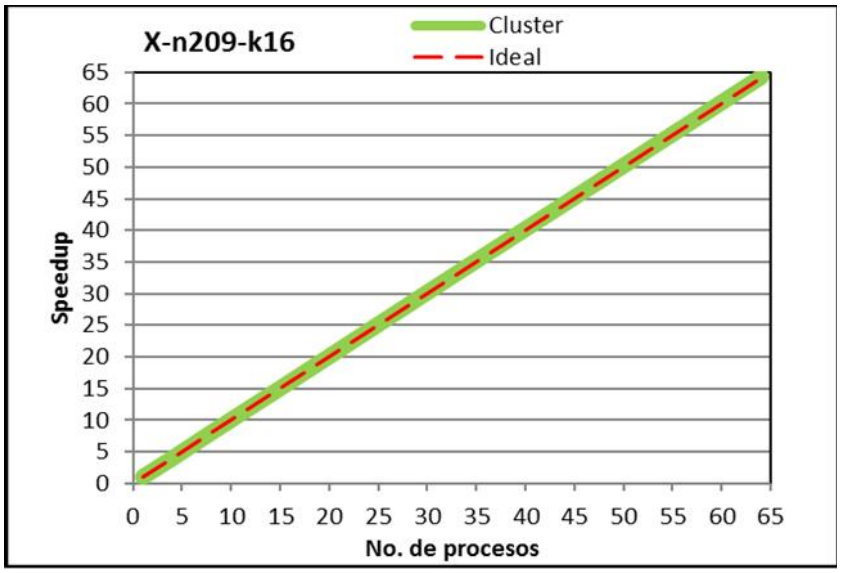
En las Figuras 51, 52, 53 y 54, se muestran las gráficas con los resultados de speed-up (el ideal y el real) obtenido a partir de los resultados, se muestra una acelaración sublineal con una eficiencia promedio calculada de la utilización de los proceadores del 99.037%, el 0.963% restante, representa la parte que no se puede paralelizar. Al utilizar un número mayor de procesos la eficiencia va ligeramente mejorando, no se ve afectado por las comunicaciones entre los procesos porque el nodo maestro hace un solo envío de la información y se recibe un solo envío de los nodos esclavos.

Un mayor número de procesos mejora el rendimiento del algoritmo en el caso del algoritmo distribuido con reinicio de reocido simulado.



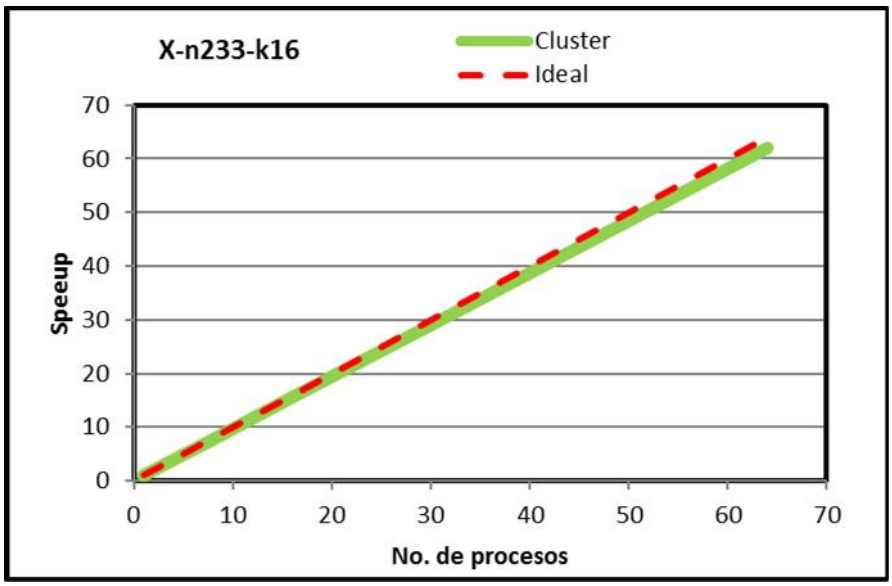
Procesos	Speedup
1	1
4	3.95
8	7.92
16	15.91
32	31.99
64	64.35

Figura 51. Comportamiento del Speedup M-n200-k17



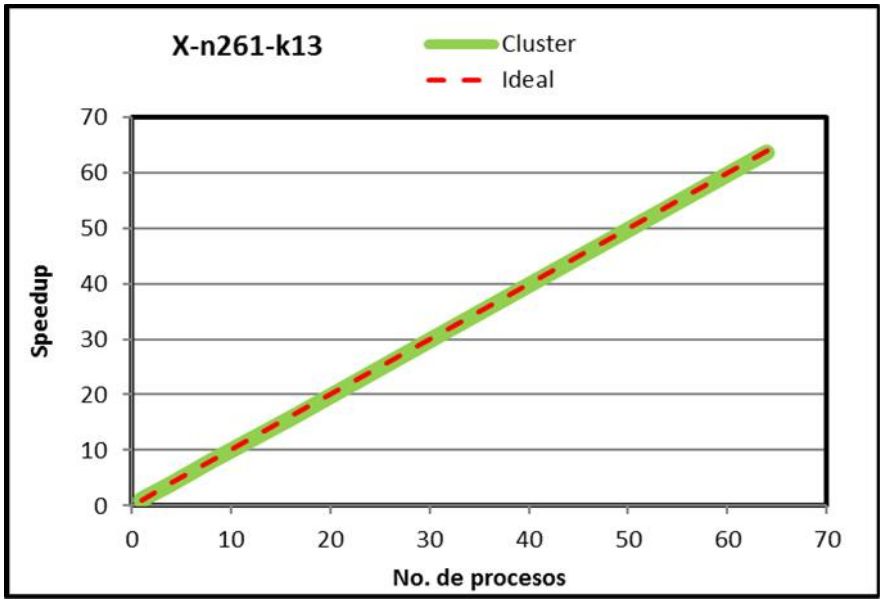
Procesos	Speedup
1	1
4	3.99
8	8.00
16	16.01
32	32.11
64	64.27

Figura 52. Comportamiento del Speed up X-n209-k16



Procesos	Speedup
1	1
4	3.86
8	7.71
16	15.46
32	30.99
64	62.04

Figura 53. Comportamiento del Speed up X-n233-k16.



Procesos	**Speedup**
1	**1**
4	**3.94**
8	**7.93**
16	**15.86**
32	**31.74**
64	**63.60**

Figura 54. Comportamiento el Speed up instancia X-n261-k13.

La Tablas 27 a 30, muestran el análisis de la aceleración (speed-up) y eficacia del algoritmo y su ejecución con varias intancias. Así, el speed-up ($S_{(n)}$) se usa para indicar el grado de ganancia de velocidad de una programación distribuida, la

eficiencia $(E_{(n)})$ mide la porción útil del trabajo total realizado por *n* procesadores. El speed up es de tipo lineal.

Las tablas se interpretan de la forma siguiente; la columna 1 muestra el número de procesos trabajados, la columna 2 muestra el tiempo de ejecución secuencial, la columna 3 la media de tiempo de 30 ejecuciones al utilizar diferente número de procesos, la columna 4 muestra el cálculo de la aceleración (Speed-up), la columna 5 muestra la eficiencia del uso de los procesadores, finalmente la columna 6 muestra la aceleración ideal.

Tabla 30. Tiempos de ejecución (segundos) del algoritmo distribuido de la instancia M-n200-k17.

1	2	3	4	5	6
# ***Procesos***	Tiempo ejecución secuencial	Tiempo ejecución paralelo	Spedd-up $S_{(n)}$	Eficiencia $E_{(n)}$	Speed-up Ideal
1	1403.25		1	1	1
4		350.81	3.95	0.987	4
8		175.41	7.92	0.990	8
16		87.70	15.91	0.994	16
32		43.85	31.99	1.000	32
64		21.93	64.35	1.000	64

Tabla 31. Tiempos de ejecución (segundos) del algoritmo distribuido de la instancia X-n209-k16.

1	2	3	4	5	6
# ***Procesos***	Tiempo ejecución secuencial	Tiempo ejecución paralelo	Spedd-up $S_{(n)}$	Eficiencia $E_{(n)}$	Speed-up Ideal
1	33238.08		1	1	1
4		8309.52	3.99	0.998	4
8		4154.76	8.00	1.000	8
16		2077.38	16.01	1.001	16
32		1038.69	32.09	1.003	32
64		519.34	64.27	1.004	64

Tabla 32. Tiempos de ejecución (segundos) del algoritmo distribuido de la instancia X-n233-k16.

1	2	3	4	5	6
# ***Procesos***	Tiempo ejecución secuencial	Tiempo ejecución paralelo	Spedd-up $S_{(n)}$	Eficiencia $E_{(n)}$	Speed-up Ideal
1	20943.47		1	1	1
4		5235.87	3.86	0.964	4
8		2617.93	7.71	0.964	8
16		1308.97	15.46	0.966	16
32		654.48	30.99	0.969	32
64		327.24	62.04	0.969	64

Tabla 33. Tiempos de ejecución (segundos) del algoritmo distribuido de la instancia X-n261-k13.

1	2	3	4	5	6
# ***Procesos***	Tiempo ejecución secuencial	Tiempo ejecución paralelo	Spedd-up $S_{(n)}$	Eficiencia $E_{(n)}$	Speed-up Ideal
1	29777.50		1	1	1
4		7444.38	3.94	0.986	4
8		3722.19	7.92	0.989	8
16		1861.09	15.85	0.991	16
32		930.55	31.72	0.992	32
64		465.27	63.60	0.994	64

Analizando las Tablas 30 a 33 y la Figura 51, se muestra una aceleración con una eficiencia promedio calculada del 99.037%, es decir que la parelización hace eficiente el uso de los procesadores, el porcentaje restante 0.963% representa la parte que no se puede paralelizar.

El algoritmo propuesto presenta un buen desempeño y muestra una ligera caída al utilizar un número mayor de procesadores, si el algoritmo conserva esta eficiencia se espera que el algoritmo pueda escalar linealmente en el orden del número de procesos aumentados.

Las gráficas de los mejores resultados obtenidos por instancia se muestran en las gráficas de las Figuras 55, 56, 57 y 58. Puede observarse en la mayoría de las rutas generadas por cada instancia que están bien definidas por un contorno que no muestra cruces entre los clientes de la misma ruta.

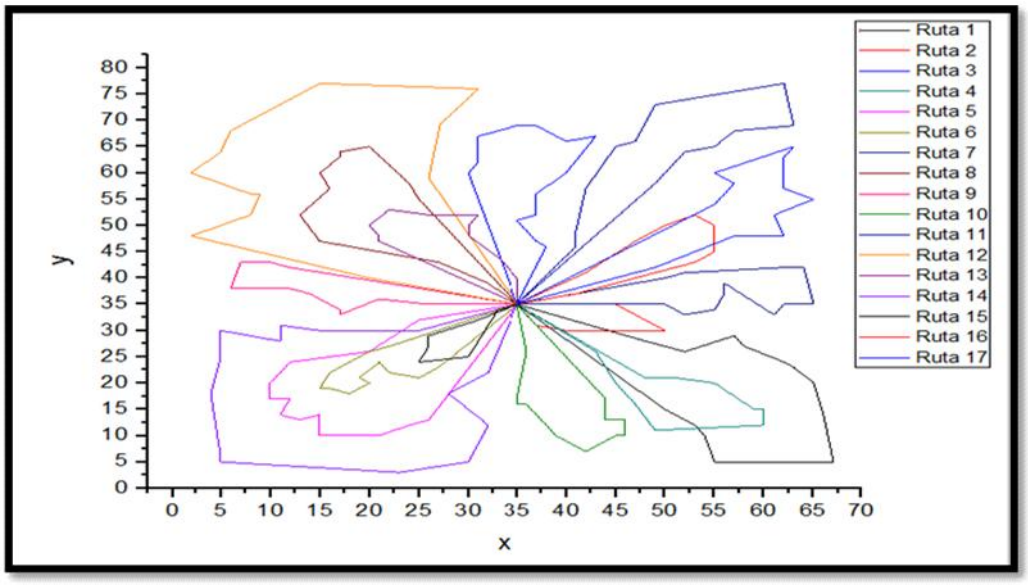


Figura 55. Ruteo solución de la instancia M-n200-17-k17.

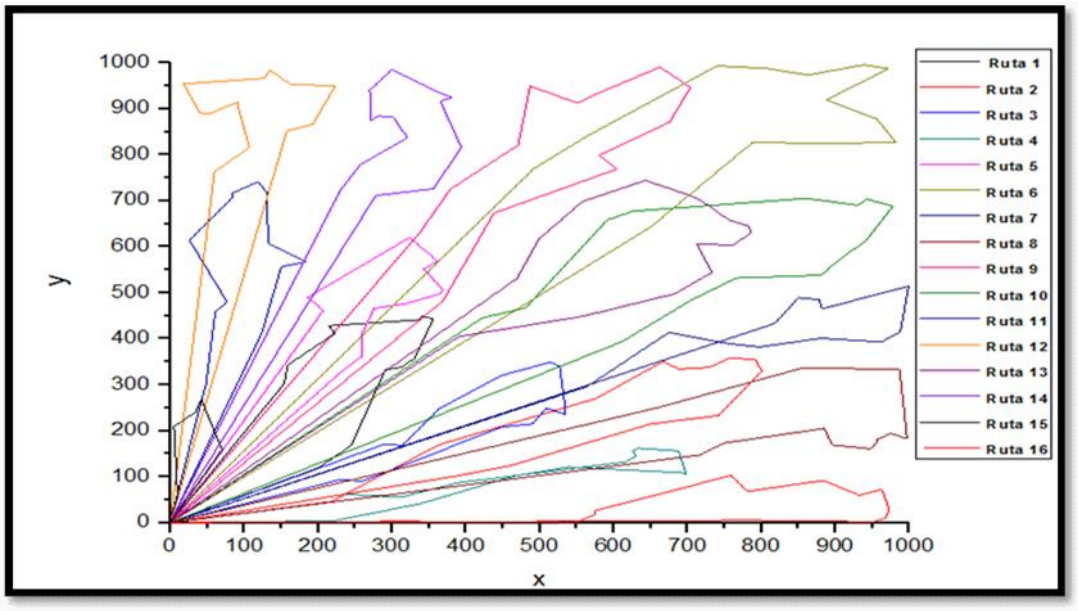


Figura 56. Ruteo solución de la instancia X-n209-k16.

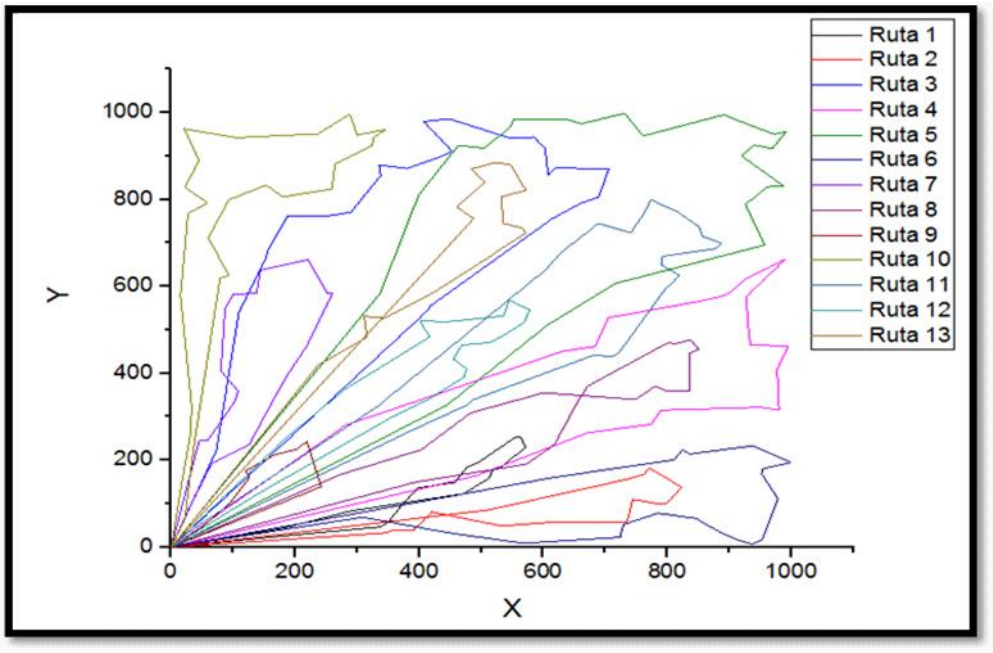


Figura 57. Ruteo solución de la instancia X-n233-k17.

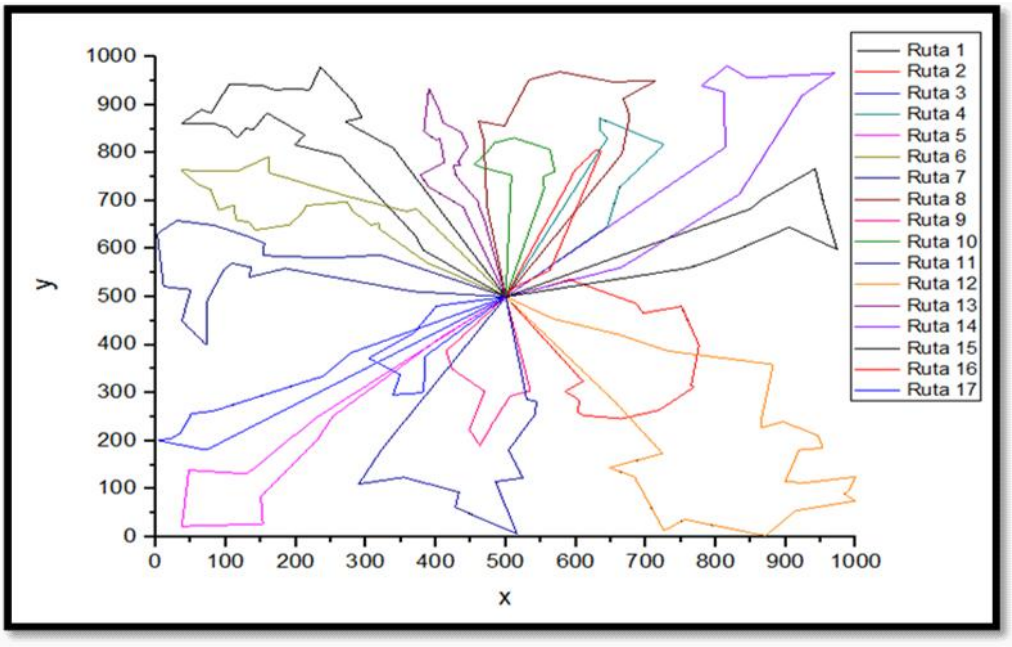


Figura 58. Ruteo solución de la instancia X-n261-k13.

CAPÍTULO 6 CONCLUSIONES Y TRABAJOS FUTUROS

6.1 Conclusiones

El trabajo realizado en esta tesis doctoral fue el de desarrollar un algoritmo con distribución de procesos para el problema de Ruteo Vehicular con Capacidades Homogéneas, que se ejecutó en un cluster computacional. La metaheurística con la que se trabajó fue Recocido Simulado que está clasificada como de tipo trayectorial; este tipo de metaheurísticas realizan una búsqueda que parte de una solución inical y van generando un camino o trayectoria en el espacio de soluciones a través de movimientos (perturbaciones).

Se desarrollaron dos versiones del algoritmo: secuencial y distribuido.

Del secuencial se puede concluir lo siguiente:

- Derivado del análisis realizado, se pudo comprobar que el trabajar con la metaheurística de recocido simulado de manera pura, no permite obtener soluciones aceptables, es necesario adicionar una estrategia que le permita mejorar su desempeño.

- Para mejorar su desempeño, se le agregó un reinicio y una estructura híbrida de vecindad. Esta estructura híbrida, tiene tres heurísticas que permiten hacer movimientos (perturbaciones) de la solución inicial. Los movimientos generados conforman el espacio de soluciones a evaluar dentro del algoritmo de Metrópolis.

- La estructura híbrida de vecindad, incluye un movimiento identificado como la función de cambiaVecino, este movimiento realiza una búsqueda en el espacio de soluciones de manera guiada en lugar de hacerlo de manera aleatoria, la cual mejoró de manera notable la eficacia de los resultados obtenidos en el algoritmo secuencial.

- Las pruebas experimentales obtenidas de la ejecución del algoritmo secuencial, presentaron una eficacia competitiva, los resultados obtenidos lograron en algunas instancias al igualar el resultado al mejor reportado en la literatura comparado con investigaciones que también aplicaron recocido simulado.

- En la comparación de resultados con otras metaheurísticas (PSOA de (Marinakis, Marinaki, & Dounias, 2010); la HGA por (Berger & Barkaoui, A new hybrid genetic algorithm for the capacitated vehicle routing problems, 2004); la GTS por (Toth & Vigo, 2003); la HST por (Lin, Lee, Ying, & Lee, 2009); la EAC por (Nagata, 2007); la GH por (Pisinger & Ropke, 2007); la EA por (Prins, 2004); la VLNS por (Ergun, Orlin, & Steele-Feldman, 2006) y IVND por (Chen, Huang, & Dong, 2010), el recocido simulado aplicado en este trabajo tiene una diferencia con el óptimo que no supera el 3.5%.

- En cuanto a la eficiencia, los resultados confirman el comportamiento del recocido simulado, el cual tiene una convergencia muy lenta. El tiempo para encontrar una solución cercana al óptimo tiene un comportamiento asintótico.

- Para la sintonización de los parámetros de los algoritmos de recocido simulado (secuencial y paralelo) se utilizó el clúster Cuexcomate, lo cual permitió hacerlo de forma distribuida y se comprobó que la metodología propuesta por (Juárez Pérez, 2013) es eficaz para reducir considerablemente el tiempo requerido para sintonizar los parámetros del esquema de enfriamiento.

La segunda versión es un algoritmo distribuido con comunicación colectiva con MPI, éste se aplicó a instancias mayores a 200 clientes.

De esta versión se tienen las conclusiones siguientes:

- El ejecutar el algoritmo en los cuatro nodos ayudó a diversificar la búsqueda en el espacio de soluciones a medida que se incrementaba el número de procesos, porque permite hacer una mejor exploración obteniendo resultados cada vez mejores.

- En cuanto a la eficacia, los resultados obtenidos del algoritmo distribuido, variaron de 2.44% a 4.19% con respecto a la mejor cota reportada para cada instancia. La eficacia se mejora porque al ejecutar el algoritmo de forma distribuida, se realiza una mayor explotación del espacio de soluciones en un menor tiempo, dado el procesamiento distribuido.

- Con respecto al Speed-up (eficiencia) su valor alcanzado es del 99.037%, presentó un comportamiento sublineal, ligeramente por debajo del ideal, debido a las demoras generadas por las comunicaciones entre los procesadores.

- Como conclusión general, se puede comentar que el desempeño del algoritmo distribuido con reinicio aplicado para dar solución al modelo del transporte vehicular es aceptable en términos de eficacia y eficiencia.

6.2 Trabajos Futuros

Consecuencia de los resultados obtenidos en este proyecto doctoral, se han generado diversas actividades que se realizarán en el futuro:

- Continuar con el diseño de algoritmos distribuidos, haciendo énfasis en nuevas propuestas y métodos de paralelización para aumentar la eficiencia de las soluciones obtenidas.

- Trabajar con las instancias del Benchmark de Uchoa (X-n), que van de 100 a mil clientes, es necesario explorar instancias más grandes y que son más apegadas a la realidad.

- Buscar nuevos métodos y movimientos de exploración y explotación que puedan ser paralelizadas para ampliar la búsqueda en el espacio de soluciones de los problemas a resolver.

- Trabajar con aplicaciones para grid computacional y aprovechar los modelos de paralelismo.

- El estudio de la aplicación de modelos de paralelismo a otras metaheurísticas como algoritmos genéticos, búsqueda tabú, entre otros, para otras áreas de la ingeniería industrial como Control de Producción, Planeación agregada y Asignación de tareas.

BIBLIOGRAFÍA

A. Grama, G. K. (2003). *Introduction to Parallel Computing* (2a Edición ed.). Addison Wesley.

Aarts, E., & Korst, J. (1989). Simulated Annealing and Boltzmann Machines. *Wiley*.

Aarts, E., & Lenstra, J. (1997). *Local search in combinatorial optimization.* Amsterdam: John wiley & sons Ltd.

Aarts, E., & Lenstra, J. K. (2003). *Local Search in Combinatorial Optimization.* Princeton : Princeton University Press.

Achuthan, N., Caccetta, L., & Hill, S. (1997). On the vehicle routing problem. (E. S. Ltd., Ed.) *Nonlinear Analysis, Theory, Methods & Applications*, 4277-4288.

Ajith, A., Grosan, C., & Pedrycz, W. (2010). *Engineering Evolutionary Intelligent Systems.* Springer Science & Business Media.

Alba, E. (2005). *Parallel metaheuristics: a new class of algorithms.* John wiley & Sons.

Alba, E. (7 de Junio de 2013). *NEO (Networking and Emerging Optimization)*. Recuperado el 13 de Julio de 2015, de http://neo.lcc.uma.es/vrp/vrp-instances/

Alba, E., & Dorronsoro, B. (2008). *A Hybrid cGA for the CVRP, Engineering Evolutionary Intelligent Systems.* (A. Ajith, C. Grosan, & W. Pedrycz, Edits.) Berlin, Germany: Springer Science & Business Media.

Alba, E., Luque, G., & Nesmachnow, S. (2013). Parallel metaheuristics: rcent advances and new trends. *International Transaction in Operational research, 20*(1), 1-48.

Alfonseca, M. (2000). Máquina de Turing. *Números (ejemplar dedicado a: Las matemáticas del siglo XX; una mirada en 101 artículos)*(43-44), 165-168.

Alonso-Pecina, F. (2008). *Programación de horarios escolares con Recocido Simulado y Búsqueda Tabú.* Cuernavaca, Morelos , México.: ITESM.

Alonso-Pecina, F. (2008). *Programación de horarios escolares con Recocido Simulado y Búsqueda Tabú.* Cuernavaca, Morelos, México: ITESM.

Amdahl, G. (1967). Validity of the single processor approach to achieving large scale computing capabilities. *Proceeding AFIPS '67*, (págs. 483-485).

Archetti, C., Feillet, D., Gendreau, M., & Grazia, S. (2011). Complexity of the VRP and SDVRP. *Transportation Research Part C: Emerging Technologies*, 741-750.

Azencott, R. (1992). *Simulated Annealing Parallelization Techniques.* Wiley and Sons.

Balseiro, S., Loiseau, I., & Ramonet, J. (2011). An ant colony algorithm hybridized with insertion heuristics for the time dependent vehicle routing problem with time windows. *Computers & Operations Research*(38), 954-966.

Banzhaf, W. (1990). The molecular traveling salesman. *Biological Cybernetics.*(64), 7-14.

Beckmann, M., & Kunzi, H. P. (1993). *Applied Simulated Annealing.* (R. v. Vidal, Ed.) Berlin: Springer-Verlag.

Benoit, L., & Jin-Kao, H. (2007). A study of neighborhood structures for the multiple depot vehicle schedulling problem. *Lecture Notes in Computer Science*, 197-201.

Berger, J., & Barkaoui, M. (2003). A new hybrid genetic algorithm for the capacitated vehicle routing problem. *Journal of the Operational Research Society, 54*(12), 1254-1262.

Berger, J., & Barkaoui, M. (2004). A new hybrid genetic algorithm for the capacitated vehicle routing problems. *Journal of the Operational Research Society*(54), 1254-1262.

Bermeo, M., & Calderón, S. (2009). Diseño de un modelo de optimización de rutas de transporte. *El hombre y la máquina*, 52-67.

Bianchi, L., Birattari, M., Chiarandini, M., Manfrin, M., & Mastrolili, M. (2004). Metaheuristics for the Vehicle Routing Problem with Stochastic Demands. *Computer Science*, 450-460.

Birattari, M., Chiarandini, M., Manfrin, M., & Mastrolilli, M. (2004). Metaheuristics for the Vehicle Routing Problem with Stochastic Demands. *Lecture Notes in computer Science, Vol. 3242*, 450-460.

Blaise Barney, L. L. (15 de 12 de 2014). *Message Passing Interface (MPI)*. Recuperado el 28 de 04 de 2015, de https://computing.llnl.gov/tutorials/mpi/

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys, Vol. 35*(3), 268-308.

Braekers, K., Ramaekers, K., & Van Nieuwenhuyse, I. (septiembre de 2016). The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering, 99*, 300-313.

Braysy, O., & Gendreau, M. (2001). *Metaheuristics for the vehicle routing problem with time windows.* Oslo, Norway: SINTEF Applied Mathematics.

Calidad, P. d. (1992). *Herramientas Básicas para el Control de Calidad.* Cuernavaca, Morelos: Dirección de Aseguramiento de Calidad Corporativo Nissan.

Chen, P., Huang, H., & Dong, X. (2010). Iterated variable neighborhood descent algorithm for the capacitated vehicle routing problem. *Expert Systems with Applications.*(37), 1620-1627.

Christofides, N., & Beasley, J. (1984). The period routing problem. *Networks*(14), 237-256.

Christofides, N., Mignozzi, A., & Toth, P. (1981). Exact Algorithms for the Vehicle Routing Problem Based on Sapnning Tree and Shortest Path Relaxations. *Mathematical Programming, 20,* 255-282.

Christofides, N., Mingozzi, A., & Toth, P. (1979). Chapter 11. En N. Christofides, A. Mingozzi, P. Toth, & C. Sandi, *Combinatorial optimization* (págs. 315-338). Chichester: John Wiley.

Cruz-Chávez, M., Peralta-Abarca, J., & Moreno-Bernal, P. (marzo-julio de 2014). Aplicación de la teoría de la complejidad en Optimización Combinatoria. *Inventio, 10*(20), 35-42.

Cruz-Chávez, M., Rodríguez-León, A., Rivera-López, R., Juárez-Pérez, F., Peralta-Abarca, J., & Martínez-Oropeza, A. (2012). Grid Platform applied to the Vehicle Routing Problem with Time Windows for the distribution of products. En C. C. Carlos Alberto Ochoa Zezzatti, *Logistics Management and Optimization through Hybrid Artificial Intelligence Systems* (págs. 52-81). USA: igi-global.

Czech, Z., & Czarnas, P. (2002). Parallel simulated annealing for the vehicle routing problem with time windows. *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (EUROMICRO-PDP'02)* (págs. 376-383). Computer Society.

Dantzing , G., & Ramser, J. (1959). The Truck Dispatching problem. *Management Science, 42,* 80-91.

Dasgupta, S., Papadimitriou, C., & Vazirani, U. (2006). *Algorithms.* McGraw-Hill Higher Education.

Daza, M., Montoya, J., & Narducci, F. (2009). Resolución del problema de enrutamiento de vhículos con limitacioes de capacidad utilizando un procedimiento heurístico de dos fases. *EIA*(12), 23-38.

D'Granda T., A. (2013). *ALGORITMO DE AGRUPAMIENTO PARA EL PROBLEMA DEL TRANSPORTE.* Cuernavaca, Morelos, México: UAEM.

D'Granda-Trejo, A. (2013). *Algoritmo de agrupamiento para el problema de transporte (Tesis de Maestría).* Cuernavaca, Morelos, México.: UAEM.

Díaz, B. A., & Dowsland, K. A. (2003). Diseño de Heurística y Fundamentos del Recocido Simulado. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 7*(19), 93-102.

Diekmann, R., Luling, R., & Simon, J. (1993). Problem Independent Distributed Simulated annealing and its Applications. En R. V. Vidal, *Applied Simulated Annealing* (págs. 23-50). Berling.

Dorronsoro Díaz, B. (01 de marzo de 2012). *NEO. Networking and Emerging Optimization*. Recuperado el diciembre de 2014, de The VRP Web: http://neo.lcc.uma.es/vrp/

Dréo , J., Pétrowski, P. S., & Taillard, E. (2006). *Metaheuristics for hard Optimization.* Berlin: Springer -Verlag.

Dréo, J., Pétrowski, P. S., & Taillard, E. (2006). *Metaheuristics for hard Optimization.* Berlin: Springer -Verlag.

Duarte, M., & Pantrigo, F. (2007). *Metaheurísticas.* Madrid, España.: Dykinson.

Eksioglu, B., Vural, A., & Reisman, A. (2009). The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering, 57*(4), 1472-1483. doi:10.1016/j.tre.2011.08.001

Elizondo, C., & Aceves, G. (agosto de 2007). Solución al problema de asignación-distribución. (U. d. Departamento de Matemática, Ed.) *Mosaicos Matemáticos*(20), 51-57.

Ergun, O., Orlin, J., & Steele-Feldman, A. (2006). Creating very large scale neighborhoods out of smaller ones by compounding moves. *Journal of Heuristics*(12), 115-140.

Fogel, D. (1988). An evolutionary approach to the travelingalesman problem. *Biological Cybernetics, 60*, 139-140.

Gallego, R., & Escobar, Z. (2008). *Técnicas Heurísticas de optimización* (2 ed.). (U. T. Pereira, Ed.) Pereira, Colombia: Textos Universitarios.

García Regis, O. R., & Cruz Martínez, E. (2003). *Paralelización.* México, D.F.: UNAM.

Garey, M., & Johnson, D. (1979). *Computers and Intractability a Guide to the Theory of NP-completeness.* New York: W. H. Freeman and Company.

Gendreau, M., Hertz, A., & Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management science, 40*(10), 1276-1290.

Gendreau, M., Laporte, G., & Potvin, J.-Y. (2002). Chapter 6 Metaheuristics for the Capacited VRP. En P. Toth, & D. Vigo, *The Vehicle Routing Problem* (págs. 129-154). Philadelphia: SIAM (Monographs on Discrete Mathematics and Applications).

Gendreau, M., Potvin, J., Braumlaysy, O., Hasle, G., & Lokketangen, A. (2008). Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography. En B. L. Golden, *The vehicle routing problem: latest advances and new challenges* (Vol. 43, págs. 143-169). Springer Science & Business Media.

Gendreau, T. C. (2002). Cooperative parallel tabu search for capacitated network design. *Journal of Heuristics, vol. 8*, pp. 601–627.

Gomez, A., Imran, A., & Salhi, S. (2013). Solution of classical transport problems with bee algorithms. *International Journal of Logistics Systems and Management, 15*(2-3), 160-170.

González, V., & González, A. (2006). Metaheuristics applied to vehicle routing. A case study Part 1: formulating the problem. *Revista Ingeniería e Investigación*, 149-156.

González-Álvarez, D. L. (2013). Tesis Doctoral Metaheurísticas, Optimización Multiobjetivo y Paralelismo para Descubrir Motifs en Secuencias de ADN. Extremadura, España: Universidad de Extremadura.

González-V, G., & González-A, F. (2006). Metaheurísticas Aplicadas al ruteo de vehículos. Un caso de estudio. Parte 1: Formulación del problema. *Ingeniería e Investigación*, 149-156.

Groër, C., Golden, B., & Wasil, E. (2011). A parallel algorithm for the vehicle routing problem. *INFORMS Journal on Computing, 23*(2), 315-330.

Gropp, W., Lusk, E., & Skjellum, A. (1995). Message Passing Interface Forum. *MPI: A Message Passing Interface Standard.*

Guerrero-Campanur, A., Pérez-Loaiza, R., & Olivares-Benitez, E. (2011). Un caso logístico del problema de ruteo vehicular múltiple m VRP resuelto con la heurística de Fisher & Jaikumar. *Taller Latino de Investigación de Operaciones.* Acapulco, Guerrero.

Habibeh, N., & Lai, S. (2012). Optimized crossover genetic algorithm for capacitated vehicle routing problem. *Applied Mathematical Modelling*(36), 2010-2017.

Harmanani, H., Azar, D., Helal, N., & Keirouz, W. (2011). A Simulated Annealing Algorithm for The Capacitated Vehicle Routing Problem. *In Proceedings of the ISCA 26th CATA,* (págs. 96-101). New Orleans, Lousiana, USA.

Hasle, G., & Kloster, O. (2007). Industrial vehicle routing problems. En G. Hastle, K. Lie, & E. Quak, *Geometric Modelling, Numerical Simulation and Optimization.* (págs. 391-426). Springer Verlag.

Hasle, G., & Kloster, O. (2007). Industrial vehicle routing problems. (G. Hasle, & O. Kloster, Edits.) *Geometric modelling, numerical simulation and optimizacion: applied mathematics at SINTEF*, 391-426.

Hennessy, J., Patterson, D., & Larus, J. (1999). *Computer organization and design : the hardware/software interface* (2da ed.). San Francisco: Morgan Kaufmann Publishers.

Hidrobo, F. y. (2005). *Introducción a MPI (Message Passing Interface).* Mérida, Venezuela: Centro Nacional de cálculo Científico, Universidad de los Andes.

Hiller, F., & Lieberman, G. (2012). *Introducción a la Investigación de Operaciones.* México: McGraw Hill.

Juárez Pérez, F. (2013). *Algoritmo genético híbrido cooperativo en ambiente Grid para talleres con flujo flexible.* Cuernavaca, Morelos.: UAEM.

Juárez-Chávez, J., Cruz-Chávez, M., Serna-Barquera, S., Campillo-Illanes, B., Peralta-Abarca, J., Moreno-Bernal, P., & Martínez-Bahena, B. (2012). Neighborhood Hybrid Structure for the Optimization of Mechanical Properties of a Microalloyed Steel Based on its Chemical Composition. *Electronics, Robotics and Automotive Mechanics Conference, CERMA2012,* (págs. 20-23). Cuernavaca, Morelos.: IEEE-Computer Society.

Kirkpatrick , S., Gelatt, C., & Vecchi, M. (1983). Optimization by Simulated annealing. *Science, 220*(4598), 671-680.

Lamb Charles, H. J. (2002). *Marketing.* International Thomson Editores S.A.

Laporte, G. (1992). The Vehicle Routing Problem: an overview of exact and approximate algorithms. *European Journal of Operational Research* , 345-358.

Laporte, G. (1992). The Vehicle Routing Problem: An overview of exact and approximate algorithms. *European Journal of Operational Research, 59*, 345-358.

Laporte, G. (2009). *Fifty Years of Vehicle Routing.* Montreal, Canadá.

Laporte, G., & Semet, F. (2001). En P. Toth, & D. Vigo, *The Vehicle routing problem* (págs. 109-125). Philadelphia: SIAM.

Laporte, G., & Semet, F. (2002). Classical Heuristics for the capacitated VRP. En P. Toth, & D. Vigo, *The Vehicle Routing Problem* (págs. 109 - 128). Philadelphia: SIAM Monographs on Discrete Mathematics and Applications.

Lenstra, J., & Rinnooy Kan, A. (1981). Complexity of vehicle routing and schedulling problems. *Networks*, 221-227.

Lenstra, J., & Rinnooy Kan, A. (1981). Complexity of Vehicle Routing and Scheduling Problems. *Networks, 11*, 221-227.

Lin, S., Lee, Z., Ying, K., & Lee, C. (2009). Applying hybrid meta-heuristics for capacitated vehicle routing problem. *Expert Systems with Applications*(36), 1505-1512.

Ling, S.-W., Ying, K.-C., Lee, Z.-J., & Lee, C.-Y. (2009). Applying hybrid meta-heuristics for Capacitated Vehicle Routing Problem. *Expert Systems with Applications ScientDirect, 36*, 1505-1512.

Lysgaard, J., Letchford, A., & Eglese, R. (2004). A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*(100(2)), 423-445.

Malek, M., Guruswamy, M., Pandya, M., & Owens, H. (1989). Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals od Operations Research, 21*, 59-84.

Marinakis, Y., Marinaki, M., & Dounias, G. (2010). A hybrid particle swarm optimization algorithm for the vehicle routing problem. *Engineering Applications of artificial Intelligence, 23*(4), 463–472.

Martínez-Bahena, B., Cruz-Chávez, M., Díaz-Parra, O., Martínez-Rangel, G., Cruz-Rosales, H., Peralta-Abarca, J., & Yanel-Juárez, J. (2012). Neighborhood Hybrid Structure for Minimum Spanning Tree Problem. *Electronics, Robotics and Automotive Mechanics Conference, CERMA2012* (págs. 191-196). Cuernavaca, Morelos.: IEEE-Computer Society.

Mazzeo, S., & Loiseau, I. (2004). An ant colony algorithm for the capacitated vehicle routing. *Electronic Notes in Discrete Mathematics, 18*, 181-186.

Media, H. (7 de Enero de 2016). *CVRPLIB (Capacitated Vehicle Routing Problem Library)*. Obtenido de http://www.galgos.inf.puc-rio.br/vrp/index.php/en/

Mester, D., & Bräysy, O. (2005). Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Computers & Operation Research, 32(6)*, 1593-1614.

Michaelewicz Z., a. F. (2000). *How to solve it: Modern Heuristics.* Germany: Springer.

Michalewicz, Z., & Fogel, D. B. (2004). *How to Solve It: Modern Heuristics.* Berlin: Springer.

Mole, R., & Jameson, S. (1976). A secuencial route-building algorithm employing a generalised savings criterion. *Operation Research Quaterly, 27*, 502-511.

Moreno Díaz, P., Huecas Fernández-Toribio, G., Sánchez Allende, J., & García Manso, A. (2007). Metaheurísticas de optimización combinatoria: Uso de simulated annealing para un problema de calendarización. *Tecnologí@ y Desarrollo. Revista de ciencia, Tecnología y Medio Ambiente*, 1-25.

Muñoz, B., Claderón Sotero, E., & Hernán, J. (02 de Enero-Junio de 2009). *El Hombre y la Máquina.* Recuperado el 2012 de agosto de 17, de http://redalyc.uaemex.mx/src/inicio/ArtPdfRed.jsp?iCve=47811604005

Nabila Azi, M. G. (2010). An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. *European Journal of Operational Research, 202*(3), 756-763.

Nagata, Y. (2007). Edge assembly crossover for the capacitated vehicle routing problem. *Evolutionary Computation in Combinatorial Optimization, LNCS*(4446), 142-153.

Nalouf, M., De Giusti, A., De Giusti, L., Chichizola, F., Sanz, V., Pousta, A., . . . Balladini, J. (2012). Algoritmos Paralelos y Distribuidos. Fundamentos, Modelos y Aplicaciones. . *XIV Workshop de Investigadores en Ciencias de la Computación*, (págs. 783-787).

NEO Research Group . (30 de Junio de 2016). *NEO (Networking and emerging Optimization)*. Obtenido de NEO (Networking and emerging Optimization): http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances/

Nourani, Y. (1998). a comparasion of simulated annealing cooling strategies. *Journal of Physics, 31*(41), 8373-8386.

Nourani, Y., & Andresen, B. (1998). A comparasion of simulated annealing cooling strategies. *J. Phys. A: Math. Gen.*(31), 8373-8385.

Olivera, A. (2004). *Heurísticas para el problema de vehículos.* Montevidieo, Uruguay: Universidad de la República.

Olivera, A. (2004). *Heurísticas para problemas de ruteo de vehículos.* Montevideo, Uruguay.: Instituto de computación, Facultad de Ingeniería.

Osman, I. (1993). "Metastrategy Simulated Annealing and Tabu Search Algoritms for the Vehicle Routing Problem". *Annals of Operations Research, 41*, 421-451.

Osman, I. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problema. *Annals of Operations Research, 41*, 421-451.

Osman, I. (1993). Metastrategy Simulated Annealing and Tabu Search Algoritms for the Vehicle Routing Problem. *Annals of Operations Research, 41*, 421-451.

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

Papadimitriou, C., & Steiglitz, K. (1998). *Combinatorial Optimization: algorithms and complexity*. New York, USA.: Dover Publications Inc.

Park, N., & Imai, H. (2000). A path-exchange-type local search algorithm for vehicle routing and its efficient search strategy. *Journal of the Operations Research Society of Japan, 43*(1), 197-208.

Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research., 34*, 2403-2435.

Pop, P., Pop, S., Zelina, I., Lupse, V., & Chira, C. (2011). Heuristic algorithms for solving the generalized vehicle routing problem. *Int. J. of computers, Communications & control*, 158-165.

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*(31), 1985-2002.

Ram, D., Sreenivas, T., & Subramaniam, K. (1996). Parallel Simulated Annealing Algorithms. *Journal of Parallel and Distributed Computing*(37), 207-212.

Reeves, C. R. (1996). Modern Heuristics Techniques. (I. O. Rayward-Smith V.J., Ed.) *Modern Heuristics Search Methods*, 1-25.

Restrepo, J. H., Medina, P. D., & Cruz, E. A. (2008). Un problema logístico de programación de vehículos con ventanas de tiempo. *Scientia et Technica Vol. 14, No. 39*, 229-234.

Restrepo, J., & Medina, V. (2008). Problema logístico de programación de vehículos con capacidad finita. *Scientia et Technica*, 253-258.

Robusté Anton, F., & Galván, D. (2005). *e-logístics.* Catalunya, España.: Ediciones UPC.

Rocha, L., González, C., & Orjuela, J. (2011). Una revisión al estado del arte del problema de ruteo de vehículos: Evolución histórica y métodos de solución. *Ingeniería*, 35-55.

Rocha, L., González, C., & Orjuela, J. (2011). Una revisión al estado del arte del problema de ruteo de vehículos: Evolución histórica y métodos de solución. *Ingeniería, 16*(2), 35-55.

Rochat, Y., & Taillard, É. (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of heuristics, 1*(1), 147-167.

Rodríguez, V. (2007). Integración de un SIG con modelos de cálculo y optimización de rutas de vehículos CVRP y software de gestión de flotas. *International Conference on Industrial Engineering & Industrial Management- CIOO*, (págs. 1849-1858). Vigo España.

Romeo, F., & Sangiovannai-Vincentelli, A. (1991). A theorical framework for simulated Annealing. *Algorithmica, 6*(1), 302-345.

Sadiq M. Sait, H. Y. (2000). *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems.* Hoboken, New Jersey, USA: Wiley-IEEE Computer Society Press.

Salazar González, J. (2001). *Programación Matemática.* Madrird (España): Díaz de Santos S. A.

Sanjeev, A., & Boaz, B. (2009). *Complexity Theory: A Modern Approach.* Cambridge, England: Cambrige University Press.

Sanvicente-Sánchez, H., & Frausto-Solis, J. (2004). Method to Establish the cooling Scheme in Simulated Annealing Like Algorithms. *International Conference, ICCSA'2004. 3095*, págs. 755-763. Assis, Italia.: LNCS.

Sedgewick, R., & Wayne, K. (2011). *Algorithms.* Massachusetts, USA: Pearson Education, Inc.

Shih-Wei, L., Zne-Jung, L., Kuo-Ching, Y., & Chou-Yuan, L. (2009). Applying hybrid meta-heuristics for capacitated vehicle routing problem. *Expert Systems with Applications, 36*, 1505-1512.

Subraminan, A., Uchoa, E., & Satoru, O. (2013). A hybrid algorithm for a clas of vehicle routing problem. *Computers & Operations Research, 40*, 2519-2531.

Szeto, W., Yongzhong, W., & Ho, S. (2011). An artificial bee colony algorithm for the capacitated vehicle routing problem. *European Journal of Operational Research, 215*, 126-135.

Talbi, E. G. (2009). Single Solution-based Metaheuristics. *Metaheuristics: From Design to Implementation*, 87-189.

Toth, P., & Vigo, D. (2003). The Granular Tabu Search and its application to the vehicle routing problem. *INFORMS Journal of Computing*(15), 333-348.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Subramanian, A., & Vidal, T. (2014). New benchmark instances for the capacitated vehicle routing problem. *Research Report Engenharia de Producao, Universidad Federal Fluminense*.

Van Breedam, A. (1994). *An Analysis of the Behavior of Heuristics for the Vehicle Routing Problem for a selection of problems with Vehicle-related, Customer-related, and Time-related constraints Contents.* Antwerp, Belgium.: University of Antwerp.

Van Breedam, A. (1995). Improvement heuristics for the Vehicle Routing Problem based on Simulated Annealing. *European Journal of Operational Research*(86), 480-490.

Van Laarhoven, P., Aarts, E., & Korst, J. (1997). Simulated Annealing. En E. Aarts, & J. Lenstra, *Local search in combinatorial Optimization* (págs. 91-120). John Wiley & Sons Ltd.

Vidal, T., Crainic, T., Gendreau, M., Lahrichi, N., & Rei, W. (2012). A hybrid genetic algorithm for multi-depot and periodic vehicle routing problems. *Operations Research*(60(3)), 611-624.

Vigeh, A. (2011). *Investigation of a Simulated Annealing Cooling Schedule used to Optimize the Estimation of the Fiber Diameter Distribution in a Peripheral Nerve Trunk* . San Luis Obispo, California: Faculty of California Polytechnic State University.

Vigo, D., & Toth, P. (2014). *Vehicle Routing Problems, Methods and Applications.* Philadelphia, USA: MOS-SIAM (Society for Industrial and Applied Mathematics) Series on Optimization.

VoB, S. (2001). Meta-heuristics: The State of the Art. (A. Nareyek, Ed.) *Local Search for Planning and Scheduling, LNAI 2148.*, pp. 1 – 23.

Yurtkuran, A., & Emel, E. (2010). A new Hybrid Electromagnetism-like Algorithm for the capacitated vehicle routing problems. *Expert Systems with applications*(37), 3427-3433.

Zanakins, S. H., & Evans, J. R. (1981). *Heuristic 'Optimization': Why, When and How to Use It* (Vol. 11(5)). Catonsville, Maryland, USA: Interfaces.

Zeng, L., Ong, H., & Ng, K. (2005). An assignment-based local search method for solving vehicle routing problems. *Asia-Pacific Journal of Operational Research, 22*(01), 85-104.

A N E X O S

ANEXO 1 Cálculo de la Complejidad del algoritmo de R.S.

***F(n)*)= 7n+9v+20+4n+11n^2+2+6m+8wn+2n+3+ (5n/2)+4+13n+11+6n+13+32n+47+(5n/2)**

***F(n)* = 64n+9v+100+11n^2+6m+8wn + (10n/2)**

***F(n)* = 64n+9v+100+11n^2+6m+8wn + 5n**

***F(n)* = 69n+9v+100+11n^2+6m+8wn**

F(n) = n (69+11n+8w) +9v+100

Línea	Código	Total
1	**leeArchivos();**	**7n+9v+20**
2	**distancias();**	**4n+11n^2+2**
3	s_actual=**calculacostoVh**(s_actual);	**6m+8wn+2n+3**
4	**mejora(solucion vecino)**	**2z+3+16n^2+15n+6mn+8wn^2**
5	s_vecino= unoSwapB(s_vecino);	**(5n/2)+4**
6	s_vecino=insercionB(s_vecino);	**13n+11**
7	s_vecino=cambioVecino(s_vecino);	**6n+13**
8	s_vecino=**perturb**a(s_vecino);	**32n+47+(5n/2)**

Línea 1 Función **leeArchivos()**

s_actual.vh = v cantidad = n

1	void leeArchivos(){		
1.1	scanf("%s",archivo);	1	
1.2	if((archDatos = fopen(archivo,"rt"))==NULL) {	1	
1.3	Else {		
1.4	fscanf(archDatos," %d\t ",&datos.C);	1	
1.5	do {	n	N
1.6	fscanf(archDatos," %d\t	1	

	",&datos.coordX[cantidad]);		
1.7	fscanf(archDatos," %d\t ",&datos.coordY[cantidad]);	1	
1.8	fscanf(archDatos," %d\t ",&datos.dema[cantidad]);	1	
1.9	cantidad++;	1	
1.10	}while(!feof(archDatos));		
1.11	cantidad--;	1	
1.12	fclose(archDatos); }	1	
1.13	s_actual.numClientes=cantidad;	1	
1.14	if((archSol = fopen("rutas.txt","rt"))==NULL) {	1	
1.15	printf("No existe el archivo de rutas inicial"); }		
1.16	else {		
1.17	fscanf(archSol, "%d\t",&s_actual.vh);	1	
1.18	for (j=0; j<cantidad;) {	1+(n+1) +n	2n+2
1.19	fscanf(archSol, "%d\t",&iValor);	1	n
1.20	if(iValor != 0) {	1	n
1.21	s_actual.ruta[j]=iValor;	1	n
1.22	j++; }// if }//for	1	n
1.23	fscanf(archSol, "%d\t",&iValor);	1	
1.24	for(j=0; j<s_actual.vh; j++) {	1+(v+1) +v	2v+2
1.25	fscanf(archSol, "%d\t",&iValor);	1	v
1.26	s_actual.limiteInferior[j]=iValor; } // for	1	v
1.27	fscanf(archSol, "%d\t",&iValor);	1	v
1.28	for(j=0; j<s_actual.vh; j++) {	2v+2	2v+2
1.29	fscanf(archSol, "%d\t",&iValor);	1	v
1.30	s_actual.limiteSuperior[j]=iValor; } //for	1	v
1.31	fclose(archSol); }	1	
		20	**7n+9v**

Línea 2 Función **distancias()**

cantidad = n

2	void distancias() {		
2.1	for(i=0;i<=cantidad;i++) {	1+(n+1)+n	2n+2
2.2	for(j=i+1;j<=cantidad;j++) {	2n+2	n
2.3	xx=(float)(datos.coordX[i]-datos.coordX[j])*(datos.coordX[i]-datos.coordX[j]);	3	n
2.4	yy=(float)(datos.coordY[i]-datos.coordY[j])*(datos.coordY[i]-datos.coordY[j]);	3	n
2.5	mDistancias[j][i] = mDistancias[i][j] = sqrt(xx + yy); } //for } //for }	3	n
		2	**$4n+11n^2$**

Línea 3 Función calculacostoVh ()

p→vh = m

p->limiteInferior = p->limiteInferior = w

3	double calculaCostoVh(solucion *p) {		
3.1	p->costoSolucion=0;	1	
3.2	for(i=0;i<p->vh;i++) {	1+(m+1)+m	2m+2
3.3	suma=0;	1	m
3.4	suma=suma+mDistancias[0][p->ruta[p->limiteInferior[i]]];	3	3m
3.5	for(j=p->limiteInferior[i]; j<p->limiteSuperior[i]; j++) {	2w+2	n
3.6	suma=suma+mDistancias[p->ruta[j]][p->ruta[j+1]]; }	2	2w
3.7	suma=suma+mDistancias[p->ruta[j]][0];	2	2w
3.8	p->costoVh[i]=suma;	1	w
3.9	p->costoSolucion=p->costoSolucion+suma; }	1	w
		3	**6m+8wn+2n**

Línea 4 Función mejora(solucion vecino)

vecino.vh = z

4	solucion mejora(solucion vecino) {		
4.1	for(w=0;w<vecino.vh;w++){	1+(z+1)+z	2z+2

4.2	for(z=0;z<1000000;z++){	2n+2	n
4.3	temp=unoSwap(temp,indice);		**n (12n+9)**
4.4	if(temp.costoVh[indice]<vecino.costoVh[indice]) }	1	n
4.5	vecino=temp; } }	1	n
4.6	suma+=vecino.costoVh[indice];		**n (6m+8wn+2n+4)**
4.7	indice++; }	1	n
4.8	vecino.costoSolucion=suma;	1	
			$2z+3+16n^2+15n+6$ $mn+8wn^2$

Línea 4.3 Función unoSwap(temp,indice)

4.3	solucion unoSwap(solucion p, int indice){		
4.3.1	if (inf!=sup) {	1	
4.3.2	do {	n	
4.3.3	num1=inf+rand()%((sup-inf)+1);	4	n
4.3.4	num2=inf+rand()%((sup-inf)+1);	4	n
4.3.5	} while(num1==num2);		
4.3.6	aux=p.ruta[num1];	1	
4.3.7	p.ruta[num1]=p.ruta[num2];	1	
4.3.8	p.ruta[num2]=aux; }	1	
4.3.9	suma=suma+mDistancias[0][p.ruta[p.limiteInferior[indice]]];	2	
4.3.10	for(j=inf;j<sup;j++) {	n	
4.3.11	suma=suma+mDistancias[p.ruta[j]][p.ruta[j+1]]; }	2	n
4.3.12	suma=suma+mDistancias[p.ruta[j]][0];	2	
4.3.13	p.costoVh[indice]=suma;	1	
			12n+9

Línea 5 Función s_vecino= unoSwapB(s_vecino);

5	solucion unoSwap(solucion p, int indice) {		
5.1	while(num1 == num2) {	n/2	
5.2	num1=rand()%cantidad;	2	n/2
5.3	num2=rand()%cantidad; }	2	n/2
5.4	aux=p.ruta[num1];	1	
5.5	aux2=p.ruta[num2];	1	
5.6	p.ruta[num1]=p.ruta[num2];	1	
5.7	p.ruta[num2]=aux;	1	
			(5n/2)+4

Línea 6 Función s_vecino=insercionB(s_vecino);

6	solucion insercionB(solucion p) {		
6.1	do{	n	
6.2	rSale=rand()%p.vh;	2	2n
6.3	rHueco=rand()%p.vh;	2	2n
6.4	}while(rSale==rHueco);		
6.5	huecoInf=p.limiteInferior[rHueco];	1	
6.6	huecoSup=p.limiteSuperior[rHueco];	1	
6.7	saleInf=p.limiteInferior[rSale];	1	
6.8	saleSup=p.limiteSuperior[rSale];	1	
6.9	hueco=huecoInf+rand()%((huecoSup-huecoInf)+1);	1	
6.10	sale=saleInf+rand()%((saleSup-saleInf)+1)	1	
6.11	if (hueco < sale)//E-S {	1	
6.12	aux= p.ruta[sale];	1	
6.13	for (c=sale; c > hueco; c--)	2n+2	
6.14	p.ruta[c]= p.ruta[c-1];	1	n
6.15	p.ruta[hueco]=aux; }	1	n
6.16	else {	/	
6.17	aux= p.ruta[sale];	/	
6.18	for (c=sale; c < hueco; c++)	/	
6.19	p.ruta[c]= p.ruta[c+1];	/	
6.20	p.ruta[hueco]=aux; }	/	
6.21	if (rHueco < rSale){	1	
6.22	p.limiteSuperior[rHueco]=huecoSup + 1;	1	
6.23	for(i=rHueco + 1 ;i< rSale ;i++) {	2n+2	
6.24	p.limiteInferior[i]= p.limiteInferior[i] + 1;	1	n
6.25	p.limiteSuperior[i]= p.limiteSuperior[i] + 1; }	1	n
6.26	p.limiteInferior[rSale]=saleInf + 1; }	1	
6.27	else {	/	
6.28	p.limiteSuperior[rSale]=saleSup - 1;	/	
6.29	for(i=rSale + 1 ;i< rHueco ;i++) {	/	
6.30	p.limiteInferior[i]=p.limiteInferior[i]- 1;	/	
6.31	p.limiteSuperior[i]= p.limiteSuperior[i]- 1; }	/	
6.32	p.limiteInferior[rHueco]=huecoInf - 1; }	/	
			13n+11

Línea 7 Función s_vecino=cambioVecino(s_vecino);

s_actual.vh = v cantidad = n

7	solucion cambioVecino(solucion p) {		
7.1	num1=(rand()%(cantidad-2))+1;	3	
7.2	min=mDistancias[num1+1][1];	1	
7.3	for(i=1;i<=cantidad;i++) {	2n+2	
7.4	if(i != (num1+1) && mDistancias[num1+1][i]<min) {	2	n
7.5	min=mDistancias[num1+1][i];	1	n
7.6	pos=i; }//if }//for	1	n
7.7	num2=pos-1;	1	
7.8	if(rand()%2)	1	
7.9	vecino=(num1-1);	1	
7.10	Else		
7.11	vecino=(num1+1);	1	
7.12	aux= p.ruta[vecino];	1	
7.13	p.ruta[vecino]=p.ruta[num2];	1	
7.14	p.ruta[num2]=aux; } función}	1	
			6n+13

Línea 8 Función perturba(s_vecino);

8	solucion perturba(solucion s_vecino){		
8.1	opcion=1+(rand()%4);	3	
8.2	switch (opcion) {	1	
8.3	case 1:s_vecino=unoSwapB(s_vecino);		**(5n/2)+4**
8.4	break;	1	
8.5	case 2:s_vecino=insercionB(s_vecino);		**13n+11**
8.6	break;	1	
8.7	case 2:s_vecino=insercionB(s_vecino);		**13n+11**
8.8	break;	1	
8.9	case 4:s_vecino=cambioVecino(s_vecino);		**6n+13**
8.10	break;	1	
		(5n/2)	**32n+47**

ANEXO 2 Código del algoritmo secuencial de recocido simulado

```
///  recocido simulado simple  para cvrp
///  sin apuntadores
///  esta version incluye una estructura de vecindad de cambio de vecino  insercion guiada por cercania
///  revisa factibilidad
///  tiene reinicio
///*************************************************************************
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
#include<time.h>
//#include<conio.h>
#define LIMITE_MAX 200
#define LM (LIMITE_MAX/2)+2

/**************ESTRUCTURAS*********************************/
typedef struct
{
   int C;
   int coordX[LIMITE_MAX+2];
   int coordY[LIMITE_MAX+2];
   int dema[LIMITE_MAX+2];
}instancia;
instancia datos;
typedef struct
{
   int vh;//numero de vehiculos (rutas creadas)
   int numClientes;//cantidad de clientes
   int ruta[LIMITE_MAX+2];    //conjunto de clientes que forman una rutae oreden de la configuracion
   int limiteSuperior[LM];//marcan los segmentos superiores de ruta en el vector.ruta
   int limiteInferior[LM];//marcan los segmentos inferiores de ruta en el vector.ruta
   double costoVh[LM];//costo de recorrido de un solo vehiculo
```

```
   double costoSolucion;//costo solucion de todas las rutas generadas (valor de la Funcion Objetivo)
   double cargaVh[LM];
}solucion;
solucion s_actual,  s_vecino, s_mejorRS;

/***********PROTOTIPO DE FUNCIONES***********************/
void leeArchivos();
void distancias();
solucion calculaCostoVh(solucion p);
solucion perturba(solucion s_vecino);
solucion unoSwapB(solucion p);
solucion cambioVecino(solucion p);
solucion mejora(solucion vecino);
solucion unoSwap(solucion p, int indice);
solucion insercionB(solucion p);
int  guardaResultado(double, double, double, double, int, float,double,int);
int imprimeRutaOptimizada(solucion s_mejorRS);

/***********VARIABLES************************************/
float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2];
int cantidad=0;
int cont=0;
float tiempo=0;
int semilla;
int banderaCapa=1;
int flag=0;

/**************FUNCION PRINCIPAL*****************************/
int main()
{
   double tempInicial,tempIni=20.0;
   double tempFin= 0.001;
   double alfa= 0.986;
   int LongM=50000;
   double costoInicial=0; //costo de entrada de la solucion incial
   int i,k;
   double u=0.0,probAcepta=0.0;
   int choco=0;float recha,acep,b,m,porcentaje;
```

```
FILE *volado;
   semilla=(time(NULL));
   srand(semilla);
   tempInicial=tempIni;
   clock_t start,end;
   leeArchivos();
   distancias();
   s_actual=calculaCostoVh(s_actual);
   s_actual=mejora(s_actual);
   volado=fopen("Estadistica.txt","w");
          costoInicial=s_actual.costoSolucion;
   s_mejorRS= s_actual ;

   start=clock();
 ///********    REINICIO  ***********///
   for(k=0;k<300;k++){
  s_actual = s_mejorRS;

   ///********** R E C O C I D O  S I M U L A D O ******************

  for( tempIni=tempInicial;tempIni >= tempFin;tempIni *= alfa)
   {
      choco++;recha=0;acep=0,b=0,m=0,porcentaje=0;
    for(i = 0;i <= LongM ;i++)
    {
         s_vecino=s_actual;
         s_vecino=perturba(s_vecino);
                          s_vecino=calculaCostoVh(s_vecino);

         if(banderaCapa)
         {
            if ((s_vecino.costoSolucion < s_actual.costoSolucion))
            {
               b++;
               s_actual=s_vecino;
               if(s_vecino.costoSolucion < s_mejorRS.costoSolucion)
                 {
```

```
                    s_mejorRS=s_vecino;
                  }
              }
              else
              {
                  probAcepta = exp((-1)*((double)(s_vecino.costoSolucion -
s_actual.costoSolucion)/10/tempIni));
                    u = ((double)rand()/ RAND_MAX);
                 if(u < probAcepta)
                    {acep++;
                    s_actual=s_vecino;}
                 else{ recha++;}
              }
           }
       }
       m=(acep+recha);
       porcentaje= ((acep * 100)/m) ;
       if(s_mejorRS.costoSolucion<VALOR INSTANCIA)
        {
           end=clock(); tiempo=(float)(end-start)/CLOCKS_PER_SEC;
           guardaResultado(tempInicial,tempFin, s_mejorRS.costoSolucion,alfa, LongM,
tiempo,costoInicial,semilla);
           imprimeRutaOptimizada(s_mejorRS);
           exit(0);
        }
    }
 }
    ///*****************************************************
    end=clock(); tiempo=(float)(end-start)/CLOCKS_PER_SEC;
    guardaResultado(tempInicial,tempFin, s_mejorRS.costoSolucion,alfa, LongM,
tiempo,costoInicial,semilla);
    imprimeRutaOptimizada(s_mejorRS);
    fclose(volado);
   return (0);
}///FIN DE MAIN

///********************** FUNCIONES **********************************
/*********************************************************************
```

```
void leeArchivos(){
   FILE *archDatos;
   FILE *archSol;
   int j,iValor;
   if((archDatos = fopen("NOMBRE INSTANCIA.txt","rt"))==NULL)
      printf("No existe el archivo de datos");
          else{
      fscanf(archDatos," %d\t ",&datos.C);
                    do{
         fscanf(archDatos," %d\t ",&datos.coordX[cantidad]);
         fscanf(archDatos," %d\t ",&datos.coordY[cantidad]);
         fscanf(archDatos," %d\t ",&datos.dema[cantidad]);
         cantidad++;
      }while(!feof(archDatos));
      cantidad--;
      fclose(archDatos);
   }
   s_actual.numClientes=cantidad;
   if((archSol = fopen("rutas.txt","rt"))==NULL){
      printf("No existe el archivo de rutas inicial");
   }else{
        fscanf(archSol,  "%d\t",&s_actual.vh );
                    for(j=0; j<cantidad;)
         {
                              fscanf(archSol,  "%d\t",&iValor );
                              if(iValor != 0)
                              {
                                        s_actual.ruta[j]=iValor;
                                        j++;
                              }
                     }

      fscanf(archSol,  "%d\t",&iValor );
      for(j=0;j<s_actual.vh;j++)
       {
         fscanf(archSol,  "%d\t",&iValor );
         s_actual.limiteInferior[j]=iValor;
       }
```

```
        fscanf(archSol,  "%d\t",&iValor );
        for(j=0;j<s_actual.vh;j++)
        {  fscanf(archSol,  "%d\t",&iValor );
           s_actual.limiteSuperior[j]=iValor;
        }
            }
         fclose(archSol);
}
/*************************************************************************/
void distancias(){
   int i,j;
   float xx,yy;
   for(i=0;i<=cantidad;i++){
      for(j=i+1;j<=cantidad;j++){
         xx=(float)(datos.coordX[i]-datos.coordX[j])*(datos.coordX[i]-datos.coordX[j]);
         yy=(float)(datos.coordY[i]-datos.coordY[j])*(datos.coordY[i]-datos.coordY[j]);
         mDistancias[j][i] = mDistancias[i][j] = sqrt(xx + yy );
      }
         }
}
/*************************************************************************/
 solucion calculaCostoVh(solucion p)
{
         int i,j=0;
         double suma=0,sumaDema=0;
         banderaCapa=1;///
         p.costoSolucion=0;
         for(i=0;i<p.vh;i++)
         {
                  suma=0;
            sumaDema=0;
                  suma=suma+mDistancias[0][p.ruta[p.limiteInferior[i]]];
                  for(j=p.limiteInferior[i];j<p.limiteSuperior[i];j++)
                   {
                            suma=suma+mDistancias[p.ruta[j]][p.ruta[j+1]];
                            sumaDema=sumaDema+datos.dema[p.ruta[j]];
                   }
                  suma=suma+mDistancias[p.ruta[j]][0];
```

```
                sumaDema=sumaDema+datos.dema[p.ruta[j]];
                p.cargaVh[i]=sumaDema;
        p.costoVh[i]=suma;
                p.costoSolucion=p.costoSolucion+suma;
                if(sumaDema > datos.C){ banderaCapa=0; }
          }
          return (p);
}
/********************* ESTRUCTURA DE VECINDAD **************/

solucion perturba(solucion s_vecino){
int opcion;
opcion=1+(rand()%4);
                        switch (opcion)
                        {
                                case 1:s_vecino=unoSwapB(s_vecino);
                                        break;
                                case 2:s_vecino=insercionB(s_vecino);
                break;
                                case 3:s_vecino=insercionB(s_vecino);
                                        break;
                                case 4:s_vecino=cambioVecino(s_vecino);
                                        break;
                        }
return(s_vecino);
}
/*********************HEURÍSTICAS DE INTERCAMBIO*******************/
/*****************************************************************/

solucion unoSwapB(solucion p)
{
          int num1=0,num2=0,aux=0,aux2=0;
          while(num1 == num2)
          {
                num1=rand()%cantidad;
                num2=rand()%cantidad;
          }
          aux=p.ruta[num1];
```

```
   aux2=p.ruta[num2];
          p.ruta[num1]=p.ruta[num2];
          p.ruta[num2]=aux;
          return(p);
}

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

solucion cambioVecino(solucion p)
{
int min,num1,num2,i,aux,vecino,pos=1;
num1=(rand()%(cantidad-2))+1;
min=mDistancias[num1+1][1];
for(i=1;i<=cantidad;i++)
   {
    if(i != (num1+1) && mDistancias[num1+1][i]<min)
       {
        min=mDistancias[num1+1][i];
        pos=i;
       }
   }
  num2=pos-1;
  if(rand()%2)
     vecino=(num1-1) ;
 else
   vecino=(num1+1);
    aux= p.ruta[vecino];
    p.ruta[vecino]=p.ruta[num2];
    p.ruta[num2]=aux;
return(p);
}

///*************************************************************************
solucion unoSwap(solucion p, int indice)
{
          int num1=0,inf=0, sup=0,num2=0,aux=0;
          int suma=0,j;
          inf=p.limiteInferior[indice];
```

```
            sup=p.limiteSuperior[indice];
            if (inf!=sup)
            {
              do
               {
        num1=inf+rand()%((sup-inf)+1);
        num2=inf+rand()%((sup-inf)+1);
       } while(num1==num2);
            aux=p.ruta[num1];
            p.ruta[num1]=p.ruta[num2];
            p.ruta[num2]=aux;
            }
                        suma=suma+mDistancias[0][p.ruta[p.limiteInferior[indice]]];
                        for(j=inf;j<sup;j++)
                        {
                                suma=suma+mDistancias[p.ruta[j]][p.ruta[j+1]];
                        }
                        suma=suma+mDistancias[p.ruta[j]][0];

                        p.costoVh[indice]=suma;
            return(p);
}

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

solucion insercionB(solucion p)
{
            int c,rHueco,huecoInf,huecoSup,hueco,rSale,saleInf,saleSup,sale,aux,i;

do{
            rSale=rand()%p.vh;
            rHueco=rand()%p.vh;
}while(rSale==rHueco);
            huecoInf=p.limiteInferior[rHueco];
            huecoSup=p.limiteSuperior[rHueco];
            saleInf=p.limiteInferior[rSale];
            saleSup=p.limiteSuperior[rSale];
           hueco=huecoInf+rand()%((huecoSup-huecoInf)+1);
```

```
        sale=saleInf+rand()%((saleSup-saleInf)+1);
   if (hueco < sale)
                {
                        aux= p.ruta[sale];
                        for (c=sale; c > hueco; c--)
                                p.ruta[c]= p.ruta[c-1];
                           p.ruta[hueco]=aux;
                }
    else
                {
        aux= p.ruta[sale];
                        for (c=sale; c < hueco; c++)
                                p.ruta[c]= p.ruta[c+1];
                         p.ruta[hueco]=aux;
      }
 if (rHueco < rSale)
                {
                p.limiteSuperior[rHueco]=huecoSup + 1;
                for(i=rHueco + 1  ;i< rSale   ;i++)
      {
         p.limiteInferior[i]= p.limiteInferior[i] + 1;
         p.limiteSuperior[i]= p.limiteSuperior[i] + 1;
      }
                p.limiteInferior[rSale]=saleInf + 1;
                }
    else
                {
                p.limiteSuperior[rSale]=saleSup - 1;
                for(i=rSale + 1  ;i< rHueco   ;i++)
      {
         p.limiteInferior[i]=p.limiteInferior[i]- 1;
         p.limiteSuperior[i]= p.limiteSuperior[i]- 1;
      }
                p.limiteInferior[rHueco]=huecoInf - 1;
      }
 return(p);
}
/*************************************************************/
```

```
solucion mejora(solucion vecino)
{
solucion temp;
unsigned int w,z;
double suma=0;
int indice=0;
temp=vecino;
   for(w=0;w<vecino.vh;w++){
       for(z=0;z<1000000;z++){
            temp=unoSwap(temp,indice );
            if(temp.costoVh[indice]<vecino.costoVh[indice])  {
           vecino=temp;  }
       }
      suma+=vecino.costoVh[indice];
      indice++;
   }
   vecino.costoSolucion=suma;
return(vecino);
}
///*********************  IMPRESION DE RESULTADOS   **********************

int guardaResultado(double tempInicial,double tempFin, double costo_RS, double alfa, int longM, float
tiempo,double costoInicial,int semilla)
{
  FILE *salida_txt;
   salida_txt = fopen("salida.txt","w");
   if (!salida_txt){
        return -1;}
   fprintf(salida_txt,"Costo Inicial   %.2f  \n",costoInicial);
   fprintf(salida_txt,"Cadena de Markov  %d  \n",longM);
   fprintf(salida_txt,"Costo MejorRS %.2f  \n",costo_RS);
   fprintf(salida_txt,"Semilla  %d  \n",semilla);
   fprintf(salida_txt,"Iteracion  %d    temperaturaInicial  %f    temperaturaFin  %f    alfa  %.3f  tiempo
%.2f\n",cont, tempInicial,tempFin, alfa, tiempo);
   fclose(salida_txt);
   return (0);
}
```

```
///************************************************
int imprimeRutaOptimizada(solucion s_mejorRS){
   FILE *ruta_txt;
   int i,k;
    ruta_txt = fopen("Mejor_ruta.txt","w");
    if (ruta_txt==NULL)
          return -1;
    else
    {
        fprintf(ruta_txt,"%d\n",s_mejorRS.vh);
        for (i=0,k=0; i<s_mejorRS.numClientes; i++)
         {
           fprintf(ruta_txt,"%2d ",s_mejorRS.ruta[i]);
           if(s_mejorRS.limiteSuperior[k]==i)
            {
                fprintf(ruta_txt,"\n");
               k++;
            }
         }
        fprintf(ruta_txt,"\n");
        for (k=0; k<s_mejorRS.vh;k++)
         {
           fprintf(ruta_txt,"%d ",s_mejorRS.limiteInferior[k]);
         }
        fprintf(ruta_txt,"\n");
        for (k=0; k<s_mejorRS.vh;k++)
          {
             fprintf(ruta_txt,"%d ",s_mejorRS.limiteSuperior[k]);
          }
        fprintf(ruta_txt,"Cargas  \n");
        for (k=0; k<s_mejorRS.vh;k++)
         {
           fprintf(ruta_txt,"%lf ",s_mejorRS.cargaVh[k]);
         }
    }
    fclose(ruta_txt);
    return 0;
}
```

ANEXO 3 Código del algoritmo distribuido de recocido simulado

```
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
#include<time.h>
#include "mpi.h"
#define LIMITE_MAX 270
#define LM ((LIMITE_MAX/2)+2)

/**************ESTRUCTURAS*********************************/

typedef struct {
   int C;
   int coordX[LIMITE_MAX+2];
   int coordY[LIMITE_MAX+2];
   int dema[LIMITE_MAX+2];
}instancia;
instancia datos;

typedef struct {
   int ruta[LIMITE_MAX+2];
   int limiteSuperior[LM]
   int limiteInferior[LM];
   double costoVh[LM];
   double cargaVh[LM];
   int vh;
   int numClientes;
   double costoSolucion;
}solucion;
solucion s_actual,s_costoMPI,  s_vecino, s_mejorRS,s_global, s_solucionfinal[4],
s_actualVEC,s_rutaIni[64];

/************PROTOTIPO DE FUNCIONES************************/

void leeArchivos();
void distancias(solucion s_actual,float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2],int
datoscoordX[LIMITE_MAX+2],int datoscoordY[LIMITE_MAX+2]);
int imprimeRutaOptimizada(double costoInicial, solucion s_mejorRS,double tempInicial, double
tempFin, double alfa, int LongM, float tiempo, int rank);
solucion calculaCostoVh(solucion p,float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2],int
datosdema[LIMITE_MAX+2]);
int evaluaCapacidad(solucion p,int datosC);
solucion mejora(solucion s_actual,float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2],int rank);
solucion mejora2(solucion s_actual,float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2],int rank);
solucion unoSwap(solucion p, int indice,float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2]);
solucion unoParAdyacente(solucion p, int indice,float
mDistancias[LIMITE_MAX+2][LIMITE_MAX+2],int cliente);
```

```
solucion perturba(solucion s_vecino ,float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2]);
solucion unoSwapB(solucion p);
solucion cambioVecino(solucion p,float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2]);
solucion insercionB(solucion p);
solucion intercambioParAdyacenteB(solucion p);

/**************FUNCION PRINCIPAL*****************************/

int main(int argc, char **argv)
{
   int nprocs;
   int rank;
   char nombrepr[MPI_MAX_PROCESSOR_NAME];
   double costoInicial=0;
   int i,k;
   int datosC;
   int datosdema[LIMITE_MAX+2];
   int datoscoordX[LIMITE_MAX+2];
   int datoscoordY[LIMITE_MAX+2];
   int limite;
   int NC;

  /********INICIA AMBIENTE PARALELO  - Cluster CUEXCOMATE*******/

   MPI_Init(&argc, &argv);
   MPI_Status  status;

/********Inicia creación de tipo de Dato Derivado ******/

   MPI_Datatype   ruteo;
   MPI_Datatype  tipoDatoSol[8]={MPI_INT, MPI_INT,  MPI_INT,MPI_DOUBLE,  MPI_DOUBLE
,MPI_INT,MPI_INT,MPI_DOUBLE};

   int  bloqueSol[8]={LIMITE_MAX+2,LM, LM,LM,LM, 1 ,1,1};
   MPI_Aint
movSol[8]={0,(LIMITE_MAX+2)*sizeof(int),(LIMITE_MAX+2)*sizeof(int)+((LM)*sizeof(int)),(LIMITE_MA
X+2)*sizeof(int)+((LM)*sizeof(int)) +((LM)*sizeof(int)),(LIMITE_MAX+2)*sizeof(int)+((LM)*sizeof(int))
+((LM)*sizeof(int))+((LM)*sizeof(double)),(LIMITE_MAX+2)*sizeof(int)+((LM)*sizeof(int))
+((LM)*sizeof(int))+((LM)*sizeof(double))+((LM)*sizeof(double)),(LIMITE_MAX+2)*sizeof(int)+((LM)*siz
eof(int))
+((LM)*sizeof(int))+((LM)*sizeof(double))+((LM)*sizeof(double))+sizeof(int),(LIMITE_MAX+2)*sizeof(int
)+((LM)*sizeof(int))
+((LM)*sizeof(int))+((LM)*sizeof(double))+((LM)*sizeof(double))+sizeof(int)+sizeof(int)};

   MPI_Type_struct(8, bloqueSol, movSol, tipoDatoSol, &ruteo);
   MPI_Type_commit(&ruteo);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
   MPI_Barrier(MPI_COMM_WORLD);
   float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2];

  srand(time(NULL)/(rank+1));

 if (rank==0){
   leeArchivos();
   limite=LIMITE_MAX;
```

```
   datosC=datos.C;

   for(i=0;i<= s_rutaIni[0].numClientes ;i++) datosdema[i]  =datos.dema[i];
   for(i=0;i<= s_rutaIni[0].numClientes ;i++) datoscoordX[i]  =datos.coordX[i];
   for(i=0;i<= s_rutaIni[0].numClientes ;i++) datoscoordY[i]  =datos.coordY[i];
   }

   MPI_Bcast(&datosdema, 234, MPI_INT, 0, MPI_COMM_WORLD);
   MPI_Bcast(&datoscoordX, 234, MPI_INT, 0, MPI_COMM_WORLD);
   MPI_Bcast(&datoscoordY, 234, MPI_INT, 0, MPI_COMM_WORLD);

   MPI_Bcast(&datosC, 1, MPI_INT, 0, MPI_COMM_WORLD);
   MPI_Bcast(&costoInicial, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
   MPI_Scatter(&s_rutaIni,1,ruteo,&s_actualVEC,1,ruteo,0,MPI_COMM_WORLD);

   double tempIni,tempInicial=15.0;
   double tempFin= 0.00001;
   double alfa= 0.986;
   int LongM=40000;
   double u=0.0,probAcepta=0.0;
   clock_t start,end;
   int metro;
   int reinicio;
   float tiempo=0;
   char NomArchivo[25];

   int capa=2;

   s_actual=s_actualVEC;
   distancias(s_actual,mDistancias,datoscoordX,datoscoordY);

   s_actual=calculaCostoVh( s_actual,mDistancias,datosdema);
   s_actual=mejora(s_actual,mDistancias,rank);
   s_actual=calculaCostoVh( s_actual,mDistancias,datosdema);

/*****************************************************REINICIO*****************************************************/

 start=clock();
 for(reinicio=0;reinicio<300;reinicio++){
     s_actual = s_mejorRS;


  ///*****************************RECOCIDO SIMULADO*****************************///

  for( tempIni=tempInicial;tempIni >= tempFin;tempIni *= alfa){
   for(metro = 0;metro <= LongM ;metro++){
              s_vecino=s_actual;
              s_vecino=perturba(s_vecino,mDistancias);
              s_vecino=calculaCostoVh(s_vecino,mDistancias,datosdema);
              capa=evaluaCapacidad(s_vecino,datosC);
              if(capa != 0)     {
                  if((s_vecino.costoSolucion < s_actual.costoSolucion)){
                        s_actual=s_vecino;
                        if(s_vecino.costoSolucion < s_mejorRS.costoSolucion) {
                             s_mejorRS=s_vecino;      }
               }
```

```
                else
                   {
                        probAcepta = exp((-1)*((double)(s_vecino.costoSolucion -
s_actual.costoSolucion)/1000/tempIni));
                        u = ((double)rand()/ RAND_MAX);
                        if(u < probAcepta)
                          {
                              s_actual=s_vecino;
                          }
                         else{ }//recha++;   }
                   }

           }    ///fin de metropolis segundo for

       }   ///fin primer for del recocido

}   ///FIN DEL REINICIO

  end=clock(); tiempo=(float)(end-start)/CLOCKS_PER_SEC;
  s_actualVEC=s_mejorRS;
  MPI_Barrier(MPI_COMM_WORLD);
  imprimeRutaOptimizada(costoInicial, s_mejorRS,tempInicial,tempFin,alfa, LongM, tiempo, rank);
  MPI_Type_free(&ruteo);
  MPI_Finalize();
  return (0);
}   //FIN DE MAIN

/************************************** ARCHIVOS DEL PROGRAMA  **********************************/

void leeArchivos(){
    FILE *archDatos;//DECLARACION DEL ARCHIVO TIPO FILE
    FILE *archSol;//DECLARACION DEL ARCHIVO TIPO FILE
    char archivo[20]; //Para ver el nombre
    int i, j,iValor;
    int cantidad=0;//numero de clientes leidos
    if((archDatos = fopen("Xn233.txt","rt"))==NULL)
        printf("No existe el archivo de datos");
else{
        fscanf(archDatos," %d\t ",&datos.C);//lee el archivo de datos en sus tres columnas x, y, demanda
        do{
            fscanf(archDatos," %d\t ",&datos.coordX[cantidad]);
            fscanf(archDatos," %d\t ",&datos.coordY[cantidad]);
            fscanf(archDatos," %d\t ",&datos.dema[cantidad]);
            cantidad++;
        }while(!feof(archDatos));

        cantidad--;
        fclose(archDatos);

    }

    for(i=0;i<64;i++)
      {
            s_rutaIni[i].numClientes=cantidad;
            sprintf(archivo, "rutas%d.txt", i);
        if((archSol = fopen(archivo,"rt"))==NULL){
```

```
        printf("No existe el archivo de rutas inicial");
      }
      else
      {
      fscanf(archSol,  "%d\t",&s_rutaIni[i].vh );
            for(j=0; j<cantidad;) {
               fscanf(archSol,  "%d\t",&iValor );//obtiene los datos del vector solucion
               if(iValor != 0) {
                    s_rutaIni[i].ruta[j]=iValor;//va leyendo los datos del vector solucion
                    j++;
               }//fin de if
            }//fin de for

  /* **************************** AQUÍ CALCULA LOS LÍMTES INF. Y SUP. **********************/

       fscanf(archSol,  "%d\t",&iValor );
       for(j=0;j<s_rutaIni[i].vh;j++)//va a leer hasta el numero de rutas creadas
          {
            fscanf(archSol,  "%d\t",&iValor );
            s_rutaIni[i].limiteInferior[j]=iValor;
          }
       fscanf(archSol,  "%d\t",&iValor );
       for(j=0;j<s_rutaIni[i].vh;j++)
          {  fscanf(archSol,  "%d\t",&iValor );
            s_rutaIni[i].limiteSuperior[j]=iValor;
          }
      }//fin del else

   fclose(archSol);

   }
}


void distancias(solucion s_actual,float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2],int
datoscoordX[LIMITE_MAX+2],int datoscoordY[LIMITE_MAX+2])
{
   int i,j;
   float xx,yy;
   for(i=0;i<=s_actual.numClientes;i++)
      {
      for(j=i+1;j<=s_actual.numClientes;j++)
      {
         xx=(float)(datoscoordX[i]-datoscoordX[j])*(datoscoordX[i]-datoscoordX[j]);
         yy=(float)(datoscoordY[i]-datoscoordY[j])*(datoscoordY[i]-datoscoordY[j]);
         mDistancias[j][i] = mDistancias[i][j] = sqrt(xx + yy );

      }
            }
}

solucion calculaCostoVh(solucion p,float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2],int
datosdema[LIMITE_MAX+2])
{
            int i,j=0;
            float suma=0,sumaDema=0;
```

```
            p.costoSolucion=0;
            for(i=0;i<p.vh;i++)
            {
                suma=0;
                sumaDema=0;
                suma=suma+mDistancias[0][p.ruta[p.limiteInferior[i]]];
                    for(j=p.limiteInferior[i];j<p.limiteSuperior[i];j++)
                        {
                            suma=suma+mDistancias[p.ruta[j]][p.ruta[j+1]];
                            sumaDema=sumaDema+datosdema[p.ruta[j]];
                        }
                 suma=suma+mDistancias[p.ruta[j]][0];
                 sumaDema=sumaDema+datosdema[p.ruta[j]];
                 p.cargaVh[i]=sumaDema;
                        p.costoVh[i]=(double)suma;
                        p.costoSolucion=p.costoSolucion+p.costoVh[i];
            }

            return (p);
}


int evaluaCapacidad(solucion p,int datosC)
{
            int i=0;
            int bCapa=1;
            for(i=0;i<p.vh;i++)
            {
                  if(p.cargaVh[i] > datosC)
                 {
               bCapa=0;
          }
                }

            return (bCapa);
}

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

solucion mejora(solucion s_actual,float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2],int rank)
{
solucion temp;
unsigned int w,z;
double suma=0;
int indice=0;
temp=s_actual;

   for(w=0;w<s_actual.vh;w++){
      for(z=0;z<10000000;z++){
      temp=s_actual;
      temp=unoSwap(temp,indice,mDistancias );
      if(temp.costoVh[indice]<s_actual.costoVh[indice])
         {
          s_actual=temp;
         }
```

```
       }
      suma+=s_actual.costoVh[indice];
       indice++;
      }
s_actual.costoSolucion=suma;
return(s_actual);
}

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

solucion unoSwap(solucion p, int indice,float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2])
{
            int num1=0,inf=0, sup=0,num2=0,aux=0;

            int j;
            float suma=0.0;
            inf=p.limiteInferior[indice];
            sup=p.limiteSuperior[indice];

            if (inf!=sup)
            {
              do
               {
                  num1=inf+rand()%((sup-inf)+1);
                  num2=inf+rand()%((sup-inf)+1);
                } while(num1==num2);

              aux=p.ruta[num1];
              p.ruta[num1]=p.ruta[num2];
              p.ruta[num2]=aux;
            }//fin DE IF
      j=inf;
                        suma=suma+mDistancias[0][p.ruta[j]];
                        for(j=inf;j<sup;j++)
                        {
                                    suma=suma+mDistancias[p.ruta[j]][p.ruta[j+1]];
                        }
                        suma=suma+mDistancias[p.ruta[j]][0];
                        p.costoVh[indice]=suma;
            return(p);
}

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

solucion mejora2(solucion s_actual,float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2],int rank)
{
solucion temp;
unsigned int w,z;
double suma=0;
int indice=0;
temp=s_actual;

   for(w=0;w<s_actual.vh;w++){
      for(z=temp.limiteInferior[w];z<temp.limiteSuperior[w];z++){
      temp=s_actual;
```

```
        temp=unoParAdyacente(temp,indice,mDistancias,z );
        if(temp.costoVh[indice]<s_actual.costoVh[indice])
           {
            s_actual=temp;
           }
        }
       suma+=s_actual.costoVh[indice];
        indice++;
       }
s_actual.costoSolucion=suma;
return(s_actual);
}

solucion unoParAdyacente(solucion p, int indice,float
mDistancias[LIMITE_MAX+2][LIMITE_MAX+2],int cliente)
{
            int j,aux=0,inf=0,sup=0;
            float suma=0.0;

   inf=p.limiteInferior[indice];
            sup=p.limiteSuperior[indice];
   j=inf;
            /*intercambio*/
            aux= p.ruta[cliente];
            p.ruta[cliente]=p.ruta[cliente+1];
            p.ruta[cliente+1]=aux;
                        suma=suma+mDistancias[0][p.ruta[j]];//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];
                        p.costoVh[indice]=suma;
            return(p);
}


solucion perturba(solucion s_vecino, float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2]){
int opcion;

opcion=1+(rand()%4);
switch (opcion)
            {
                 case 1:s_vecino=unoSwapB(s_vecino);
                   break;
                 case 2:s_vecino=cambioVecino(s_vecino,mDistancias);
                   break
                 case 3:s_vecino=insercionB(s_vecino);
                   break;
                 case 4:s_vecino=insercionB(s_vecino);
                   break;
           }
return(s_vecino);

}
```

```
/********************* ESTRUCTURAS DE VECINDAD **************/
solucion unoSwapB(solucion p)
{
            int num1=0,num2=0,aux=0,aux2=0;
            while(num1 == num2)
            {
                        num1=rand()%p.numClientes ;
                        num2=rand()%p.numClientes;
   }
            aux=p.ruta[num1];
            aux2=p.ruta[num2];
            p.ruta[num1]=p.ruta[num2];
            p.ruta[num2]=aux;

            return(p);
}

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

solucion cambioVecino(solucion p,float mDistancias[LIMITE_MAX+2][LIMITE_MAX+2])
{
   int min,num1,num2,i,aux,vecino,pos=1;
   num1=(rand()%(p.numClientes-2))+1;
   min=mDistancias[num1+1][1];
   for(i=1;i<=p.numClientes;i++)
   {
          if(i != (num1+1) && mDistancias[num1+1][i]<min)
            {
                min=mDistancias[num1+1][i];
                pos=i;
            }
   }
  num2=pos-1;
  if(rand()%2)
        vecino=(num1-1) ;
  else
        vecino=(num1+1);
  aux= p.ruta[vecino];
  p.ruta[vecino]=p.ruta[num2];
  p.ruta[num2]=aux;
 return(p);
}

///***********************************************
solucion insercionB(solucion p)//inserta un cliente y recorre toda la fila
{
      int c,rHueco,huecoInf,huecoSup,hueco,rSale,saleInf,saleSup,sale,aux,i;
      int k;
          do{
              rSale=rand()%p.vh;
              rHueco=rand()%p.vh;
             }while(rSale==rHueco);
          huecoInf=p.limiteInferior[rHueco];
          huecoSup=p.limiteSuperior[rHueco];
          saleInf=p.limiteInferior[rSale];
          saleSup=p.limiteSuperior[rSale];
```

```
    hueco=huecoInf+rand()%((huecoSup-huecoInf)+1);
    sale=saleInf+rand()%((saleSup-saleInf)+1);
    if (hueco < sale)
            {
                aux= p.ruta[sale];//le damos el valor de num2 a aux para que se lo guarde
                for (c=sale; c > hueco; c--)
                p.ruta[c]= p.ruta[c-1];
                p.ruta[hueco]=aux;
             }//del if
   else
            {
                aux= p.ruta[sale];//le damos el valor de num2 a aux para que se lo guarde
                for (c=sale; c < hueco; c++)
                p.ruta[c]= p.ruta[c+1];
                p.ruta[hueco]=aux;
            }//del else

   if (rHueco < rSale)
            {
               p.limiteSuperior[rHueco]=huecoSup + 1;
               for(i=rHueco + 1  ;i< rSale   ;i++)
                {
                    p.limiteInferior[i]= p.limiteInferior[i] + 1;
                    p.limiteSuperior[i]= p.limiteSuperior[i] + 1;
                }//del for
               p.limiteInferior[rSale]=saleInf + 1;
            }//del if
   else
            {
                p.limiteSuperior[rSale]=saleSup - 1;
                for(i=rSale + 1  ;i< rHueco   ;i++)
                   {
                      p.limiteInferior[i]=p.limiteInferior[i]- 1;
                      p.limiteSuperior[i]= p.limiteSuperior[i]- 1;
                    }
                p.limiteInferior[rHueco]=huecoInf - 1;
            }//del else
  return(p);
}//de la función


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

int imprimeRutaOptimizada(double costoInicial, solucion s_mejorRS, double tempInicial,double
tempFin, double alfa, int longM, float tiempo, int rank){
  FILE *ruta_txt;
  char rutas[20];
  int i,k;//contador de clientes
  sprintf(rutas, "Mejor_ruta%d.txt", rank);
   ruta_txt = fopen(rutas,"w");
   if (ruta_txt==NULL)
       return -1;
   else
   {
      fprintf(ruta_txt,"%d\n",s_mejorRS.vh);
```

```
        for (i=0,k=0; i<s_mejorRS.numClientes; i++)
         {
           fprintf(ruta_txt,"%2d ",s_mejorRS.ruta[i]);
           if(s_mejorRS.limiteSuperior[k]==i)
             {
                fprintf(ruta_txt,"\n");
                k++;
             }
         }//del for
        fprintf(ruta_txt,"\n");
        for (k=0; k<s_mejorRS.vh;k++)
         {
            fprintf(ruta_txt,"%d ",s_mejorRS.limiteInferior[k]);
         }
        fprintf(ruta_txt,"\n");

        for (k=0; k<s_mejorRS.vh;k++)
         {
           fprintf(ruta_txt,"%d ",s_mejorRS.limiteSuperior[k]);
         }

         fprintf(ruta_txt,"\n");
         fprintf(ruta_txt,"\nCargas  \n");
         for (k=0; k<s_mejorRS.vh;k++)
         {
           fprintf(ruta_txt,"%lf ",s_mejorRS.cargaVh[k]);
         }
         fprintf(ruta_txt,"\n");
         //fprintf(ruta_txt,"\nCosto inicial  %.2f\n",costoInicial);
         fprintf(ruta_txt,"\nSolucion final  %.2f\n",s_mejorRS.costoSolucion);
         fprintf(ruta_txt,"\ntemperaturaIni %f  temperaturaFin %f  alfa %.3f  Cadena Markov
%d\n",tempInicial,tempFin, alfa,longM);
         fprintf(ruta_txt,"\ntiempo en segundos  %.2f\n",tiempo);

       // fprintf(ruta_txt,"%d ",0);
    }//del else
    fclose(ruta_txt);
    return 0;
}//fin imprimeRutaOptimizada

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