CENTRO DE CIENCIAS BÁSICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

TESIS

OPTIMIZACIÓN DE UN ALGORITMO DE COLONIA DE HORMIGAS MEDIANTE CUDA PARA RESOLVER PROBLEMAS DE RUTEO DE VEHÍCULOS

PRESENTA
Alfonso Recio Hernández

PARA OBTENER EL GRADO DE MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

TUTOR(ES)
D.C.C. Julio Cesar Ponce Gallegos
Dr. Alejandro Padilla Díaz

COMITÉ TUTORAL
Dra. Aurora Torres Soto

Aguascalientes, Ags., a 28 de Noviembre de 2014

UNIVERSIDAD AUTONOMA
DE AGUASCALIENTES

FORMATO DE CARTA DE VOTO APROBATORIO

M. en C. José de Jesús Ruiz Gallegos
DECANO (A) DEL CENTRO DE CIENCIAS

PRESENTE

Por medio del presente como Tutor designado del estudiante **ALFONSO RECIO HERNÁNDEZ** con ID 159773 quien realizó la tesis titulada: **OPTIMIZACIÓN DE UN ALGORITMO DE COLONIA DE HORMIGAS MEDIANTE CUDA PARA RESOLVER PROBLEMAS DE RUTEO DE VEHICULOS**, y con fundamento en el Artículo 175, Apartado II del Reglamento General de Docencia, me permito emitir el **VOTO APROBATORIO**, para que él pueda proceder a imprimirla, así como continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

ATENTAMENTE
"Se Lumen Proferre"
Aguascalientes, Ags., a 10 de Noviembre de 2014.

D.C.C. Julio Cesar Ponce Gallegos
Tutor de tesis

c.c.p.- Interesado
c.c.p.- Secretaría de Investigación y Posgrado
c.c.p.- Jefatura del Depto. de Ciencias de la Computación
c.c.p.- Consejero Académico
c.c.p.- Minuta Secretario Técnico

UNIVERSIDAD AUTONOMA
DE AGUASCALIENTES

FORMATO DE CARTA DE VOTO APROBATORIO

M. en C. José de Jesús Ruiz Gallegos
DECANO (A) DEL CENTRO DE CIENCIAS

PRESENTE

Por medio del presente como Tutor designado del estudiante **ALFONSO RECIO HERNÁNDEZ** con ID 159773 quien realizó la tesis titulada: **OPTIMIZACIÓN DE UN ALGORITMO DE COLONIA DE HORMIGAS MEDIANTE CUDA PARA RESOLVER PROBLEMAS DE RUTEO DE VEHICULOS**, y con fundamento en el Artículo 175, Apartado II del Reglamento General de Docencia, me permito emitir el **VOTO APROBATORIO**, para que él pueda proceder a imprimirla, así como continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

ATENTAMENTE
"Se Lumen Proferre"
Aguascalientes, Ags., a 10 de Noviembre de 2014.

Dr. Alejandro Padilla Díaz.
Tutor de tesis

c.c.p.- Interesado
c.c.p.- Secretaría de Investigación y Posgrado
c.c.p.- Jefatura del Depto. De Ciencias de la Computación
c.c.p.- Consejero Académico
c.c.p.- Minuta Secretario Técnico

UNIVERSIDAD AUTONOMA
DE AGUASCALIENTES

FORMATO DE CARTA DE VOTO APROBATORIO

M. en C. José de Jesús Ruiz Gallegos
DECANO (A) DEL CENTRO DE CIENCIAS

PRESENTE

Por medio del presente como Tutora designada del estudiante **ALFONSO RECIO HERNÁNDEZ** con ID 159773 quien realizó la tesis titulada: **OPTIMIZACIÓN DE UN ALGORITMO DE COLONIA DE HORMIGAS MEDIANTE CUDA PARA RESOLVER PROBLEMAS DE RUTEO DE VEHICULOS**, y con fundamento en el Artículo 175, Apartado II del Reglamento General de Docencia, me permito emitir el **VOTO APROBATORIO**, para que él pueda proceder a imprimirla, así como continuar con el procedimiento administrativo para la obtención del grado.

Pongo lo anterior a su digna consideración y sin otro particular por el momento, me permito enviarle un cordial saludo.

ATENTAMENTE
"Se Lumen Proferre"
Aguascalientes, Ags., a 10 de Noviembre de 2014.

Dra. Aurora Torres Soto.
Asesora de tesis

c.c.p.- Interesado
c.c.p.- Secretaría de Investigación y Posgrado
c.c.p.- Jefatura del Depto. De Ciencias de la Computación
c.c.p.- Consejero Académico
c.c.p.- Minuta Secretario Técnico

UNIVERSIDAD AUTONOMA DE AGUASCALIENTES
CENTRO DE CIENCIAS BÁSICAS

L. I. ALFONSO RECIO HERNÁNDEZ
ALUMNO (A) DE LA MAESTRÍA EN CIENCIAS
CON OPCIÓN A LA COMPUTACIÓN,
P R E S E N T E .

Estimado (a) alumno (a) Recio:

Por medio de este conducto me permito comunicar a Usted que habiendo recibido el voto aprobatorio de su tutor de tesis titulada: ***"OPTIMIZACIÓN DE UN ALGORITMO DE COLONIA DE HORMIGAS MEDIANTE CUDA PARA RESOLVER PROBLEMAS DE RUTEO DE VEHÍCULOS",*** hago de su conocimiento que puede imprimir dicho documento y continuar con los trámites para la presentación de su examen de grado.

Sin otro particular me permito saludarle muy afectuosamente.

A T E N T A M E N T E
Aguascalientes, Ags., 12 de noviembre de 2014
"SE LUMEN PROFERRE"
EL DECANO

M. en C. JOSÉ DE JESÚS RUIZ GALLEGOS

UNIVERSIDAD AUTONOMA DE AGUASCALIENTES
CENTRO DE CIENCIAS BÁSICAS

c.c.cp.- Interesado
JJRG,mjda

UNIVERSIDAD AUTÓNOMA DE AGUASCALIENTES

POSGRADOS UAA

DICTAMEN DE REVISIÓN DE LA TESIS / TRABAJO PRÁCTICO

DATOS DEL ESTUDIANTE	
NOMBRE: Alfonso Recio Hernández	ID (No. de Registro): 159773
PROGRAMA: Maestría en Ciencias con Opción a la Computación	ÁREA: Inteligencia Artificial
TUTOR/TURORES: D.C.C. Julio Cesar Ponce Gallegos, Dr. Alejandro Padilla Díaz y Dra. Aurora Torres Soto.	
TESIS (X) TRABAJO PRÁCTICO ( )	
OBJETIVO: Análisis y la implementación de una metaheurística (Algoritmo en colonia de hormigas) en una arquitectura de programación paralela (CUDA).	
DICTAMEN	
CUMPLE CON CRÉDITOS ACADÉMICOS:	(X)
CONGRUENCIAS CON LAS LGAC DEL PROGRAMA:	(X)
CONGRUENCIA CON LOS CUERPOS ACADÉMICOS:	(X)
CUMPLE CON LAS NORMAS OPERATIVAS:	(X)
CONINCIDENCIA DEL OBJETIVO CON EL REGISTRO:	(X)

Aguascalientes, Ags. a 14 de Noviembre de 2014

FIRMAS

CONSEJERO ACADÉMICO DEL ÁREA

SECRETARIO TÉCNICO DEL POSGRADO

SECRETARIO DE INVESTIGACIÓN Y POSGRADO

Código: FO-040200-23
Revisión: 00
Emisión: 21/02/11

Agradecimientos

Este agradecimiento es para toda mi familia, pareja, amigos(as) y profesores(as) quienes fueron de gran apoyo durante todo este tiempo, gracias por su tiempo, paciencia y palabras oportunas para no permitirme claudicar y poder alcanzar este proyecto personal.

A mi Madre Celia, Hermana Ana, Tíos Octavio y Arturo por el amor, apoyo incondicional y demostrarme la gran fe que tienen en mí. A mi Papá Alfonso aunque ya no se encuentra en este mundo físicamente, fuiste un excelente ejemplo de vida y superación a seguir.

A mi Pareja Carolina por ser parte importante de mi vida, y acompañarme durante todo este arduo camino, por compartir momentos de alegrías y frustración, de éxitos y fracasos. Siento mucha alegría saber que ambos hemos alcanzado nuestras metas y terminado nuestros respectivos proyectos personales.

A mis amigos(as) Roberto, Norma, Omar, Jorge, Juan José, Priscila, Alejandro, Claudia, Magda, “los chavos del dominó” y demás amistades que omito con la idea de no excluir a nadie. A quienes, a pesar del distanciamiento y la disminución en la frecuencia de nuestra convivencia, en cada reencuentro es como si no hubiese pasado más de un día y en cada abrazo se renuevan nuestros lazos de amistad.

A mis Co-Tutores el D.C.C. Julio Cesar Ponce Gallegos y Dr. Alejandro Padilla Díaz y a mi Asesora la Dra. Aurora Torres Soto, al Dr. José Alberto Hernández Aguilar por su valiosa guía, tiempo y asesoramiento en el desarrollo del proyecto y la escritura de este trabajo.

A la Universidad Autónoma de Aguascalientes y al Consejo Nacional de Ciencia y Tecnología (CONACYT), agradezco la oportunidad de permitirme desarrollar, apoyar y culminar este proyecto, que refleja un escalón más en mi formación profesional.

Son muchas las personas que de una manera directa o indirecta, consciente o inconsciente han contribuido a mi desarrollo y formación profesional, laboral, educativa y personal. Sin importar en donde estén quiero darles las gracias y expresarles mi gran cariño, por formar parte de mí y por todo lo que me han brindado de ustedes.

A todos y a todas, Muchas Gracias.

"…I rejected those answers. Instead, I chose something different. I chose the impossible. I chose.. Rapture…"

Andrew Ryan,, Bioshock.

Índice General

Índice de Tablas

Índice de Figuras

Índice de Ecuaciones

Índice de Códigos

Índice de Pseudocódigos

Índice de Restricciones

Resumen

En esta tesis se muestra cómo pueden los algoritmos de colonias de hormigas resolver problemas de optimización de gran complejidad que requieren de grandes capacidades de cómputo, los cuales pueden ser combinados con nuevas arquitecturas de programación en paralelo mediante el uso de tarjetas gráficas (GPU´s) para tratar de mejorar su rendimiento.

Al inicio se encuentra una introducción, que es una explicación general de los principales elementos que integran el método propuesto, los cuales a través de los diversos capítulos se irán desarrollando de manera más extensa. El capítulo 1 es un marco teórico conformado por la justificación, la hipótesis y los objetivos generales y particulares que se resuelven. Posteriormente, en el capítulo 2 encontraremos una introducción a las técnicas metaheurísticas más conocidas como son los algoritmos genéticos, recocido simulado, colonias de hormigas, entre otras. En el capítulo 3 se expone el tema de complejidad computacional, la dificultad de resolver problemas computacionales y la manera en que algunos problemas consumen grandes cantidades de almacenamiento y capacidad de procesamiento.

En el capítulo 4 se encontrará información referente al problema de ruteo de vehículos, en que consiste este problema, su formulación matemática y algunas áreas de aplicación. En el capítulo 5 se explican los algoritmos de optimización basados en colonias de hormigas, la descripción de su estructura y funcionamiento. En el capítulo 6 se habla sobre la arquitectura de programación CUDA, de la cual nos apoyamos para implementar algunas de las funciones para su funcionamiento en paralelo.

En el capítulo 7, se explica el algoritmo de solución propuesto, el análisis de las funciones que fueron convenientes paralelizar y su implementación en la nueva arquitectura. En el capítulo 8 se detalla el equipo de cómputo empleado para realizar las pruebas, así como también, el software instalado para desarrollar el algoritmo en paralelo. En el capítulo 9 se explican las pruebas realizadas con el método de solución, los parámetros utilizados para realizar pruebas factoriales, y la manera en que se obtuvieron los resultados, para finalmente ser analizados.

En la última sección se explica el análisis estadístico hecho a los resultados, con la finalidad de permitirnos confirmar o rechazar algunas de las hipótesis formuladas, y también formular posibles trabajos futuros.

Abstract

This thesis shows how can the ant colony algorithms to solve optimization problems of great complexity that require large computing capabilities, which can be combined with new architectures of parallel programming using graphics cards (GPU's) to try to improve their performance.

At the beginning is an introduction, a general explanation of the main elements that make up the proposed method, which through the various chapters will be developed more extensively. Chapter 1 is a theoretical framework consisting rationale, assumptions and general and specific objectives that are resolved. Later, in chapter 2 we find an introduction to popular metaheuristics such as genetic algorithms, simulated annealing, ant colonies, among other metaheuristic techniques. In chapter 3 the issue of computational complexity, the difficulty of solving computational problems and how some problems consume large amounts of storage and processing capacity is exposed.

In chapter 4 will find information about the vehicle routing problem, that is the problem, mathematical formulation and some application areas. In chapter 5 the optimization algorithms based on ant colonies are explained, describing their structure and function. In Chapter 6 we talk about the architecture of CUDA programming, which we rely to implement some functions to operate in parallel.

In Chapter 7, the proposed solution algorithm, analysis of the functions that were convenient parallelize and its implementation in the new architecture is explained. Computer equipment used for testing is detailed in chapter 8, as well as the software installed to develop the algorithm in parallel. In Chapter 9 tests with the solution method are explained, the parameters used for factorial testing, and how the results were obtained, finally to be analyzed.

In the last section the statistical analysis done on the results, in order to allow us to confirm or reject some of the assumptions made, and formulate possible future work is explained.

Introducción

Con el motivo de hacer una contribución en la investigación en el área de las meta-heurísticas y para alcanzar una mejora en ellas, se realizó la implementación de un algoritmo con funciones que se ejecutan de forma paralela en CUDA, con la meta de intentar resolver problemas con una gran complejidad computacional, como son en este caso los problemas de ruteo de vehículos, obteniendo soluciones que se aproximen a lo óptimo y de una manera más veloz, que si se realizaran con programación de manera secuencial o con métodos exhaustivos.

El problema de ruteo de vehículos o como se le conoce en la literatura VRP (Vehicule Routing Problem) es uno de los problemas de optimización combinatoria más difíciles de resolver, su definición data de los años 50's en un trabajo realizado por Dantzig y Ramser, en donde establecieron la formulación matemática del problema para resolver la entrega de combustible a las estaciones de servicio (Dantzig G. B., 1959).

Las principales aplicaciones de un Problema de Ruteo de Vehículos son en logística y distribución como: la entrega y recolección de materiales, el transporte de personal, enrutamiento de redes computacionales, planificación de rutas de auxilio en caso de desastres naturales, distribución de correspondencia, etc.

Para solucionar este tipo de problemas el método que se presenta es utilizando un Algoritmo computacional conocido como Optimización mediante Colonias de Hormigas, es un algoritmo bioinspirado basado en la manera de buscar trayectos más cortos entre su hormiguero y una fuente de alimento, estas generan rutas al depositar una sustancia química denominada feromona, los caminos más prometedores van acumulando mayor concentración debido al paso más frecuente de las hormigas. En este algoritmo cada hormiga es vista como un agente artificial el cual va construyendo su ruta mediante una fórmula probabilística y simula la evaporación e incrementos de feromona, intentando

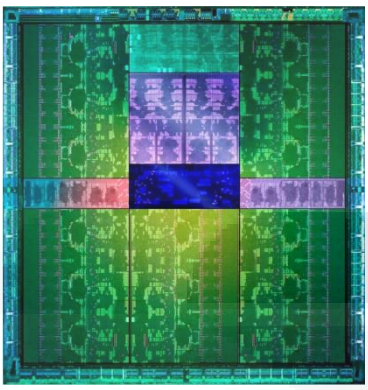
encontrar una solución óptima en un tiempo polinomial determinado (Dorigo & Stützle, Ant Colony Optimization., 2004.)

La computación de propósito general en unidades de procesamiento de gráficos o GPU´s intenta aprovechar las capacidades de procesamiento de las unidades de procesamiento grafico actuales. En esta área se tiene la arquitectura de programación llamada CUDA desarrollada por la compañía NVIDIA, esta arquitectura maneja extensiones para los lenguajes C / C++ principalmente. Estas extensiones permiten la programación de funciones o núcleos que se pueden ejecutar de manera simultánea en la GPU, obteniendo resultados de manera más rápida que si se hubiera ejecutado solamente en la CPU (Plataforma de Computacion Paralela CUDA.).

El algoritmo de optimización mediante colonias de hormigas tiene diferentes variantes que han sido implementadas en diferentes trabajos de manera secuencial, llevando a cabo su ejecución mediante la CPU (Gambardella, Taillard, & Agazzi, 1999.) (Bullnheimer, Hartl, & Strauss, An Improved Ant System Algorithm for the Vehicle Routing Problem., 1999.) (Dorigo & Gambardella, Ant Colonies for the Travelling Salesman Problem., 1997.). Hoy en día existen otras arquitecturas como CUDA que permiten el procesamiento de operaciones en paralelo en la GPU para hacer más veloz su ejecución.

En este documento se presenta un método para solucionar problemas de ruteo de vehículos con capacidad limitada, el método que se propone está basado en un procedimiento conocido como Rutear Primero – Asignar Después, utilizando un Algoritmo con Colonia de Hormigas con funciones modificadas para ejecutarse de manera paralela en la GPU. Y además se muestran algunos resultados obtenidos de experimentos, utilizando instancias de prueba con soluciones óptimas publicadas anteriormente por otros autores.

1. Marco Teórico



1.1 Planteamiento del Problema.

Dentro del área de ciencias de la computación existen una gran variedad de problemas que buscan ser solucionados mediante el uso de algoritmos computacionales, una de las clases de problemas que se pueden encontrar son los llamados Problemas de Ruteo de Vehículos (Vehicle Routing Problem o VRP). Los autores de esta clase de problemas fueron George Dantzig y John Ramser en el cual intentan encontrar las rutas óptimas para un grupo de vehículos los cuales dan servicio a un gran número de clientes abasteciéndose mediante un depósito (Dantzig G. B., 1959). Este problema tiene algunas variantes en las que se busca satisfacer condiciones como mejorar el tiempo del trayecto de los vehículos, obtener la distancia mínima recorrida por cada uno de los vehículos, la capacidad limitada de cada vehículo, los clientes manejan lapsos de tiempo para realizar sus entregas, los vehículos cuentan con diferentes capacidades de carga, manejar más de un depósito (almacenes o bodegas), entre muchas otras condiciones que se pueden mezclar y dar como resultado problemas más complicados de resolver. Se ha intentado buscar la solución de esta clase de problemas mediante el uso de métodos matemáticos como la programación lineal con la cual se han obtenido soluciones cercanas a la óptima, también se han utilizado algoritmos computacionales para encontrar una solución óptima al problema, estos métodos son efectivos para un número reducido de clientes, sin embargo si el número de clientes a

visitar por las flotas de vehículos aumenta, se refleja en un aumento exponencial de las combinaciones de clientes y los vehículos que les darán abastecimiento; por lo que encontrar soluciones óptimas se hace más complicado y tardado.

Debido al aumento de la complejidad para solucionar estos problemas, se ve la necesidad de encontrar equipos de cómputo más potentes, que cuenten con grandes cantidades de memoria y almacenamiento, con el objetivo de lograr mejoras en los tiempos de procesamiento necesarios para encontrar una solución óptima o por lo menos una solución aceptable y lo más cercana posible a la óptima.

Por otra parte existen técnicas y algoritmos computacionales que están enfocados a solucionar problemas de optimización conocidos como metaheurísticas, dentro del campo de las metaheurísticas encontramos los llamados algoritmos de colonias de hormigas, éstos realizan una imitación del comportamiento de las hormigas reales en su búsqueda de alimento, durante esta tarea las hormigas van dejando en su trayecto un rastro de una sustancia llamada feromona que sirve para comunicarse con otros individuos de su colonia y que les ayuda a intercambiar información como: si se ha encontrado alimento o no, la calidad del alimento encontrado, y una de las más importantes es la calidad y las distancias de su trayecto, este intercambio de información mediante feromonas ayuda a las hormigas a encontrar los caminos más cortos entre su hormiguero y la fuente de alimentos encontrada (Dorigo & Stützle, Ant Colony Optimization., 2004.) (Brownlee, 2011.). Con lo anterior se puede ver que los algoritmos de colonia de hormigas buscan siempre la optimización de resultados, para esto se tiene como entrada de información un archivo conocido como instancia el cual contiene la información básica del problema como el tipo de problema que se está tratando, el número de puntos a visitar, las coordenadas de ubicación de cada punto, etc. Enseguida el algoritmo comienza el procesamiento de los datos mediante el uso de fórmulas probabilísticas dentro de un proceso iterativo, y como resultado intenta darnos la mejor ruta encontrada en un lapso de tiempo transcurrido, un determinado número de iteraciones o alguna función que siga un criterio de convergencia. Este funcionamiento junto con algunas modificaciones les permite tener aplicaciones en áreas como las de

transporte, distribución y logística algunos ejemplos son los problemas del agente viajero, problemas de enrutamiento de redes de computadoras, diseño de circuitos electrónicos o problemas de ruteo de vehículos, transporte de personas y materiales, entre otras muchas aplicaciones que se pueden encontrar, siempre y cuando se modele correctamente el problema y la forma en que se muestren los resultados para realizar comprobaciones y comparaciones con otros trabajos y métodos del estado del arte (Hoos & Stützle, 2004.) (Dorigo & Stützle, Ant Colony Optimization., 2004.) (Dorigo & Gambardella, Ant Colonies for the Travelling Salesman Problem., 1997.) (Gambardella, Taillard, & Agazzi, 1999.) (Bullnheimer, Hartl, & Strauss, Applying the Ant System to the Vehicle Routing Problem., 1997.).

Resulta de gran importancia investigar y aprender a modelar los problemas de ruteo de vehículos ya que muchas de sus aplicaciones son útiles en herramientas de uso diario o en el funcionamiento y mejoramiento de fábricas y empresas, gracias a estas aplicaciones se pueden encontrar rutas de transporte de personal o de estudiantes que cubran mayores áreas y realicen recorridos más rápidos. Las fábricas y empresas se pueden ver beneficiadas al momento de crear sus rutas de entrega de mercancía a clientes ahorrando el número de vehículos y con ello disminuir gastos de combustible, disminuir del número de choferes para hacer las entregas, ahorro en el tiempo de entregas de mercancía y de igual manera en fábricas de manufactura se pueden hacer estudios de cuál sería la mejor ubicación para las máquinas y materias primas, para reducir el número de movimientos y los tiempos de espera, realizar adaptaciones en la maquinaria, llevar mejores controles en los embarques, etc. En resumen todas estas aplicaciones se pueden ver reflejadas en ganancias de tiempos y dinero que son los objetivos principales buscados por la mayoría de las empresas y fábricas (Adam & Ebert, 1978.)

Como se explicó anteriormente un factor importante y también un problema es buscar la reducción de tiempos de procesamiento para encontrar soluciones cercanas a lo óptimo para un determinado problema, pero no siempre se va a encontrar con problemas fáciles de resolver con un número de clientes pequeño, por el contrario existen ejemplos de

compañías que trabajan a nivel mundial y que manejan una cantidad enorme de clientes por lo que su tiempo de procesamiento mediante una computadora y un algoritmo que busque su solución crece de manera exponencial y puede tardar desde varias horas, días o inclusive hasta meses completos buscando una solución (Bäck, Fogel, & Michalewicz, 1997.), debido a este problema de tiempo que se presenta cuando el tamaño de los problemas crece se busca utilizar herramientas que ayuden a reducir el tiempo de solución, para esto una de las herramientas que se utilizará para la realización de este proyecto es una arquitectura llamada CUDA creada por la compañía NVIDIA (Plataforma de Computacion Paralela CUDA.). CUDA es una plataforma que permite la programación paralela haciendo uso de la potencia de procesamiento y de la memoria que tienen las actuales tarjetas de video (Graphics Processing Unit - GPU), las cuales cuentan con un número de núcleos de procesamiento mucho mayor comparados a los de una CPU. La plataforma CUDA permite utilizar las tarjetas gráficas no solamente para el procesamiento de gráficos sino que ahora se pueden utilizar también para ejecutar y procesar instrucciones e información de manera paralela, efectuando una misma tarea a diferentes datos reduciendo así el tiempo de procesamiento en operaciones con altos costos aritméticos (Sanders & Kandrot, 2011), estas mejoras en el rendimiento de software han generado un extenso catálogo de aplicaciones impulsadas por la arquitectura CUDA (GPU-Accelerated Applications.).

Explicadas las ventajas del procesamiento en paralelo bajo la arquitectura de programación CUDA, y la desventaja respecto a los tiempos que les toman a los algoritmos de optimización resolver problemas con explosión combinatoria que manejan una extensa cantidad de puntos o clientes a visitar, se hizo la implementación en la arquitectura CUDA de un algoritmo de optimización como es el algoritmo de colonias de hormigas, adaptando sus funciones en paralelo para buscar soluciones rápidas y aceptables a problemas de ruteo de vehículos mejorando los tiempos y la cantidad de información a analizar en comparación con otros algoritmos procesados de manera tradicional mediante la CPU.

1.2 Justificación.

Los problemas de optimización como su nombre lo indica tienen como objetivo buscar ahorros, éstos aplicados en situaciones y procesos reales de las diferentes ramas industriales se pueden ver reflejados en diferentes sectores relacionados a la producción o las inversiones que se necesitan para brindar un servicio, como son ahorros de tiempo, de dinero, de personal involucrado en las áreas de logística, de construcción, de producción, y de la entrega de un servicio. Es importante señalar que se busca obtener todos estos beneficios sin sacrificar la calidad de los productos y evitando realizar grandes inversiones de dinero.

Cuando se intenta resolver problemas de optimización combinatoria cuyo espacio de soluciones puede ser extenso o tener que cumplir con varias condiciones, el principal problema que se presenta es el tiempo necesario de procesamiento para encontrar una o varias soluciones al problema. Una de las formas en que se busca atacar este inconveniente y reducir el tiempo de procesamiento es la de implementar modificaciones que mejoren las funciones del algoritmo para resolver problemas de ruteo de vehículos. La otra medida que se piensa utilizar es la implementación del algoritmo de colonia de hormigas en una arquitectura de programación en paralelo como CUDA, tomando en cuenta su aplicación en problemas de propósito general y enfocar los recursos de la GPU, como su memoria y los núcleos de procesamiento para aumentar las cantidades de información procesada y la velocidad de la ejecución del algoritmo, para alcanzar un tiempo de ejecución menor en comparación con un algoritmo que se ejecute solamente en la CPU, y mediante estas modificaciones esperar obtener buenas soluciones cercanas o iguales al óptimo.

1.3 Hipótesis.

Es posible desarrollar e implementar un método para solucionar problemas de ruteo de vehículos basándose en una metaheurística y utilizando un algoritmo de colonia de hormigas bajo una arquitectura en paralelo como CUDA.

Es posible encontrar soluciones buenas o de mejor calidad a las encontradas en la literatura de problemas de ruteo de vehículos mediante un método metaheurístico apoyado de un algoritmo de optimización cuyas funciones e instrucciones sean implementadas en una arquitectura de programación paralela.

1.4 Objetivo General.

El objetivo de este trabajo, fue la implementación en una arquitectura de programación paralela de un algoritmo de colonia de hormigas para resolver problemas de ruteo de vehículos. Para implementar este método se necesitó realizar un análisis de las funciones que tenían posibilidades de implementarse bajo una arquitectura en paralelo, y posteriormente se realizó la adaptación en paralelo de estas funciones seleccionadas, utilizando la plataforma de programación CUDA. Con el objetivo de conocer las ganancias que se pueden obtener referentes a los costos computacionales, la calidad de las soluciones encontradas, identificar el parámetro o grupo de parámetros con los que se alcancen las mejores soluciones a los diversos problemas, los tiempos de ejecución o la cantidad de información analizada durante la búsqueda de soluciones también se realizaron una serie de pruebas con el algoritmo y se analizaron los resultados obtenidos.

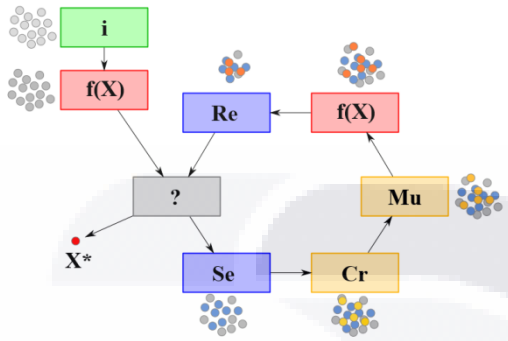
Una vez que se realizó el estudio y la implementación del algoritmo en paralelo, posteriormente se realizaron una serie de pruebas y experimentos para encontrar los mejores parámetros con los cuales el algoritmo encuentra los mejores resultados, y bajo estos parámetros realizar una serie de ejecuciones para detectar los mejores resultados obtenidos por este método.

1.5 Objetivos Específicos.

- Estudiar y desarrollar algoritmos de colonia de hormigas que trabajen de manera secuencial.
- Seleccionar un algoritmo de colonia de hormigas cuyas funciones sea posible implementar para su ejecución en paralelo.

- Identificar las funciones del algoritmo de colonia de hormigas que sean posible implementarse en paralelo.
- Desarrollar las funciones del algoritmo de colonia de hormiga para su implementación en la arquitectura CUDA.
- Realizar el diseño de experimentos con instancias utilizadas en el estado de arte.
- Verificar los resultados obtenidos por el método heurístico que se está proponiendo con resultados en el estado del arte.
- Realizar pruebas estadísticas sobre los resultados de los experimentos para detectar los parámetros que generen los mejores resultados.

2. Técnicas Metaheurísticas



2.1 Introducción a las Técnicas Metaheurísticas.

Por la década de los años 70´s surgieron nuevos tipos de algoritmos de aproximación que hacían uso de métodos heurísticos simples con estructuras de datos de alto nivel, estas dos herramientas permitían la exploración eficiente y efectiva en un espacio de búsqueda. A este nuevo tipo de algoritmos se les llamó metaheurísticas, este término fue introducido por Glover en (Glover & Kochenberger, Handbook of Metaheuristics. (Hardcover)., 2003.). El término metaheurística proviene de dos palabras griegas. *Heuristic* que proviene del verbo *heuriskein* que significa "*encontrar*", y el sufijo *meta* que significa "*más allá, en un nivel superior*". Algunas descripciones de las metaheurísticas permiten ver algunas de sus propiedades como son: las estrategias utilizadas para guiar el proceso de búsqueda, su objetivo de explorar eficientemente el espacio de búsqueda con el fin de encontrar las soluciones óptimas o las más cercanas posibles, están constituidas por técnicas que van desde procedimientos de búsquedas locales simples, hasta complejos procesos de aprendizaje estocásticos.

Estas técnicas no están vinculadas a un problema en específico y sus conceptos se pueden describir de manera abstracta, para facilitar la modelación de diversos problemas reales (Alba, 2005).

2.2 Aplicación de las Técnicas Metaheurísticas.

Las técnicas metaheurísticas se pueden encontrar en cualquier lugar en donde las actividades se busquen optimizar, en el mundo de la industria se tienen un gran número de procesos que se buscan mejorar con el propósito de ahorrar recursos y la mayoría de éstos se pueden modelar como problemas de optimización.

La optimización puede consistir en el ahorro de tiempos, dinero, personal, vehículos, distancias, suministros, etc. Un ejemplo de optimización podría ser la programación de horarios de una universidad, en este problema se manejan variables como la hora de las clases, las aulas disponibles, así como la cantidad de profesores y su disponibilidad de horario. Conforme van aumentando el número de variables en el problema, se hace más complejo encontrar la programación de horarios óptima. Muchos problemas de optimización de diferentes áreas son complejos y difíciles de solucionar mediante métodos exactos en un tiempo específico, en estos casos la mejor opción que se tiene es utilizar métodos o algoritmos de aproximación los cuales intentan encontrar buenas soluciones a un problema, en un menor tiempo.

Entre los algoritmos de aproximación tenemos las llamadas metaheurísticas, éstas son capaces de encontrar la solución a problemas muy complejos, al principio comienzan explorando un gran número de posibles soluciones, mediante el uso de funciones se van optimizando y evaluando con el objetivo de ir reduciendo el número de soluciones y llegar al punto de generar una buena solución o en el mejor caso encontrar la solución óptima al problema. Las metaheurísticas tienen tres propósitos principales: resolver problemas rápidamente, resolver problemas grandes, producir algoritmos estables que al mismo tiempo son fáciles de implementar y con una gran flexibilidad.

Las técnicas metaheurísticas son una rama de las ciencias computacionales que mezclan las matemáticas aplicadas, formulas probabilísticas, y teorías de la complejidad computacional y son muy utilizadas en áreas como la inteligencia artificial, computación inteligente, cómputo suave, programación matemática e investigación de operaciones. La mayoría de las técnicas metaheurísticas son imitaciones de fenómenos naturales.

2.3 Tipos de Técnicas Metaheurísticas.

Así como podemos encontrar en diversas áreas un gran número de problemas de optimización, de la misma manera se han propuesto y generado diferentes metaheurísticas. Algunas de las técnicas metaheurísticas más conocidas son:

Búsqueda Tabú o Tabú Search (TS) se le atribuye a Fred Glover por sus publicaciones en los años 1989 y 1990 (Glover, Tabu Search, Part 1., 1989.) (Glover, Tabu Search, Part 2., 1990.), la búsqueda tabú aumenta el rendimiento del método de búsqueda local utilizando estructuras de memoria. Cuando se determina una solución, ésta es marcada como "tabú" para que el algoritmo no vuelva a visitar esa posible solución.

Procedimiento de búsqueda Adaptativa Aleatoria Avariciosa o Greedy Adaptive Search Procedure (GRASP) esta metaheurística fue introducida por Feo y Resende en el año 1989 (Feo & Resende, 1989.). Como una metaheurística de propósito general. GRASP es un procedimiento que consiste en dos fases la primera va construyendo iterativamente una solución factible de partida y una segunda fase de mejora en donde se realiza un proceso de búsqueda local a partir de la solución inicial.

Algoritmos Genéticos o Genetic Algorithms (GA) se le atribuye a John Holland por su libro "Adaptation in Natural and Artificial Systems" publicado en el año 1975 (Holland, 1992.). Esta metaheurística consiste en una población de soluciones candidatas al problema, cada solución se evalúa para obtener un valor que mide su validez como solución

al problema, y en función de este valor se les otorgará una probabilidad para reproducirse, y además se realizarán cruces y mutaciones en estas soluciones.

Recocido Simulado o Simulated Annealing (SA) la primera aplicación de este algoritmo fue hecha por Kirkpatrick y Gelatt en el año 1983 (Kirkpatrick, Gelatt, & Vecchi, 1983.), para encontrar soluciones al problema del vendedor viajero con un número relativamente grande de ciudades este método está inspirado en el proceso de recocido del acero y cerámicas, una técnica que consiste en calentar y después enfriar lentamente el material para variar sus propiedades físicas.

Búsqueda en Vecindad Variable o Variable Neighbourhood Search (VNS) Es una metaheurística relativamente nueva implementada por Mladenovic y Hansen en el año 1997 (Mladenovic & Hansen, 1997.), que evita quedarse en óptimos locales cambiando la estructura de la vecindad donde se realiza la búsqueda para generar mejores soluciones, la búsqueda se realiza de manera aleatoria o eligiendo un conjunto de vecinos para obtener diversos óptimos locales, este método además explota el hecho de que utilizar varios vecinos en la búsqueda local puede generar diferentes óptimos locales y de que el óptimo global es el óptimo local de un determinado grupo de vecinos.

Optimización por enjambre de Partículas o Particle Swarm Optimization (PSO) se atribuyen originalmente a los investigadores Kennedy, Eberhart y Shi es una técnica evolutiva inspirada en el comportamiento social de organismos como los enjambres de abejas, bancos de peces o bandadas de aves (Kennedy & Eberhart, 1995.). Los PSO permiten optimizar problemas a partir de una población de soluciones candidatas llamadas "partículas" y moviendo estas en el espacio de búsqueda manejando la posición y la velocidad de las partículas, el movimiento de cada partícula se ve influido por su mejor posición local, esto se repite hasta hacer que la nube de partículas converja rápidamente hacia las mejores soluciones.

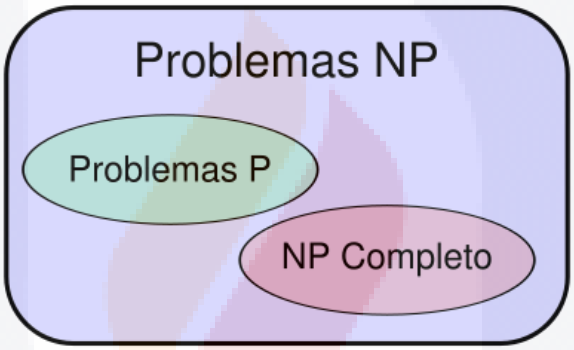
Algoritmo de Estimación de Distribución de Probabilidades o Estimation of Distribution Algorithm (EDA) son una clase de algoritmos basados en la computación evolutiva implementados por Larrañaga y Lozano (Larrañaga & Lozano, 2002.), los EDA´s sustituyen las funciones de cruce y mutación tradicionales de los algoritmos evolutivos por la generación de individuos obtenidos por simulación de una distribución de probabilidad. La distribución es estimada a partir del proceso iterativo de competencia de los individuos seleccionados en la generación anterior.

Búsqueda Dispersa o Scatter Search (SS) la primera descripción del método fue publicada en 1977 por Fred Glover (Glover, Heuristics for Integer Programming Using Surrogate Constraints., 1977.), es un método evolutivo basado en estrategias para combinar reglas de decisión así como en la combinación de restricciones. Se basa en el principio de que la calidad de un conjunto de soluciones puede mejorar mediante la combinación de estas, en resumen si se tienen 2 soluciones, se puede obtener una nueva mediante su combinación de modo que esta sea mejor a las que la originaron.

Optimización con Colonia de hormigas o Ant Colony Optimizacion (ACO) es un método probabilístico de la familia de algoritmos de colonia de hormigas orientado a la solución de problemas, para encontrar las mejores rutas o caminos implementado por Marco Dorigo (Dorigo & Stützle, Ant Colony Optimization., 2004.). Este método se basa en el comportamiento de las hormigas reales en su búsqueda de alimento, durante esta búsqueda las hormigas depositan una sustancia llamada feromona en los caminos más utilizados, mientras que en los caminos menos utilizados se va evaporando este rastro. Por lo tanto cuando una hormiga encuentra un buen camino hacia el alimento las probabilidades de que otras hormigas de la colonia sigan este camino es mayor.

Debido a que el problema de optimización que se busca resolver está enfocado a encontrar los caminos más cortos para las rutas de una flota de vehículos así como minimizar el número de vehículos utilizados, el algoritmo de hormigas se eligió como la metaheurística base para implementar un método de solución al problema de ruteo de vehículos, el funcionamiento de este algoritmo permite fácilmente modelar y trabajar con problemas que buscan encontrar los recorridos más cortos entre diversos puntos, contiene funciones que van calculando, optimizando y evaluando diferentes recorridos para intentar encontrar el de menor distancia, más adelante se hará una descripción más completa sobre el algoritmo de colonia de hormigas.

3. Teoría de la Complejidad Computacional y Clases de problemas P, NP, NP-Duros y NP-Completos



3.1 Introducción a la Teoría de la Complejidad Computacional.

Las ciencias computacionales son aquellas que incluyen las bases teóricas tanto de la información como de la computación, y cómo estas 2 áreas tienen aplicación en todos los sistemas computacionales (Denning, 2005.). Existen diversas ramas dentro de las ciencias computacionales una de ellas es la teoría de la complejidad computacional, que estudia aquellos problemas que requieren de la implementación de algoritmos para realizar uno varios cómputos o cálculos para resolverse.

Por otra parte aun cuando los problemas son computables y solucionables, existen problemas que no son posibles de resolver en la práctica ya que requieren grandes cantidades de memoria o tiempos de ejecución demasiado largos, es aquí donde la teoría de

la complejidad computacional se enfoca en conocer las capacidades y limitaciones fundamentales de las computadoras como la memoria, almacenamiento, tiempos de comunicación, tiempos de ejecución entre otros recursos.

Para conocer las limitaciones de las computadoras, las teorías de la complejidad computacional buscan crear modelos matemáticos que formalicen el concepto de realizar un cómputo o cálculo mediante un algoritmo, y realizar una clasificación de los problemas separándolos en clases de complejidad. Esta teoría se puede aplicar en todas las áreas en donde se necesite resolver un problema mediante un equipo de cómputo, debido a que no solo se busca tener una forma de resolver el problema sino que se busca encontrar la manera más rápida y con el menor número de recursos posibles.

La teoría de la complejidad computacional es una rama de la teoría de la computación que está centrada en separar y clasificar problemas computacionales, de acuerdo, a su dificultad inherente y en sus relaciones con las distintas clases de complejidad.

Los problemas computacionales se catalogan como difíciles cuando su solución requiere de una gran cantidad de recursos independientemente del algoritmo que se esté utilizando. La teoría de la complejidad computacional hace uso de modelos matemáticos, para estudiar estos problemas y establecer de manera formal una cuantificación de la cantidad de recursos necesarios para resolverlos y saber cuan eficiente es un algoritmo al hacerlo.

La complejidad computacional intenta determinar qué problemas se pueden realizar en una computadora y cuáles no, teniendo una cierta cantidad limitada de recursos. Otras áreas relacionadas son el análisis de algoritmos y la teoría de la computabilidad, la primera se dedica a determinar la cantidad de recursos requeridos por un algoritmo en particular para resolver un problema, mientras que la segunda analiza todos los posibles algoritmos utilizados para resolver el mismo problema.

Si se analizara un algoritmo y se ve que tarda mucho tiempo en resolver un problema difícilmente seria de utilidad al igual que uno que ocupe varios gigabytes de memoria probablemente no sería utilizado, si un cálculo requiere más tiempo que otro decimos que es más complejo, se le llama complejidad de tiempo. Y si un cálculo requiere de mayor espacio de almacenamiento que otro decimos que es más complejo y se le conoce como complejidad de espacio.

Al momento de implementar un algoritmo se deben considerar siempre las complejidades de tiempo y de espacio. Evidentemente, es importante saber si un algoritmo producirá una respuesta en un microsegundo, un minuto, o mil millones de años. Del mismo modo, la memoria requerida debe estar disponible para resolver un problema, por lo que la complejidad de espacio debe ser tenida en cuenta. Consideraciones sobre la complejidad espacial están vinculadas con las estructuras de datos especiales utilizadas para implementar un algoritmo (Rosen, 2003.).

En el año 1965 el matemático Jack R. Edmonds definió como un "buen" algoritmo, aquel cuyo tiempo de ejecución para resolver un problema puede ser acotado por un polinomio, esto es que el tiempo de ejecución es menor que el valor calculado a partir del número de variables de entrada para una fórmula polinómica, se dice que dicho problema puede resolverse en un tiempo polinómico (Karp, Combinatorics, Complexity, and Randomness., 1986.). Si, por ejemplo, para la solución de 3 problemas que se tardan $2a^2+5b$, $4a^6+7b^4-2c^2$ y $7a^4-3b^{498}$ segundos se tiene que los problemas se resuelven en un tiempo polinómico pero si se resolvieran en $4a^y$, $1a^x+5b^y-3c^z$ o 2^x segundos significaría que los problemas no son resolubles en tiempo polinómico.

El análisis requerido para estimar el uso de recursos de un algoritmo es una cuestión teórica, y por tanto necesitan una base formal, es por esta razón que estudios realizados por científicos como Richard Karp sobre los problemas que podían resolverse en un tiempo polinómico o no, condujeron al surgimiento de conceptos importantes hoy en día en la

teoría de la complejidad computacional como son las clases de problemas N, NP y los NP-Completos estas clases sirven para medir la complejidad de un problema.

3.2 Clases de Complejidad de Algoritmos.

Diferentes algoritmos poseen diferentes funciones de complejidad de tiempo, y gracias a estas funciones se puede determinar cuáles de ellos son "suficientemente eficientes" y cuáles son "demasiado ineficientes". Algunas de las funciones de complejidad más utilizados para describir la complejidad de tiempo se muestran en la **Tabla 3.1**.

Tabla 3.1 Terminología Común para la Complejidad de Algoritmos.

Complejidad	Orden de Complejidad
$\Theta(1)$	Constante
$\Theta(\log n)$	Logaritmica
$\Theta(n)$	Lineal
$\Theta(n \log n)$	Linearítmicas
$\Theta(n^2)$	Cuadrático
$\Theta(n^3)$	Cúbico
$\Theta(n^b)$	Polinomial
$\Theta(b^n)$, donde b > 1	Exponencial
$\Theta(n!)$	Factorial

Por ejemplo, un algoritmo que encuentra el más grande de los primeros 100 términos de una lista de n elementos mediante la aplicación de un algoritmo para recorrer los primeros 100 términos, donde n es un número entero mayor o igual a 100, tiene *complejidad constante* ya que utiliza 99 comparaciones sin importar el valor de n. El algoritmo de búsqueda lineal tiene *complejidad lineal* (en el peor de los casos o los casos promedio) y el algoritmo de búsqueda binaria tiene *complejidad logarítmica* (en el peor de los casos). Muchos algoritmos importantes tienen $\Theta(n \log n)$ o *complejidad linearítmica* (en el peor de los casos) como es el caso del algoritmo de ordenamiento por mezcla (La palabra linearítmica es una combinación de las palabras lineal y logarítmica). Un algoritmo tiene *complejidad polinomial* si tiene complejidad $\Theta(n^b)$, donde b es un número entero mayor o igual a 1. Por ejemplo, el algoritmo de ordenamiento de burbuja es un algoritmo de tiempo

polinomial porque utiliza $\Theta(n^2)$ comparaciones (en el peor de los casos). Un algoritmo tiene *complejidad exponencial* si tiene complejidad $\Theta(b^n)$, donde b es mayor que 1. El algoritmo que determina si una proposición compuesta de *n* variables se satisface mediante la comprobación de todas las posibles asignaciones de las variables de verdad es un algoritmo con *complejidad exponencial*, ya que utiliza $\Theta(2^n)$ operaciones. Por último, un algoritmo tiene *complejidad factorial* si tiene $\Theta(n!)$ complejidad de tiempo. Por ejemplo un algoritmo que encuentra todos los recorridos que un agente viajero puede utilizar para visitar n ciudades tiene complejidad factorial (Rosen, 2003.) (Garey & Johnson, 1979.).

3.2.1 Tratabilidad Computacional.

Cuando un problema se puede resolver mediante un algoritmo con complejidad polinomial en el peor de sus casos se le conoce como tratable, ya que se espera que el algoritmo produzca la solución al problema para una entrada de tamaño razonable en un tiempo relativamente corto.

Sin embargo, si el polinomio tiene un grado de complejidad alto (por ejemplo grado 100) o si los coeficientes son extremadamente grandes, el algoritmo puede tomar un tiempo extremadamente largo para resolver el problema. Por consiguiente, un problema que puede resolverse utilizando un algoritmo con complejidad polinomial, no garantiza que el problema pueda ser resuelto en una cantidad razonable de tiempo, inclusive para las entradas de valores relativamente pequeñas. Afortunadamente, en la práctica, el grado y los coeficientes de los polinomios para tales estimaciones son a menudo pequeños.

La situación es mucho peor para los problemas que no pueden resolverse utilizando algoritmos de complejidad polinomial. Dichos problemas se conocen como intratables estos problemas son tan complicados que no existe un algoritmo que los pueda resolver en un tiempo polinomial. Por lo general, pero no siempre, se requiere de una cantidad extremadamente grande de tiempo para resolver estos problemas en sus peores casos incluyendo para los de entradas de valores pequeñas. En la práctica sin embargo, también

existen situaciones donde un algoritmo con un cierto grado de complejidad puede ser capaz de resolver un problema mucho más rápido para la mayoría de casos pero no así para el peor caso. Cuando se está dispuesto a permitir que algunos, tal vez un pequeño número de casos no se puedan resolver en un plazo razonable de tiempo, el tiempo de complejidad promedio es una mejor medida de la duración que necesita un algoritmo para resolver un problema (Rosen, 2003.).

Se piensa que muchos de los problemas importantes en la industria son intratables, pero pueden ser prácticamente resueltos con el uso de ideas o herramientas surgidas en la vida diaria. Otra forma en que se manejan los problemas intratables cuando se busca una aplicación práctica es que en lugar de buscar soluciones exactas al problema, se buscan soluciones aproximadas. Como puede verse con el uso de algoritmos veloces creados para buscar dichas soluciones aproximadas, con la garantía de que los resultados no difieren en mucho a los de una solución exacta.

Por otra parte se pueden ver aquellos problemas que existen incluso aun cuando se puede demostrar que no hay ningún algoritmo capaz de resolverlos. Tales problemas se conocen como irresolubles (en oposición a problemas solubles que pueden ser resueltos utilizando un algoritmo sin importar su complejidad). La primera prueba de que hay problemas sin solución fue proporcionada por el gran matemático Inglés y científico de la computación Alan Turing cuando mostró que el problema de la parada es irresoluble (Turing, 1936.).

3.3 Problemas P, NP y NP-Completos.

En la sección anterior se ha hablado sobre algoritmos y su grado de complejidad espacial y más importante para este trabajo complejidad de tiempo. Cuando es abordado un problema en concreto se podrá saber si este tiene solución o no, y si es posible aplicar uno o varios algoritmos para solucionarlo, se dice que el grado de complejidad de un problema es igual al del mejor algoritmo que se utilice para resolverlo. De esta manera se clasifican los problemas, y los estudios que se hacen sobre los algoritmos que se aplican a problemas reales.

Estos estudios sobre los algoritmos confirman que existen problemas muy difíciles, problemas que desafían la utilización de equipos de cómputo para resolverlos. Para clasificar los problemas mediante la teoría de la complejidad computacional, se han establecido las clases de complejidad que son conjuntos formados por problemas de decisión que cuentan con una complejidad relacionada. Entre estas clases de complejidad están la P, NP, NP-Duros y los NP-Completos se puede ver un diagrama en la **Figura 3.1**, cada una de estas clases se explicarán a continuación.

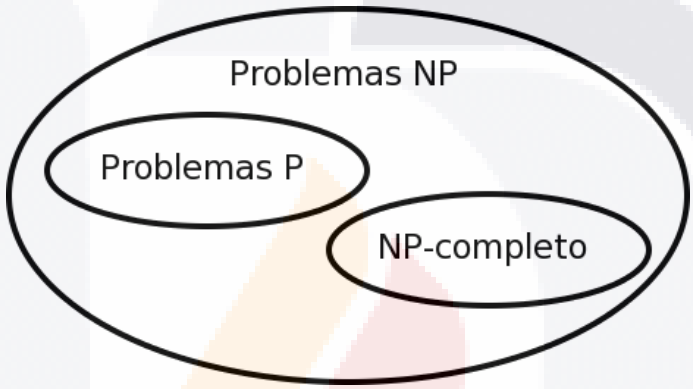


Figura 3.1 Diagrama de Clases de Complejidad. Clases de Problemas P, NP y NP-Completo.

3.3.1 Concepto de Problemas P.

Los algoritmos de complejidad polinomial se dicen que son tratables en el sentido de que pueden solucionarse en la práctica. Los problemas para los que se conocen algoritmos con esta complejidad se dice que forman la clase P (Tiempo Polinomial). Aquellos problemas para los que la mejor solución que se conoce es de complejidad superior a la polinomial, se dice que son problemas intratables. Sería muy interesante encontrar alguna solución polinomial que permitiera solucionarlos (Mañas, 1997).

3.3.2 Concepto de Problemas NP.

Algunos de estos problemas intratables pueden caracterizarse por el hecho de que puede aplicarse un algoritmo polinomial para comprobar si una posible solución es válida o no. Esta característica lleva a un método de resolución no determinista que consiste en aplicar métodos heurísticos para obtener soluciones hipotéticas que se van desestimando o aceptando a un ritmo polinomial. Los problemas de esta clase se denominan NP (Tiempo Polinomial No Determinista) (Mañas, 1997).

3.3.3 Concepto de Problemas NP-Duro.

Un problema es NP-duro si y sólo si es al menos tan duro como un problema NP-completo. Los problemas NP-duros no tienen algoritmos polinomiales por lo que un algoritmo que los resuelva en forma exacta puede tardar un tiempo prohibitivo, por esta razón se debería de conformar con el uso de algoritmos polinomiales que solo produzcan soluciones aproximadas, existen 2 categorías de estos algoritmos: los algoritmos de aproximación y los algoritmos heurísticos.

3.3.4 Concepto de Problemas NP-Completo.

Se conoce una amplia variedad de problemas de la clase NP, de entre los cuales destacan algunos de ellos con extrema complejidad. Como se mostró gráficamente en la **Figura 3.1** se puede decir que algunos problemas se hayan en la "frontera externa" de la clase NP. Son problemas dentro de la clase NP, y son los peores problemas posibles de la clase NP, a esta clase de problemas se le llama NP-Completos. Estos problemas se caracterizan por ser todos "iguales" en el sentido de que si se descubriera una solución en la clase P para alguno de ellos, esta solución sería fácilmente aplicable a todos ellos. Si se descubriera una solución para los problemas NP-completos, esta sería aplicable a todos los problemas NP y, por tanto, la clase NP desaparecería (Mañas, 1997).

Un ejemplo común de problema NP-Completo es el problema del agente viajero, este problema se puede analizar en 2 versiones distintas:

La versión de Optimización: Dado un grafo completo G ponderado con pesos enteros positivos, encontrar un ciclo Hamiltoniano en G con el peso mínimo.

La versión de Decisión: Dado un grafo completo G ponderado con pesos enteros positivos y un peso máximo m decidir si G tiene un ciclo Hamiltoniano con un peso total a lo sumo de m.

El Problema del Agente Viajero como Problema NP: Dado un grafo G, un límite superior de m, y una posible solución en la forma de un camino Hamiltoniano, es posible verificar o rechazar la solución con n adiciones y una sola comparación numérica. Esto se puede lograr en tiempo polinomial, debido a que una posible solución puede ser verificada o rechazada en tiempo polinomial, esta versión del Problema del Agente Viajero es NP.

El Problema del Agente Viajero como Problema NP-Duro: El problema del ciclo Hamiltoniano dirigido es un problema NP-Duro, si se logra reducir este problema al Problema del Agente Viajero se puede demostrar que también es NP-Duro.

La reducción es bastante sencilla. Dado un grafo G dirigido con n vértices se construye un nuevo grafo completo ponderado G' los vértices donde el peso de la arista (v, u) tienen peso igual a 1, si la arista (v, u) se encuentra en el grafo original G, de otra manera la arista tendrá un peso de n+1. G tiene un ciclo Hamiltoniano si existe un ciclo Hamiltoniano en G' con un peso total de n.

El problema del ciclo Hamiltoniano dirigido se reduce al Problema del Agente viajero, por lo tanto, es un problema NP-duro.Por lo tanto, si el Problema del Agente Viajero es un problema que existe en el conjunto NP y en el conjunto NP-Duro, se trata de un problema NP-Completo (Wilf, 2002.).

3.4 Ejemplos de Problemas NP-Completos.

Personas como Stephen Cook con su teorema sobre complejidad computacional (Cook, 1971.), y posteriormente Richard Karp basándose en el trabajo de Cook trabajó en la reducción y demostraciones de problemas combinatorios (Karp, Reducibility Among Combinatorial Problems., 1972), como resultado de estas investigaciones se obtuvo una lista de 21 problemas computacionales los cuales cumplen con la característica de pertenecer a la clase de problemas NP-Completos y se muestran en la **Figura 3.2**.

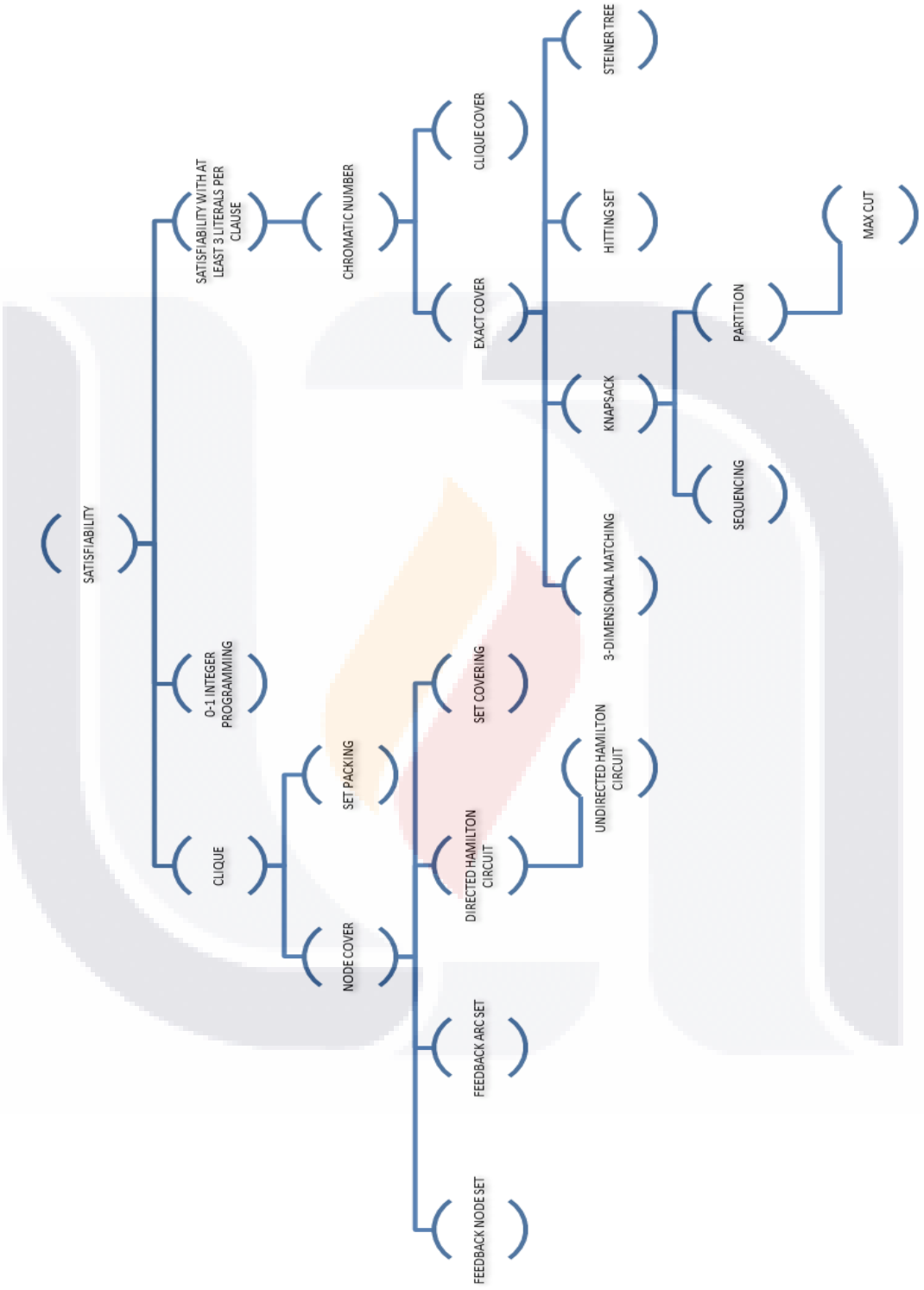
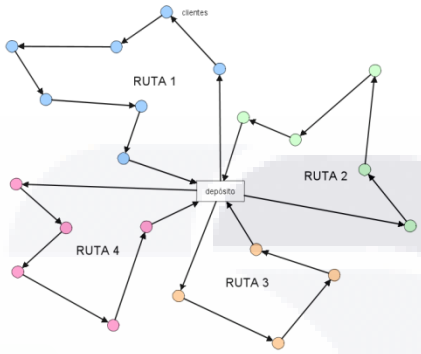


Figura 3.2 Problemas Computacionales de la Clase NP-Completo.

4. Problema de Ruteo de Vehículos



4.1 Antecedentes y Definición del Problema de Ruteo de Vehículos.

El Problema de Ruteo de Vehículos (Vehicle Routing Problem - VRP) se remonta a los años 50´s del siglo pasado, cuando George Dantzig y John Ramser realizaron la formulación de la programación matemática y el primer enfoque algorítmico para resolver el problema de la entrega de gasolina para estaciones de servicio (Dantzig & Ramser, 1959). Algunos años después fueron Clarke and Wright quienes propusieron una técnica heurística voraz para mejorar al método de Dantzig y Ramser (Clarke & Wright, 1964.). A partir de estos 2 trabajos iniciales, el interés en el problema de ruteo de vehículos ha ido en aumento y desarrollándose desde pequeños grupos de matemáticos a una amplia gama de investigadores y profesionales de diferentes disciplinas, que intervienen en este campo desarrollando en la actualidad aplicaciones que modelan y resuelven problemas reales.

La definición del VRP establece que *m* vehículos colocados inicialmente en un depósito tiene como tarea entregar cantidades discretas de mercancía a un número *n* de clientes con demandas conocidas y diferentes localizaciones geográficas, la determinación de la ruta óptima utilizada por un grupo de vehículos cuando hacen las entregas a los usuarios

representa un problema VRP clásico. Lo que se intenta alcanzar en un problema de ruteo de vehículos es:

- **Minimizar el costo total de la operación.**
- **Minimizar el tiempo total de transporte.**
- **Minimizar la distancia total recorrida.**
- **Minimizar el tiempo de espera.**
- **Maximizar el beneficio.**
- **Maximizar el servicio al cliente.**
- **Minimizar la utilización de vehículos.**
- **Equilibrar la utilización de los recursos, etc.**

La solución del problema VRP clásico es un conjunto de rutas que comienzan y terminan en el depósito, y que cumple la restricción de que cada cliente se visita sólo una vez. El costo de transporte se puede mejorar mediante la reducción de la distancia total recorrida y con la reducción del número de vehículos requeridos.

Respecto a su complejidad el problema de ruteo de vehículos se trata de un problema de clase NP-Duro ya que no existe un algoritmo capaz de resolverlo en tiempo polinomial, las principales razones que motivan su estudio son, los impactos de su aplicación en problemas reales y su alto grado de dificultad. Las instancias que se pueden resolver eficientemente por algoritmos exactos están conformadas por aproximadamente unos 50 clientes, mientras que instancias con un número mayor de clientes solo han sido resueltas óptimamente en casos particulares (Toth & Vigo, 2002.).

Utilizar un método heurístico no garantiza lograr una respuesta óptima, pero en la práctica si es posible lograr resultados aceptables cercanos al óptimo. En los últimos veinte años, la aplicación de técnicas metaheurísticas se ha convertido en la dirección más prometedora en la investigación de problemas de ruteo de vehículos (Caric & Gold, 2008.).

Se han obtenido buenos resultados mediante el uso de técnicas metaheurísticas en áreas como la investigación de operaciones, en técnicas de programación matemática, administración de productos y servicios de los sistemas de distribución. Un gran número de aplicaciones reales a nivel mundial ha mostrado que el uso de procedimientos computarizados en la planeación del proceso de distribución genera ahorros significativos en los costos de transportación total (del 5% al 20%). Es fácil ver el impacto que tendrían estos ahorros en la economía a nivel mundial. En realidad los procesos de transportación están involucrados en todas las etapas de producción y distribución de productos las cuales representan del 10% al 20% del costo final del producto (Toth & Vigo, 2002.).

Debido a la complejidad computacional del problema y viendo los beneficios económicos que es posible obtener en la fabricación y distribución de productos, en esta investigación se ha decidido utilizar una técnica metaheurística que es capaz de obtener una respuesta aceptable, cercana a una respuesta óptima y en un lapso de tiempo razonable, la metaheurística que se utilizará es conocida como Algoritmo de Colonia de Hormigas que tiene las características para ajustarse a los problemas de ruteo como el ordenamiento secuencial, problema del agente viajero, el ruteo de vehículos, así como en una gran variedad de aplicaciones cotidianas reales (Dorigo & Stützle, Ant Colony Optimization., 2004.) (Bullnheimer, Hartl, & Strauss, Applying the Ant System to the Vehicle Routing Problem., 1997.) (Hoos & Stützle, 2004.).

4.2 Características del Problema de Ruteo de Vehículos.

El problema de ruteo de vehículos consiste en determinar un conjunto de rutas de costo mínimo para una flota de vehículos, los cuales deben visitar un conjunto de clientes dispersos geográficamente siempre comenzando y terminando desde un punto inicial llamado depósito. Diferentes características de los clientes, vehículos y depósitos al igual que distintas restricciones con las que deben cumplir las rutas dan lugar a diferentes variantes del mismo problema.

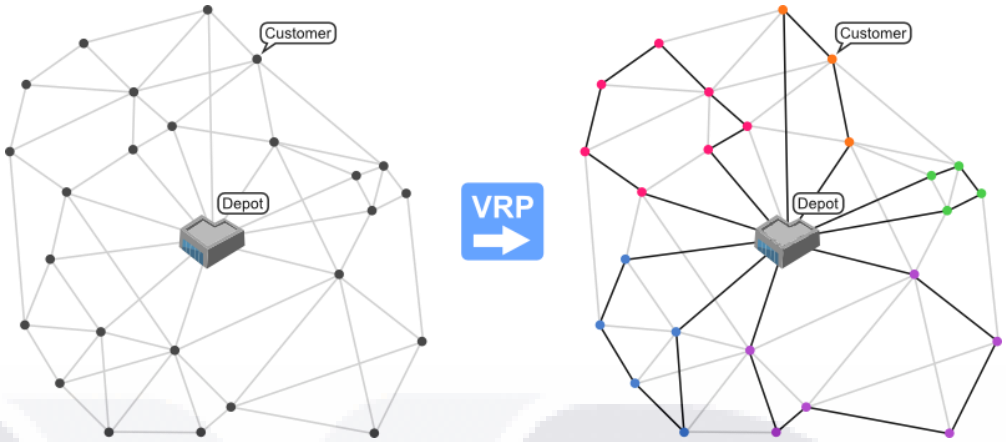


Figura 4.1 Solución simple para un problema de ruteo de vehículos.

En la **Figura 4.1** se muestra un ejemplo de una solución para un problema de ruteo de vehículos formado por 25 clientes (Customers) dispersos geográficamente, 1 depósito (Depot) y se pueden identificar las 5 rutas de vehículos que visitaran a todos los clientes. A continuación se hace una explicación de los componentes de un problema de ruteo de vehículos clásico y algunas características.

4.2.1 Clientes (Customers).

Los clientes tienen una cierta cantidad de demanda que requiere ser satisfecha por un vehículo, en la mayoría de los problemas la demanda es un producto que ocupa espacio en los vehículos y es común que un solo vehículo no pueda cubrir la demanda de todos los clientes de una misma ruta. En otros problemas la demanda no es un producto sino un servicio en el que el cliente simplemente debe ser visitado por el vehículo. En otras variantes el cliente tiene la función de proveedor y se desea que sea transportado hacia otro sitio, aquí la capacidad de productos que puede transportar simultáneamente un vehículo es una restricción.

En la mayoría de los problemas el cliente debe ser visitado solamente una vez, pero en algunos casos si se permite que la demanda de un cliente pueda ser satisfecha por vehículos diferentes. Los clientes pueden tener restricciones como el horario en que pueden recibir la visita de un vehículo, estas restricciones se manejan en forma de intervalos de tiempo

(ventanas de tiempo). En problemas con diferentes tipos de vehículos pueden existir restricciones de compatibilidad (como el peso, las dimensiones, o el tipo de carga de un vehículo) entre éstos y los clientes. En estos casos cada cliente solo puede ser visitado por algunos tipos de vehículos.

4.2.2 Depósitos (Depots).

En los problemas de ruteo de vehículos se maneja un punto llamado depósito, es el lugar en donde están ubicados inicialmente todos los vehículos y productos a distribuir. Generalmente se maneja que cada ruta comience y termine en el mismo depósito, aunque algunos problemas se manejan con varios depósitos con características como su ubicación y su capacidad máxima de producción. También puede ocurrir que cada depósito tenga asignada una flota de vehículos específica.

Los depósitos al igual que los clientes también pueden manejar ventanas de tiempo, las cuales son el tiempo necesario para abastecer un vehículo antes de que comience su ruta, o el tiempo de limpieza cuando regresa de su ruta. Otros problemas evitan las congestiones considerando la limitación de espacio en los depósitos y evitar que todos los vehículos se encuentren en un mismo depósito a la vez.

4.2.3 Vehículos (Vehicles).

Los vehículos pueden tener diferentes dimensiones, por ejemplo por peso o por volumen. Cuando en un problema se manejan distintos productos, los vehículos pueden manejar compartimentos, de manera que la capacidad del vehículo depende del tipo de productos que lleve. Generalmente cada vehículo tiene un costo fijo al utilizarlo y un costo variable proporcional a la distancia de la ruta.

Cuando los atributos de los vehículos (capacidad, costo, etc.) son iguales para todas las flotas de vehículos se les llama flotas homogéneas. Y si los vehículos tienen diferentes atributos se le llama flotas heterogéneas. El tamaño de la flota de vehículos es un dato de

entrada o una variable de decisión, pero el objetivo principal es encontrar la menor cantidad de vehículos que se utilizaran para realizar las entregas y en segundo lugar es minimizar las distancias de las rutas.Se asume que cada vehículo sólo recorre una sola ruta, pero también hay modelos en los que un vehículo puede realizar más de una ruta.

4.3 Variantes del Problema de Ruteo de Vehículos.

La mayoría de los problemas del mundo real son a menudo mucho más complejos que un problema VRP clásico. Pero en la práctica, el problema VRP clásico también tiene variantes surgidas de añadir diferentes restricciones aplicables al problema, Algunas de las principales son:

- Problema del agente viajero (The travelling salesman problem - TSP).
- Problema con múltiples agentes viajeros (The multiple travelling salesman problem – *m*-TSP).
- Problema de ruteo de vehículos con capacidad limitada (Capacitated vehicle routing problem - CVRP).
- Problema de ruteo de vehículos con ventanas de tiempo (Vehicle routing problem with time windows - VRPTW).
- Problema de ruteo de vehículos con varios depósitos (Multiple depot vehicle routing problem - MDVRP).
- Problema de ruteo de vehículos con recogidas y entregas (Vehicle routing problem with pick-up and delivering - VRPPD).
- Problema de ruteo de vehículos con entregas divididas por diferentes vehículos (Split delivery vehicle routing problem - SDVRP).
- Problema de ruteo de vehículos con valores aleatorios como el número de clientes, cantidades de demanda o tiempos de viaje (Stochastic vehicle routing problem - SVRP).
- Problema de ruteo de vehículos con entregas solo en determinados días (Periodic vehicle routing problem - PVRP).

- Problema de ruteo de vehículos con recogidas (Vehicle routing problem with backhauls - VRPB).

4.3.1 Problema del Agente Viajero (The Travelling Salesman Problem - TSP).

El problema del agente viajero (Travelling Salesman Problem - TSP) es uno de los problemas más estudiados en el campo de la optimización combinatoria computacional. A pesar de la aparente sencillez de su planteamiento, este problema es uno de los más complejos de resolver y existen demostraciones que equiparan la complejidad de su solución a la de otros problemas aparentemente mucho más complejos.

El planteamiento del problema es el siguiente: un vendedor que quiere encontrar la ruta más corta posible, saliendo desde un punto inicial y terminando en el mismo punto, visitando a todos sus clientes sólo una vez, es decir sin pasar dos veces por el mismo punto.

Desde el punto de vista práctico, el problema no está resuelto y desde el punto de vista teórico, las técnicas empleadas son sólo aproximaciones a una solución óptima. No suponen una resolución real del TSP y sólo ofrecen soluciones aproximadas lo suficientemente aceptables.

La solución más directa es mediante la búsqueda exhaustiva para evaluar todas las posibles combinaciones de recorridos y quedarse con aquella cuyo trazado utiliza la menor distancia. El problema reside en el número de posibles combinaciones que viene dado por el factorial del número de ciudades (N!) y esto hace que la solución por búsqueda exhaustiva sea impracticable para valores de N incluso moderados con los equipos de cómputo actuales. Por ejemplo, si una computadora fuera capaz de calcular cada combinación en un microsegundo (0.000001 segundos), tardaría aproximadamente 3.6288 segundos en resolver el problema para 10 ciudades, aproximadamente 39.9168 segundos en resolver el problema para 11 ciudades y 2432902008.17664 segundos que es igual a 77.14681659616438 años para resolver el problema de sólo 20 ciudades.

Los algoritmos clásicos no son capaces de resolver el problema general, debido a la explosión combinatoria de las posibles soluciones. Para ello, se ha buscado aplicar distintas técnicas computacionales: heurísticas evolutivas, redes de Hopefield, metaheurísticas etc.

4.3.2 Problema con Múltiples Agentes Viajeros (The Multiple Travelling Salesman Problem – *m*-TSP).

El problema de los múltiples agentes viajeros o m-TSP es una generalización del problema del agente viajero TSP en el que se permite una cantidad *m* de agentes. Dado un conjunto de *n* ciudades, un depósito donde se encuentran los agentes. El objetivo de dicho problema, es determinar el conjunto de rutas para los agentes viajeros a fin de minimizar el costo total de las rutas. Las métricas de costo de los recorridos se pueden representar por la distancia o el tiempo. Los requisitos que deben cumplir las rutas son que todas las rutas deben comenzar y terminar en el mismo depósito, y que cada ciudad debe ser visitada solamente una vez por un solo vendedor.

El problema *m*-TSP es una relajación del problema de ruteo de vehículos clásico, si la capacidad del vehículo es un valor suficientemente grande para no restringir la capacidad del vehículo, entonces el problema es el mismo que el *m*-TSP. Por lo tanto, todas las formulaciones y enfoques de solución para el VRP son válidos para el *m*-TSP. El problema *m*-TSP es una generalización del TSP, si el valor de *m* es igual a 1 entonces el problema *m*-TSP es igual al TSP y por lo tanto todas las formulaciones y métodos de solución para el *m*-TSP son válidas para el TSP.

Algunas variaciones del *m*-TSP que han surgido con la aplicación de restricciones algunas son, el *m*-TSP con manejo de múltiples depósitos, el *m*-TSP con la especificación del número de agentes viajeros, el *m*-TSP con cargos fijos, o el *m*-TSP con manejo de ventanas de tiempo.

4.3.3 Problema de Ruteo de Vehículos con Capacidad Limitada (Capacitated Vehicle Routing Problem - CVRP).

Este problema es una extensión del problema del agente viajero (TSP), en que las rutas permitidas son limitadas por la necesidad de que los objetos deben ser entregados desde un punto fuente hasta su destino por un vehículo de capacidad finita.

En este tipo de problema, se cuenta con un punto central llamado depósito, un número de *m* vehículos con capacidad limitada para cada uno. Éstos deben salir y regresar al depósito después de cumplir una secuencia de visita a clientes que se debe definir. Las restricciones que se deben tener en cuenta son las de iniciar y terminar el recorrido en el depósito y visitar a todos los clientes solamente una vez, satisfacer la demanda total y no sobrepasar la capacidad de carga máxima de cada vehículo.

El objetivo es minimizar la flota de vehículos y la suma del tiempo de viaje, y a la vez la demanda total para cada ruta no puede exceder la capacidad del vehículo que realiza esa ruta.

Una solución es factible si la cantidad total asignada a cada ruta no excede la capacidad del vehículo que realizará la ruta.

Algunos de los trabajos relacionados son por ejemplo, The Capacitated Vehicle Routing Problem (CVRP) (Chalasani & Motwani, 1999.). En este problema se tiene un conjunto de puntos en un espacio métrico, un grupo de vehículos de cierta capacidad y una colección de rutas de vehículos empezando en un origen, los cuales cada uno deben visitar un punto determinado. Otro trabajo es The Precedence- Constrained TSP (Bianco, Mingozzi, & Ricciardell, 1994.), que implica la existencia de un número finito de puntos que se deben visitar antes de visitar un punto definido.

4.3.4 Problema de Ruteo de Vehículos con Ventanas de Tiempo (Vehicle Routing Problem with Time Windows - VRPTW).

Es el mismo problema que el VRP clásico, pero con la restricción adicional que maneja ventanas de tiempo en las que los clientes deben ser atendidos. Las ventanas de tiempo implican la existencia de un límite o intervalo de tiempo dentro del cual un cliente debe ser abastecido. El objetivo es minimizar la flota de vehículos, el tiempo total de viaje y el tiempo de espera necesario para abastecer a todos los clientes en sus respectivos horarios.

Las soluciones son factibles si, además de las características de las soluciones de VRP, se agregan las siguientes características: una solución se convierte en no factible si un cliente es abastecido después del límite superior de su ventana horaria, si el vehículo llega antes del límite inferior de la ventana horaria causa un aumento en el tiempo de espera, cada ruta debe empezar y terminar dentro de la ventana de tiempo asociada al depósito y en el caso de ventanas menos estrictas, un servicio tardío no afecta a la factibilidad de la solución pero si se penaliza agregando un valor a la función objetivo.

Un trabajo en el que se utiliza el algoritmo de optimización de colonia de hormigas es propuesto como solución a este problema en (Doerner, Hartl, & Reimann, A hybrid ACO algorithm for the Full Truckload Transportation Problem., 2011.), el cual es basado en el sistema de hormigas, que fue inspirado sobre el comportamiento real de las hormigas. El objetivo es minimizar el costo total, el cual consiste de costos en la cantidad de vehículos utilizados y la distancia total viajada. Para ello usan una función objetivo de pesos para minimizar el tamaño del conjunto de vehículos necesarios. Finalmente se presentan unos resultados de la simulación para los cuales usaron un algoritmo heurístico y una implementación secuencial del ACO para ver el desempeño del ACO hibrido frente a los otros.

4.3.5 Problema de Ruteo de Vehículos con Varios Depósitos (Multiple Depot Vehicle Routing Problem - MDVRP).

Una compañía puede tener varios depósitos desde donde abastece a sus clientes. Si los clientes se encuentran agrupados alrededor de los depósitos, entonces el problema de distribución se puede modelar como varios VRP independientes. No obstante, si los clientes y los depósitos se localizan entremezclados se tiene un problema MDVRP. En este tipo de problema se consideran varios depósitos, donde en cada uno de ellos existe una flota de vehículos. A cada depósito se le asigna un número de clientes, los cuales son atendidos por los vehículos asignados al depósito. El objetivo de este problema, junto al ya mencionado de reducir la distancia recorrida, es minimizar la flota de vehículos asignados a cada depósito.

En (Tansini, Urquhart, & Viera, 2001.), se enfoca principalmente en la asignación de los clientes a los almacenes, y compara los resultados obtenidos por seis heurísticas con asignación (asignación a través de urgencias, asignación paralela, asignación simplificada y barrido de asignación) obtenidos para resolver un problema de transporte para los mismos casos.

4.3.6 Problema de Ruteo de Vehículos con Recogida y Entrega (Vehicle Routing Problem Pick-up and Delivering - VRPPD).

Esta variante del VRP incluye la posibilidad de recoger y entregar mercancía en lugar de sólo entregarla. También se contempla la devolución de productos por parte de los clientes. Por lo que hay que considerar que el vehículo tenga espacio para los productos que entregan. Esta restricción de devolución dificulta la planificación, lo que se traduce en un aprovechamiento ineficiente de la capacidad del vehículo y un aumento en las distancias de viaje. El fin es encontrar rutas óptimas de visita a los lugares de entrega y recibo para una flota de vehículos.

Por consiguiente es usual que se consideren situaciones restringidas, en donde todas las demandas de entrega empiezan desde el depósito y todos los productos recogidos serán llevados de vuelta al depósito, logrando de esta manera que no se produzcan intercambios de productos entre clientes. Otra alternativa es relajar la restricción de que los clientes deben ser visitados exactamente una vez. Otra simplificación usual es considerar que cada vehículo debe entregar todo su contenido antes de recoger otros productos de los clientes.

4.3.7 Problema de Ruteo de Vehículos con Entregas Divididas por diferentes vehículos (Split Delivery Vehicle Routing Problem - SDVRP).

SDVRP es una relajación del VRP, en donde se permite que un mismo cliente sea abastecido por distintos vehículos, siempre y cuando esto ayude a reducir los costos totales de la ruta. El problema consiste en que dada una flota de vehículos homogéneos estacionados en un depósito central y un conjunto de clientes requiriendo que sus demandas sean satisfechas, se deben encontrar las rutas de los vehículos empezando y terminando en el depósito cuando cada cliente sea visitado. Se diferencia de otros problemas más conocidos de rutas con capacidades en que se permite abastecer la demanda de cada cliente utilizando más de un vehículo.

Para solucionar este problema (Ho & Haugland, 2004.), proponen una búsqueda heurística tabú con cuatro diferentes estructuras de vecindad. Este problema también es abordado con el estudio del poliedro asociado al Problema de Rutas de Vehículos con Demanda Compartida (Martínez & Mota, 2000.), problema de distribución que surge cuando hay que repartir mercancías a un conjunto de clientes utilizando una flota fija de vehículos de capacidad limitada. El objetivo es diseñar las rutas de forma que se minimice la distancia total recorrida.

4.3.8 Problema de ruteo de Vehículos con Valores (como clientes, demandas o tiempos de viaje) Aleatorios (Stochastic Vehicle Routing Problem - SVRP).

En este problema se asume alguno(s) componente(s) no determinista(s) presentes en el sistema. Ya sea que se tengan clientes con una probabilidad de presencia o de ausencia, a clientes cuya demanda es una variable al azar y clientes donde el tiempo de servicio, junto con el tiempo de recorrido, son variables aleatorias.

En los SVRP existen dos etapas para llegar a una solución. Una primera solución es determinada antes de conocer las realizaciones de las variables aleatorias. En una segunda etapa, un recurso o acción correctiva puede tomarse cuando se conocen los valores de las variables aleatorias.

Debido a la aleatoriedad de algunos datos, no es posible requerir que todas las restricciones se satisfagan para todas las realizaciones de las variables aleatorias. Así que la decisión puede requerir, ya sea el cumplimiento de ciertas restricciones con una probabilidad dada, o la incorporación en el modelo de acciones correctivas a considerarse cuando una restricción es violada.

4.3.9 Problema de ruteo de Vehículos con entregas solo en Determinados días (Periodic Vehicle Routing Problem - PVRP).

En los VRP clásicos el periodo de planeación es de un día. En el caso de los PVRP, el clásico VRP es generalizado extendiendo el periodo de planeación a *M* días.Para esta variante se debe considerar las restricciones de que un cliente debe ser visitado como mínimo una vez dentro de un período de tiempo definido, cada vehículo debe tener una capacidad de carga definida, se tiene que satisfacer la demanda de cada cliente y respetar algunos horarios de recepción de mercadería y no es necesario que el vehículo tenga que regresar al depósito en el mismo día que salió, sino que debe regresar dentro del período de tiempo ya definido.

El objetivo es minimizar el costo de la suma de todas las rutas. Cada cliente tiene una demanda diaria conocida que debe ser completamente satisfecha en sólo una visita y con exactamente un vehículo. Si el periodo de planeación es de solo 1 día, entonces el problema se convierte en un VRP simple. Cada cliente debe ser visitado *k* veces, donde $1 \leq k \leq M$. En el modelo clásico de PVRP la demanda diaria de un cliente está siempre arreglada. El PVRP puede ser visto como un problema para generar un grupo de rutas para cada día, cuidando siempre que las restricciones involucradas sean satisfechas y los costos globales sean minimizados. También estos problemas pueden ser vistos como problemas de optimización combinatoria multinivel.

- En el primer nivel el objetivo es generar un grupo de combinaciones factibles para cada cliente.
- En el Segundo nivel se debe seleccionar una de las alternativas para cada cliente, de manera que las restricciones diarias sean satisfechas. De este modo se debe seleccionar los clientes a ser visitados en cada día.
- En el tercer nivel se resuelve el VRP para cada día.

4.3.10 Problema de Ruteo de Vehículos con Retorno de Bienes (Vehicle Routing Problem with Backhauls - VRPB).

El problema con retorno de bienes los clientes pueden demandar o retornar algunos productos. Es necesario tener en cuenta si el vehículo tiene la capacidad para contener los productos regresados por los clientes. Algo que se debe pensar también es que en primer lugar deben realizarse todas las entregas de cada ruta antes de iniciar la recolección, esto pensando en el hecho de que los vehículos son cargados por la parte trasera y las cargas son reacomodadas en los camiones. También las cantidades a ser distribuidas y recolectadas son fijas y conocidas con anticipación (Toth & Vigo, 2002.) (Dorronsoro, 2013).

En el problema tipo VRPB el conjunto de clientes se divide en dos. El primero contiene *n* centros de consumo con línea de recorrido (linehauls), que requieren que una cantidad dada de productos sea entregada y el segundo contiene *m* centros de consumo con recorrido de

vuelta o retornos (backhauls), donde se requiere que una cantidad de productos dada deba ser recogida.

En esta variante de VRP hay una restricción importante entre linehaul y backhaul, cuando una ruta pueda servir a ambos tipos de consumidores todos los centros de consumo con linehauls deben ser atendidos antes que algún centro de consumo con backhauls, si los hay (Toth & Vigo, 2002.) (Dorronsoro, 2013).

Este VRPB consiste en encontrar una cantidad de recorridos simples con mínimo costo de tal manera que cada ruta o recorrido visita el centro de distribución, cada centro de consumo es visitado solamente por una ruta y la suma de las demandas de los centros de consumo con linehaul y backhaul visitados por un recorrido no excede, individualmente, la capacidad del vehículo (Toth & Vigo, 2002.) (Dorronsoro, 2013).

En los últimos cincuenta años, muchos problemas del mundo real han requerido de formulaciones más amplias que han dado lugar a otras variantes del VRP clásico en la **Tabla 4.1** (Rocha, González, & Orjuela, 2011.) Se muestran algunas de las variantes de problemas VRP conocidas.

Tabla 4.1 Variantes de Problemas de Ruteo.

SIGLAS	SIGNIFICADO
TSP	Travelling Salesman Problem (Problema del Agente Viajero).
M-TSP	Multiple Travelling Salesman Problem (Problema del Agente Viajero Múltiple).

M-TSPTW	Multiple Travelling Salesman Problem with Time Windows (Problema del Agente Viajero Múltiple con Ventanas de Tiempo).
M-PTSP	Multiple Probabilistic Travelling Salesman Problem (Problema del Agente Viajero Múltiple Probabilístico).
PTSP	Probabilistic Travelling Salesman Problem (Problema del Agente Viajero Probabilístico).
CVRP	Capacitated Vehicle Routing Problem (Problema de Ruteo de Vehículos Capacitado).
DVRP	Distance Vehicle Routing Problem (Problema de Ruteo de Vehículos con restricciones de Distancia).
DCVRP	Distance and Capacitated Vehicle Routing Problem (Problema de Ruteo de Vehículos con restricciones de Capacidad y Distancia).
VRPTW	Vehicle Routing Problem Time Windows (Problema de Ruteo de Vehículos con Ventanas de Tiempo).
VRPMTW	Vehicle Routing Problem with Multiple Time Windows (Problema de Ruteo de Vehículos con Ventanas de Tiempo Múltiples).

VRPTD	Vehicle Routing Problem with Time Deadlines (Problema de Ruteo de Vehículos con Ventanas de Tiempo Duras).
VRPSTW	Vehicle Routing Problem with Soft Time Windows (Problema de Ruteo de Vehículos con Ventanas de tiempo Flexibles).
VRPB	Vehicle Routing Problem with Backhauls (Problema de Ruteo de Vehículos con Retornos).
VRPBTW	Vehicle Routing Problem with Backhauls and Time Windows (Problema de Ruteo de Vehículos con Retornos y Ventanas de Tiempo).
SDVRP	Split Delivery Vehicle Routing Problem (Problema de Ruteo de Vehículos con Entregas Divididas).
SDVRPTW	Split Delivery Vehicle Routing Problem with Time Windows (Problema de Ruteo de Vehículos con Entregas Divididas y Ventanas de Tiempo).
VRPHF	Vehicle Routing Problem Heterogeneous Fleet (Problema de Ruteo de Vehículos Flota Heterogénea).
VRPPD	Vehicle Routing Problem Pickup and Delivery (Problema de Ruteo de Vehículos con Recogida y Entrega).

VRPPDTW	Vehicle Routing Problem Pickup and Delivery and Time Windows (Problema de Ruteo de Vehículos con Recogida y Entrega con Ventanas de Tiempo).
SITE-DEPENDENT VRP	Vehicle Routing Problem Site-dependent (Problema de Ruteo de Vehículos Dependiente del Sitio).
HVRPFD	Heterogeneous Vehicle Routing Problem with Vehicle Dependent Routing Fixed Cost (Problema de Ruteo de Vehículos Heterogéneo con Costo Fijo y Vehículos Dependientes de Ruta).
HVRPD	Heterogeneous Vehicle Routing Problem with Vehicle Dependent Routing Cost (Problema de Ruteo de Vehículos Heterogéneo con Costo y Vehículos Dependientes de Ruta).
FSVRP	Vehicle Routing Problem with Fleet Size (Problema de Ruteo de Vehículos con Tamaño de Flota).
FSMFD	Fleet Size and Mix Vehicle Routing Problem with Fixed Costs and Vehicle Dependent Routing (Problema de Ruteo de Vehículos Mixto y Tamaño de Flota con Costo Fijo y Vehículos Dependientes de Ruta).

FSMD	Fleet Size and Mix Vehicle Routing Problem with Costs and Vehicle Dependent Routing (Problema de Ruteo de Vehículos Mixto y Tamaño de Flota con Costo y Vehículos Dependientes de Ruta).
FSMF	Fleet Size and Mix Vehicle Routing Problem with Fixed Costs (Problema de Ruteo de Vehículos Mixto y Tamaño de Flota con Costo Fijo).
PVRP	Vehicle Routing Problem with Periodic (Problema de Ruteo de Vehículos con Entregas Periódicas).
MULTI-TRIP VRP	Vehicle Routing Problem Multiple Trips (Problema de Ruteo de Vehículos Múltiples Viajes).
MULTI-DEPOT VRP	Vehicle Routing Problem Multiple Depots (Problema de Ruteo de Vehículos Múltiples Depósitos).
MCVRP	Multi Capacity Vehicle Routing Problem (Problema de Ruteo de Vehículos Múltiples Capacidades).
MOVRP	Multi Objective Vehicle Routing Problem (Problema de Ruteo de Vehículos Múltiples Objetivos).
SVRP	Stochastic Vehicle Routing Problem (Problema de Ruteo de Vehículos Estocástico).

VRPUD	Vehicle Routing Problem Uncertain Demand (Problema de Ruteo de Vehículos Demanda Incierta).
VRPSTT	Vehicle Routing Problem with Stochastic Travel Times (Problema de Ruteo de Vehículos).
SVRP	Vehicle Routing Problem Stochastic Nodes (Problema de Ruteo de Vehículos Estocástico con Nodos Estocásticos).

4.4 Formulación Matemática del Problema de Ruteo de Vehículos con Capacidad Limitada.

El problema de ruteo de vehículos con capacidad limitada consiste en la entrega de un producto o mercancía a un conjunto de clientes dispersos mediante una flota de vehículos. Cada uno de los clientes tiene asociado una demanda de mercancías definida. Los vehículos también tienen una capacidad de transporte limitada y homogénea. El objetivo del problema consiste en construir un conjunto de rutas que comiencen y finalicen en el depósito tratando de minimizar los costos (generalmente medido en distancias o tiempo) de transportación de los productos o mercancías. Las restricciones a las que está sujeto el CVRP son las siguientes:

- La demanda de mercancía de cada uno de los clientes debe ser satisfecha por completo.
- Un cliente solo puede ser atendido o asignado a un solo vehículo o ruta.
- El número de vehículos utilizados en la solución no tiene límite. (sin embargo algunas instancias y autores manejan un número máximo de vehículos permitidos).

A continuación se describe los parámetros utilizados para la formulación matemática del CVRP.

$i =$ Nodo o cliente de partida (1, 2,….., n).

$j =$ Nodo o cliente de llegada (1, 2,….., n).

$k =$ Número de vehículos disponibles (1, 2,….., k).

$n =$ Cantidad de clientes.

$G = (V, A)$ Es un grafo completo no direccionado.

$V = (v_0, v_1, v_2, \ldots, v_n)$ Conjunto de nodos o clientes, v_0 generalmente representa al depósito.

$A = \{ (i, j) : i, j \in V, \quad i \neq j \}$ Es el conjunto de aristas o de caminos entre los nodos o clientes.

$Q =$ Capacidad de transporte de los vehículos.

$C_{ij} =$ Costo de viajar del cliente *i* al cliente *j*.

$C = (c_{ij})$ Matriz de costos de los clientes.

$d_i =$ Demanda de mercancías del cliente *i*.

$$x_{ij} = \begin{cases} 1. & \text{si la solución utiliza el arco (i, j)} \\ 0. & \text{en otro caso} \end{cases}$$

El modelo basado en aristas considera variables de tipo entero x_{ij} que indican si un vehículo se mueve del cliente *i* al cliente *j* en c_{ij} .

A continuación se presenta el problema CVRP.

En donde se desea Minimizar:

$$\sum_{i \in V} \sum_{j \in V} c_{ij} \, x_{ij}$$

Ecuación 4.1 Minimizar Distancia entre *i* y *j*

Sujeto a las siguientes restricciones:

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall \, j \in V \setminus \{0\}$$

Restricción 4.1

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V \setminus \{0\}$$

Restricción 4.2

$$\sum_{i \in V} x_{i0} = K$$

Restricción 4.3

$$\sum_{j \in V} x_{oj} = K$$

Restricción 4.4

$$x_{ij} \in \{0,1\} \qquad i, j \in V$$

Restricción 4.5

$$\sum_{i \notin S} \sum_{j \in S} xij \geq r(S), \quad \forall S \subset V \setminus \{0\}, \quad S \neq \phi$$

Restricción 4.6

De la **Restricción 4.1** a la **Restricción 4.4** corresponden a restricciones de ruteo, la **Restricción 4.5** es para el modelo de dos índices que considera las variables de decisión. La **Restricción 4.6** impide la existencia de sub-tours, controlando restricciones de capacidad y corte. Donde $r(S)$ es el número mínimo de vehículos necesarios para satisfacer la demanda en *S*.

4.5 Aplicaciones del Problema de Ruteo de Vehículos.

El problema de ruteo de vehículos tiene como objetivo encontrar un camino que recorra un conjunto de clientes, comenzando y terminando en el depósito, minimizando la suma de los tiempos de espera o distancias de los clientes. Este es un problema de optimización simple y natural, que puede ser encontrado en diversas situaciones de la vida real.

Dentro del área de redes de computadoras, el problema de ruteo de vehículos tiene aplicaciones en la búsqueda de información en una red determinada. Queremos encontrar información que se encuentra en algún punto de la red en forma equiparable mediante un agente que se mueve a través de ella. El objetivo es diseñar una forma de recorrer estos puntos que minimicen el tiempo de búsqueda.

Otra motivación surge en la diagramación del recorrido de algún servicio de reparaciones. Supongamos que un técnico tiene que reparar un conjunto de máquinas que se encuentran en distintos lugares, y que el tiempo de reparación es insignificante o el mismo para todas las maquinas. El objetivo es encontrar una ruta que minimice el tiempo total de espera de todas las maquinas.

Otras aplicaciones se pueden encontrar para el ruteo de vehículos guiados automáticamente de celdas en un sistema flexible de manufactura, operaciones de recolección de mercadería por camiones, distribución postal, transporte escolar, distribución de productos perecederos, etc. El problema de ruteo también puede ser interpretado como un problema de programación de tareas (planificación) en una única máquina.

4.6 Métodos de Solución para el VRP.

En lo que respecta a métodos de solución para los problemas de ruteo de vehículos, se abordan 3 categorías principales basándose en los trabajos realizados por Gilbert Laporte (Laporte, 1992.) y por Linda Rocha, Elsa González y Javier Orjuela (Rocha, González, & Orjuela, 2011.), agrupadas de la siguiente manera: métodos exactos, heurísticas y metaheurísticas. En la **Figura 4.2** se muestra la estructura mencionada y a partir de ella se desprenden las demás clasificaciones que se explican a continuación.

Figura 4.2 Estructura de los Métodos de Solución para VRP.

4.6.1 Métodos Exactos.

Los métodos exactos son eficientes en problemas hasta los 50 clientes debido a restricciones de tiempo computacional. Los métodos exactos se pueden clasificar en 3 grupos: búsqueda directa de árbol, programación dinámica, programación lineal y entera. En la **Figura 4.3** se muestra la clasificación de estos métodos.

Figura 4.3 Estructura de los Métodos Exactos de Solución para VRP.

La **Figura 4.4** muestra la clasificación de los métodos de solución para VRP mediante la búsqueda directa de árbol.

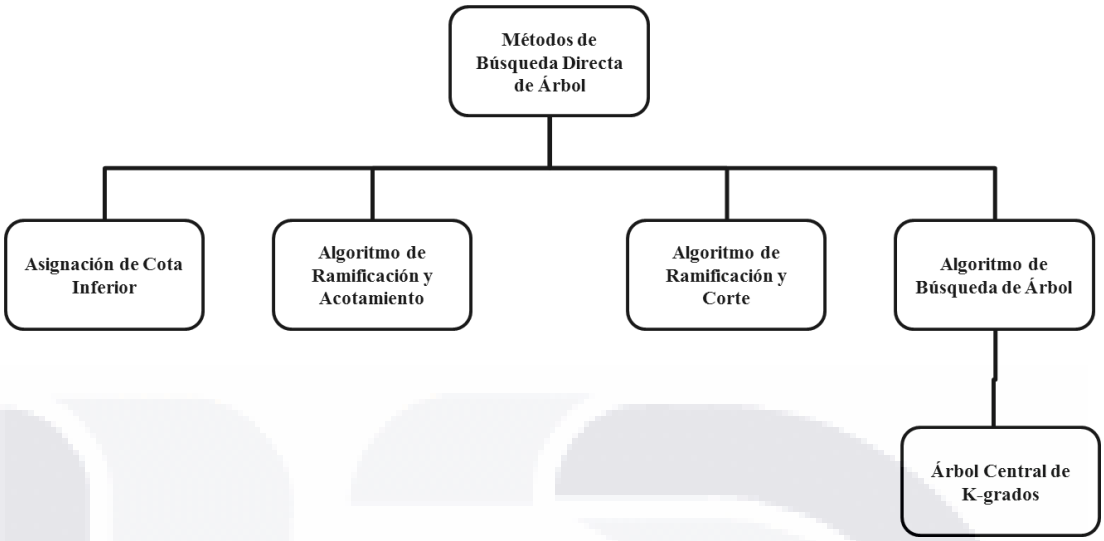


Figura 4.4 Estructura de los Métodos de Solución con Búsquedas de Árbol.

La **Figura 4.5** muestra la clasificación de los métodos de solución de programación lineal y entera.

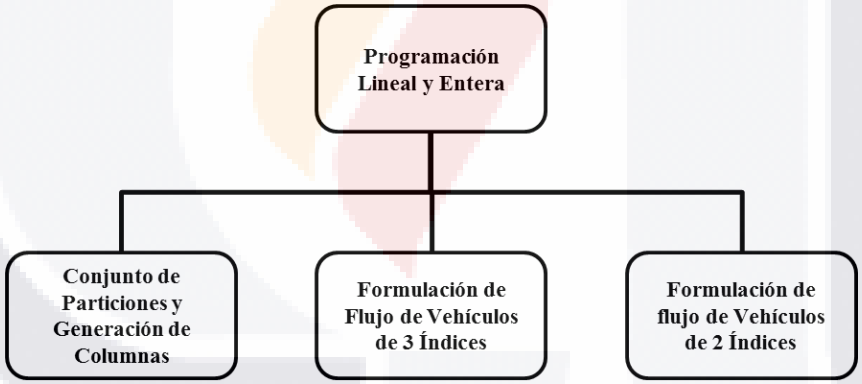


Figura 4.5 Estructura de los Métodos de Solución con Programación Lineal y Entera.

4.6.2 Heurísticas.

Las heurísticas son procedimientos que proporcionan soluciones de aceptable calidad mediante una exploración limitada del espacio de búsqueda. Clarke y Wright, propusieron el primer algoritmo que resultó efectivo para resolver el VRP en 1964. La mayoría de heurísticas clásicas para resolver el VRP fueron desarrolladas entre 1960 y 1990. Estos

métodos parten de rutas que contienen un único nodo para encontrar el mejor par (nodo, ruta) que representa la mejor intersección.

Los métodos heurísticos se pueden clasificar de diversas maneras, la más común es métodos constructivos, métodos de 2 fases y heurísticas de mejora como se muestra en la **Figura 4.6.**

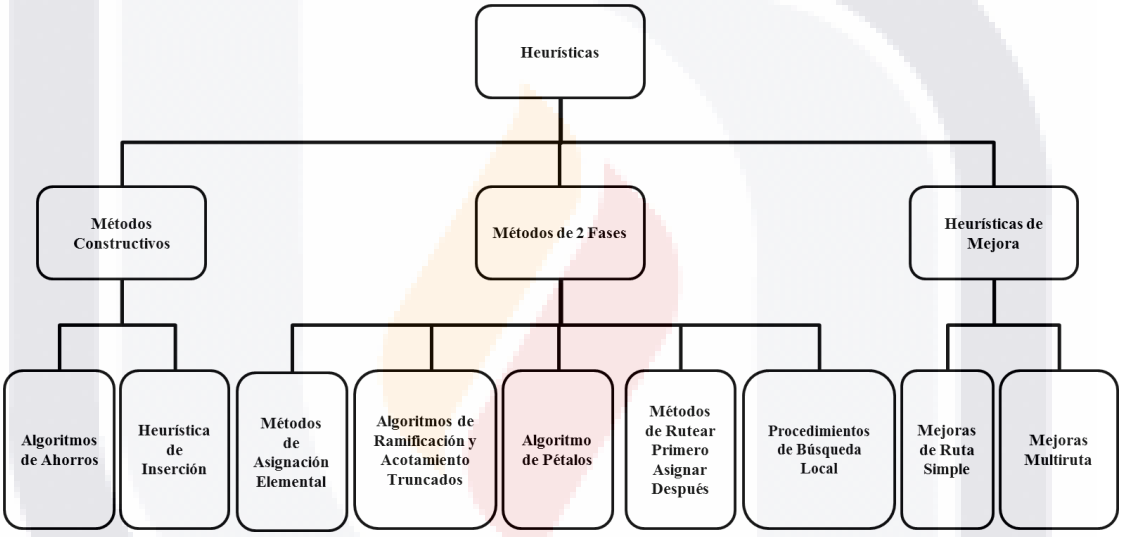


Figura 4.6 Estructura de los Métodos de Solución por Heurísticas.

La **Figura 4.7** muestra la clasificación establecida de los algoritmos de ahorros.

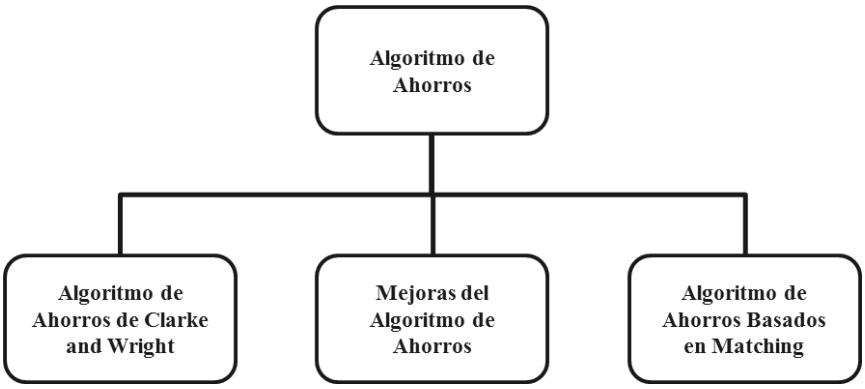


Figura 4.7 Estructura de los Algoritmos de Solución por Algoritmos de Ahorros.

La **Figura 4.8** muestra la clasificación para heurísticas de inserción.

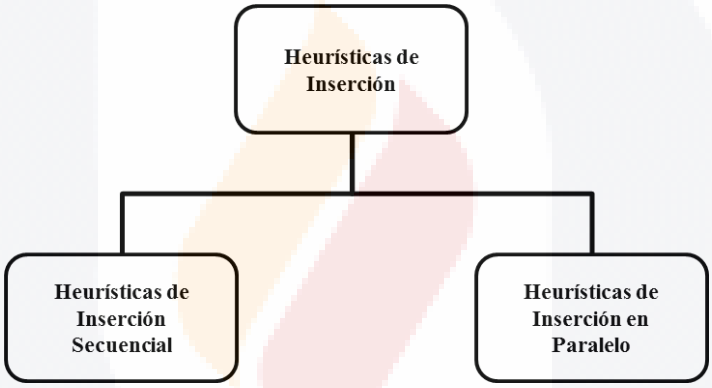


Figura 4.8 Estructura de los Algoritmos de Solución por Heurística de Inserción.

La **Figura 4.9** muestra la clasificación de métodos de solución de 2 fases.

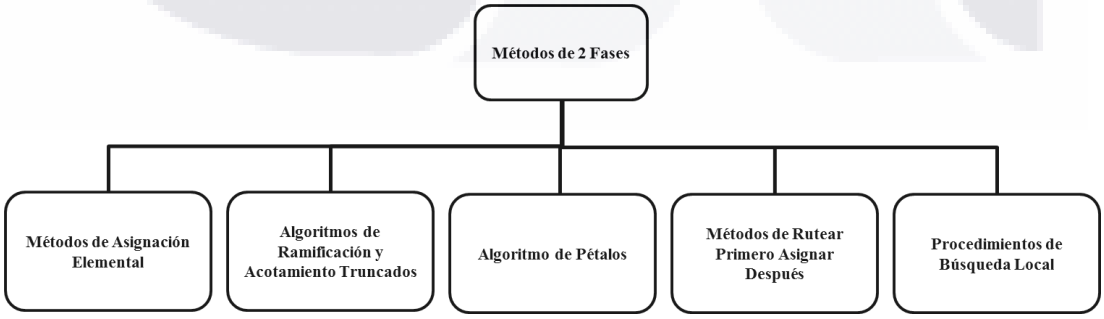


Figura 4.9 Estructura de los Algoritmos de Solución de 2 Fases.

4.6.3 Metaheurísticas.

Las metaheurísticas fueron desarrolladas en los años 70´s y se caracterizan porque realizan un procedimiento de búsqueda para encontrar soluciones de aceptable calidad, mediante la aplicación de operadores independientes del dominio que modifican soluciones intermedias guiadas por la idoneidad de su función objetivo. Dentro de esas se encuentran el recocido simulado, redes neuronales, búsqueda tabú, algoritmos genéticos, algoritmos de colonia de hormigas y búsquedas de vecindades, la **Figura 4.10** muestra la clasificación de los métodos de solución mediante metaheurísticas.

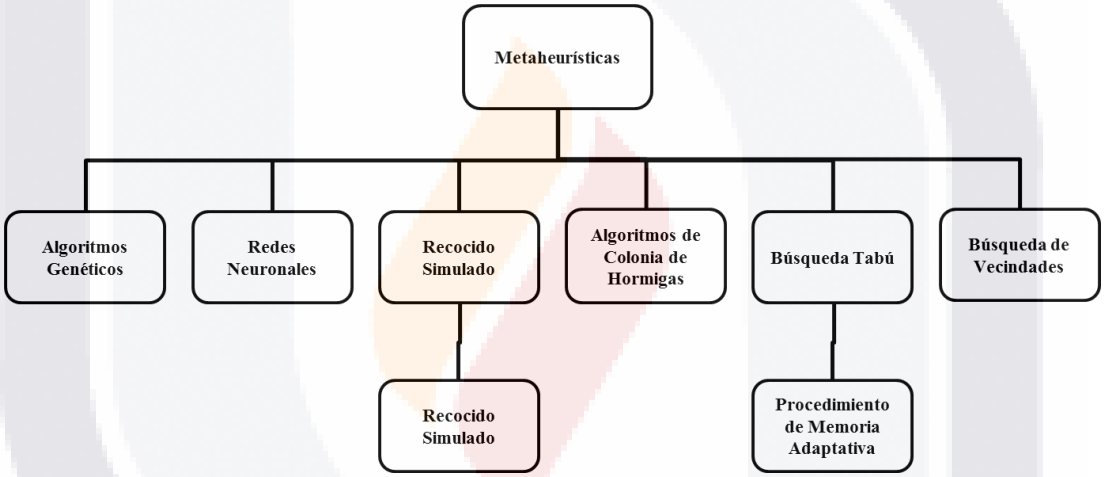


Figura 4.10 Estructura de los Algoritmos de Solución con Metaheurísticas.

4.7 Heurística Rutear Primero – Asignar Después.

Para resolver los problemas de ruteo de vehículos se han propuesto diferentes métodos tanto exactos como aproximados con el objetivo de encontrar la solución óptima, de los métodos publicados se decidió utilizar uno llamado Rutear Primero – Asignar Después por su sencillez de implementación y sus buenos resultados, a continuación se hace una breve descripción de este método el cual forma parte de la implementación para solucionar problemas CVRP´s.

El método que se presenta a continuación es el método Rutear primero - Asignar después propuesto por Beasley en el año 1983 con el objetivo de resolver problemas de ruteo de vehículos en su trabajo (Beasley, 1983.). Este método se llama así porque primero se crea una ruta para decidir el orden en que serán visitados los clientes y después se separan los clientes en grupos cuyas demandas puedan ser satisfechas por las rutas de vehículos.

Algunas características que hacen atractivo este método son: el uso de un tour para todos los clientes asegura que habrá más posibilidades de que los clientes más cercanos entre si pertenezcan a la misma ruta, mediante la técnica de separación podemos ser capaces de generar y considerar un gran número de rutas de vehículos factibles y de este grupo elegir la mejor, la partición del recorrido es rápida en términos computacionales y se podría mejorar empleando algoritmos de mejora como el de Dijkstra (Dijkstra, 1959) que maneja operaciones de complejidad O^2, en algunos casos el trabajar con un solo recorrido puede resultar en particiones de clientes que no puedan ser satisfechas por las rutas de vehículos para evitar estas situaciones, se pueden crear aleatoriamente varios recorridos diferentes para producir más rutas de vehículos que si se puedan llegar a satisfacer.

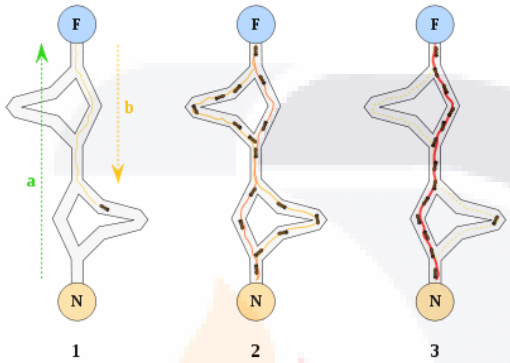
Este método original en el trabajo de Beasley consiste en los siguientes pasos:

1) Se genera un recorrido inicial al azar y se calcula una matriz de distancias llamada C_{ij}, al recorrido inicial se le aplica un procedimiento de intercambio 2-Opt hasta que ya no sea posible realizar mejoras al recorrido.
2) Mientras se realiza el procedimiento de intercambio 2-Opt en el recorrido, se sigue calculando la matriz de distancias C_{ij} en la cual se añaden valores positivos en cada elemento, con el objetivo de que las rutas producidas utilicen el menor número de vehículos posibles.
3) Se utilizó el algoritmo de Floyd´s (Floyd, 1962.), para calcular las rutas de menor costo en el grafo dirigido utilizando los valores de la matriz C_{ij} y obtener una partición optima del recorrido.

4) Cada una de las rutas de la partición óptima del recorrido se optimizan con el método de intercambio 3–Opt, siendo más sencillo aplicarlo con un pequeño número de clientes en la ruta, con esto se trata de asegurar que el recorrido del agente viajero a través de los clientes es casi óptima.

Como se trata de un problema de ruteo de vehículos con capacidad limitada es muy importante tener siempre en cuenta la capacidad de transporte del vehículo al momento de realizar las particiones del recorrido.

5. Algoritmos de Optimización Basados en Colonias de Hormigas



5.1 Antecedentes.

El significado de optimizar es buscar la mejor manera de realizar una actividad, o planificar una actividad para obtener los mejores resultados; sin embargo visto desde un contexto científico e informático, optimizar es determinar los valores de las variables que intervienen en un determinado sistema o problema, para que el resultado que se obtenga sea el mejor posible.

Los problemas de optimización pueden tener diferentes soluciones y manejan criterios para diferenciar entre ellas. Se puede definir de una manera más precisa los problemas de optimización como el procedimiento de buscar los valores de las variables con los que una función objetivo alcance sus valores máximos (o en ocasiones los mínimos). En ocasiones las variables se encuentran sujetas a restricciones dependiendo del problema.

Los problemas de optimización que pertenecen al conjunto de problemas NP-Duros es decir que poseen un alto grado de complejidad computacional, son aquellos en los que no se puede garantizar encontrar una solución óptima en un tiempo razonable. La existencia de

grandes cantidades y variedades de problemas de este tipo que necesitan ser resueltos motivó el estudio y desarrollo de procedimientos eficientes para encontrar soluciones aceptables, aunque no sean las óptimas. Estos procedimientos en los que el tiempo es tan importante como la calidad de la solución obtenida, se conocen como métodos heurísticos.

Los métodos heurísticos son procedimientos para resolver problemas de optimización mediante aproximaciones, en donde la información del problema se utiliza de manera inteligente para obtener soluciones aceptables. En comparación a los métodos exactos que dan una solución óptima del problema empleando una enorme cantidad de tiempo, los métodos heurísticos buscan dar una solución aceptable utilizando una cantidad de tiempo menor a los métodos exactos, pero no significa que la solución obtenida sea la óptima. Otras razones para utilizar métodos heurísticos que se pueden destacar son:

- Debido a la naturaleza y complejidad de los problemas no se conoce ningún método exacto para su resolución.
- Aunque exista un método exacto para resolver el problema, su uso tiene un costo computacional muy alto.
- El método heurístico tiene más flexibilidad que un método exacto, permitiendo por ejemplo, la incorporación de condiciones de difícil modelización.
- El método heurístico se utiliza como parte de un procedimiento global que garantiza el óptimo de un problema. Existen 2 alternativas:
    1. El método heurístico proporciona una buena solución para iniciar.
    2. El método heurístico actúa en uno de los pasos intermedios del procedimiento.

Debido a que los métodos heurísticos son de naturaleza compleja y variada, además su diseño y construcción fueron pensados para resolver problemas particulares; Es por estas razones que hacer una clasificación y lograr su aplicación a problemas diferentes sea una tarea complicada. A continuación se muestra una clasificación de los métodos heurísticos más conocidos, manejando categorías amplias y no excluyentes (Martí, 2003):

Métodos de Descomposición: El problema original se descompone en sub-problemas más sencillos de resolver, teniendo en cuenta, aunque sea de manera general, que ambos pertenecen al mismo problema.

Métodos Inductivos: La idea de estos métodos es generalizar de versiones pequeñas o más sencillas al caso completo. Propiedades o técnicas identificadas en estos casos más fáciles de analizar pueden ser aplicadas al problema completo.

Métodos de Reducción: Consiste en identificar propiedades que se cumplen mayoritariamente por las buenas soluciones e introducirlas como restricciones del problema. El objeto es restringir el espacio de soluciones simplificando el problema. El riesgo obvio es dejar fuera las soluciones óptimas del problema original.

Métodos Constructivos: Consisten en construir literalmente paso a paso una solución del problema. Usualmente son métodos deterministas y suelen estar basados en la mejor elección en cada iteración. Estos métodos han sido muy utilizados en problemas clásicos como el problema del agente viajero.

Métodos de Búsqueda Local: A diferencia de los métodos anteriores, los procedimientos de búsqueda local comienzan con una solución del problema y la mejoran progresivamente. El procedimiento realiza en cada paso un movimiento de una solución a otra con mejor valor. El método finaliza cuando, para una solución, no existe ninguna solución accesible que la mejore.

Todos estos métodos han ayudado a solucionar problemas reales, pero principalmente los métodos constructivos y los métodos de búsqueda local han sido la base del funcionamiento de las técnicas metaheurísticas. Hoy en día se están realizando numerosas investigaciones y desarrollo de técnicas metaheurísticas, algunos de los más conocidos son (Glover & Kochenberger, Handbook of Metaheuristics. (Hardcover)., 2003.):

- **Algoritmos Genéticos (Genetic Algorithms).**
- **Algoritmos Meméticos (Memetic Algorithms).**
- **Búsqueda Dispersa y Re enlazado de Caminos (Scatter Search and Path Relinking).**
- **Búsqueda Tabú (Tabú Search).**
- **Búsqueda de Vecindad Variable (Variable Neighborhood Search).**
- **Búsqueda Local Guiada (Guided Local Search).**
- **Búsqueda Adaptativa Aleatoria y Voraz (Greedy Ramdomized Adaptive Search Procedures).**
- **Búsqueda Local (Local Search).**
- **Métodos Multiarranque (Multi-Start Methods).**
- **Optimización Basada en Colonias de Hormigas (Ant Colony Optimization Metaheuristic).**
- **Optimización Basada en Enjambres de Partículas (Particle Swarm Optimization Metaheuristic).**
- **Recocido Simulado (Simulated Annealing).**
- **Redes Neuronales Artificiales (Artificial Neural Networks).**

Uno de los algoritmos metaheurísticos seleccionados para encontrar la solución al problema de ruteo de vehículos con capacidad limitada es el algoritmo de optimización basado en colonia de hormigas, este algoritmo permite encontrar soluciones aceptables a problemas de optimización por medio de agentes llamados hormigas artificiales que basan su funcionamiento en el comportamiento de las colonias de hormigas reales.

5.2 Concepto de Algoritmo de Colonia de Hormigas (Ant Colony Optimization - ACO).

El método de optimización basado en colonia de hormigas, es un método inspirado en el comportamiento de las hormigas reales, este método resulta muy útil para la resolución de problemas de optimización combinatoria. Como su nombre lo indica, ACO es un algoritmo de optimización que utiliza hormigas artificiales, que imitan la capacidad de las hormigas

reales de construir caminos invisibles marcados por una sustancia química producida por las hormigas llamada feromona, estos caminos con rastros de feromona son utilizados por las hormigas para guiarse entre su nido y una fuente de alimentos (Deneubourg, Aron, & Goss, 1990.).

El proceso en que las hormigas reales marcan sus caminos con rastros de feromona para optimizar de manera colectiva su estrategia para buscar alimentos, es conocido como estigmergia y consiste en una forma indirecta de comunicación realizando modificaciones al ambiente. Este concepto se puede aplicar en distintos problemas, como la búsqueda de mejores caminos en el enrutamiento de redes, y más.

La idea básica de un ACO se puede visualizar a través de un grafo, en el que se consideraran los vértices del grafo como ubicaciones, y las aristas como las rutas o caminos de acceso entre las ubicaciones. Un grupo de hormigas se distribuyen a través de los vértices del grafo y posteriormente de manera probabilística elegirán el próximo vértice, basándose en la intensidad del rastro de feromonas de las posibles aristas a los que se puedan avanzar. Cuanto más alto sea el valor del rastro de feromonas en una arista mayor será la probabilidad de que la hormiga simulada tome ese camino. Después de que todas las hormigas artificiales completan su recorrido en el grafo, se mide la longitud o costo de su recorrido (utilizando alguna técnica representativa para el problema), y se realiza una actualización del rastro de feromona de acuerdo a la calidad del recorrido. Con el transcurso del tiempo y de los recorridos hechos por las hormigas el mejor camino se hace evidente y cada hormiga comenzará a tomar el camino tomado por otras. A partir de esta sencilla explicación, se puede ver cómo la estrategia empleada por las hormigas en su búsqueda de alimento con el proceso de estigmergia es exitosa.

Las aplicaciones de métodos de optimización basados en colonias de hormigas son interesantes ya que han mostrado su versatilidad en diferentes aplicaciones como ejemplo se tienen:

1. Para resolver problemas del tipo NP-Duros, que no pueden ser resueltos por algoritmos exactos tradicionales.
2. Para resolver problemas dinámicos que busquen la ruta más corta, en los que las propiedades del problema van cambiando al mismo tiempo que se busca la solución.
3. Para problemas en los que la arquitectura computacional está distribuida espacialmente.

5.2.1 Aplicación del Algoritmo Basado en Colonia de Hormigas en el Problema del Agente Viajero (Traveling Salesman Problem – TSP).

Se puede encontrar en el estado del arte como el algoritmo de colonia de hormigas es aplicado al problema del agente viajero que busca encontrar el camino de menor costo a través de los puntos en un grafo, visitando cada punto solamente una vez (Dorigo & Stützle, Ant Colony Optimization., 2004.). Como se muestra en el gráfico de la **Figura 5.1**.

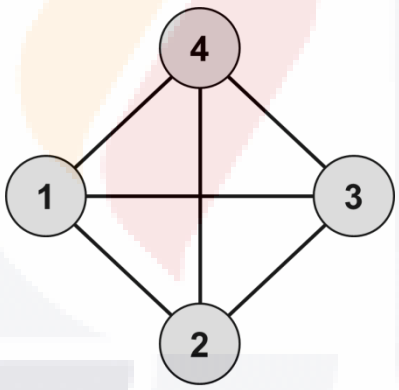


Figura 5.1 Grafico que Muestra los Tours para el Problema del Agente Viajero.

La **Figura 5.1** es de un grafo sencillo conformado por cuatro puntos (con las etiquetas 1, 2, 3 y 4). En donde cada punto se encuentra conectado a los demás, de manera que el vendedor puede tomar cualquiera de los caminos. Esto se traduce en que un recorrido por todos los puntos podría ser $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$ (sería la notación para un recorrido completo, donde el agente viajero inicia y termina en el mismo punto), otro recorrido

podría ser $1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1$. Cada uno de estos recorridos es sub-optimo, Siendo uno de los dos recorridos óptimos $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$.

El TSP está en el grupo de problemas NP-Duros, y se han ideado distintos algoritmos y heurísticas para resolverlo como los algoritmos de ramificación y acotamiento o algoritmos que realizan mejoras progresivas. Los algoritmos de colonia de hormigas quizá no puedan encontrar el recorrido óptimo, pero seguramente encontraran uno muy bueno.

Una ventaja de utilizar algoritmos de colonia de hormigas para el problema del agente viajero es que soporta cambios dinámicos en la ubicación de los puntos. Por ejemplo, si al problema se le agregan, quitan o cambian de ubicación los puntos, el algoritmo de colonia de hormigas puede seguir trabajando continuamente adaptándose a los cambios realizados de manera dinámica, proporcionando una buena solución.

5.3 Análisis del Algoritmo de Colonia de Hormigas.

Los componentes de la optimización basada en colonia de hormigas son: el grafo que es una representación gráfica que sirve para identificar los puntos a visitar, asumiendo que éstos se encuentran totalmente conectados entre sí, y una población de hormigas que son un conjunto de agentes artificiales donde cada uno almacena la información de sus movimientos, los puntos que ha visitado y cuales le restan por visitar. En este proceso las longitudes de cada recorrido se van calculando conforme se van visitando nuevos puntos, éstos son los elementos fundamentales utilizados en un algoritmo de colonia de hormigas.

El algoritmo de colonia hormigas consiste en una serie de pasos que se repiten durante un determinado número de iteraciones o periodo de tiempo, los pasos son los siguientes:

- **Colocación de las Hormigas en los Puntos del Grafo.**
- **Construcción de Recorridos por las Hormigas.**
- **Cálculo del Costo de los Recorridos.**

- **Evaporación del Rastro de Feromonas.**
- **Actualización del Rastro de Feromonas.**

Si se describiera en forma de pseudocódigo, el procedimiento de la metaheurística ACO tendría el control de las funciones principales hasta que terminara de realizar las iteraciones. El **Pseudocódigo 5.1** sería un ejemplo de la estructura general.

```
Inicio Procedimiento Metaheurística-ACO
    Mientras (iteraciones)
        Colocar_Hormigas ( )
        Construir_Recorridos ( )
        Calcular_Distancia_Recorridos ( )
        Evaporar_Feromonas ( )
        Actualizar_Feromonas ( )
    Fin Mientras
Fin Procedimiento
```

Pseudocódigo 5.1 Estructura General de una Metaheurística ACO.

5.3.1 Distribución de las Hormigas en el Grafo.

El algoritmo de colonia de hormigas inicia con la asignación de cada hormiga a un punto del grafo. Si el número de hormigas es menor al número de puntos la distribución puede ser de manera aleatoria, intentando siempre lograr la mejor distribución posible, pero para obtener mejores resultados se sugiere que el número de hormigas sea igual al número de puntos en el grafo (Dorigo & Stützle, Ant Colony Optimization., 2004.) (Jones, 2010.). Una vez que las hormigas han sido asignadas a un punto, el siguiente paso es que cada hormiga visite cada uno de los puntos para construir un recorrido completo, este paso involucra la selección de la mejor ruta a seguir por una hormiga basándose en la cantidad de feromonas y en la distancia del siguiente punto a visitar.

5.3.2 Selección de la Mejor Ruta.

La manera en que una hormiga selecciona el siguiente punto de su recorrido se basa en una sencilla ecuación de probabilidad. En la **Ecuación 5.1** se calcula la probabilidad de que una hormiga *k* seleccione moverse de un punto *i* a un punto *j*. Cuando la hormiga se encuentra en el punto inicial de su recorrido se debe considerar que los siguientes puntos candidatos a ser visitados tendrán la misma probabilidad de ser seleccionados.

$$P_{ij}^{k} = \frac{(\tau_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta}}{\sum_{l \in N_i^k} (\tau_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta}} \quad , \quad Si \quad j \in N_i^k$$

Ecuación 5.1 Cálculo de la Probabilidad de Seleccionar el punto *j* a partir de *i*.

La cantidad de feromona que existe entre los puntos *i* y *j* se conoce como rastro de feromona artificial y es el atractivo de visitar un punto *j* después de un punto *i* y está representado por τ_{ij} (Tao). También se maneja un valor heurístico entre la posición actual *i* y la posición candidata a visitar *j*, este valor heurístico está representado por η_{ij} (Eta), su valor es inversamente proporcional a la distancia entre 2 puntos y se calcula con $\eta_{ij} = 1/d_{ij}$. Otros parámetros utilizados son α (Alfa) y β (Beta) que indican el nivel de importancia, el primero de la intensidad del rastro de feromona y el segundo de la información heurística. Si $\alpha = 0$ significa que los puntos más cercanos serán seleccionados, en cambio sí $\beta = 0$ significa que los puntos con mayor concentración de feromonas serán los seleccionados, N_i^k es el conjunto de puntos vecinos aun no visitados por una hormiga *k* que se encuentra en el punto *i*.

Este paso se realiza hasta que cada hormiga haga un recorrido completo visitando todos los puntos del grafo. Cuando los recorridos se han completado, se gestiona la feromona en el grafo como se define en los siguientes 2 pasos de evaporación y actualización de feromona.

5.3.3 Evaporación de Feromona.

Una vez que todas las hormigas terminan de construir sus recorridos, el método de optimización realiza 2 operaciones en donde se modifica el valor de la feromona en los recorridos. El primer paso es disminuir la cantidad de feromona en todas las aristas por un factor constante, esto se conoce como evaporación. La evaporación evita el aumento del rastro de feromonas en todos los caminos y ayuda a mejorar la construcción de recorridos, la evaporación imita el proceso cuando las hormigas dejan de utilizar ciertos caminos y factores como el viento provocan la disminución de la concentración de feromonas. La fórmula utilizada para la evaporación de feromonas se muestra en la siguiente **Ecuación 5.2**.

$$\tau_{ij} = (1-\rho) \cdot \tau_{ij} \ , \forall (i, j) \in \mathrm{L}$$

Ecuación 5.2 Cálculo para la Evaporación de Feromonas del Trayecto i a j.

En la **Ecuación 5.2** aparece el parámetro ρ (Ro) conocido como tasa de evaporación de feromona, este parámetro sirve para evitar la acumulación ilimitada de feromonas, además ayuda a las hormigas a no volver a tomar caminos que anteriormente dieron malos resultados, en resumen si una arista deja de ser recorrida por las hormigas su valor de feromona disminuye con el paso de las iteraciones. El valor de este parámetro debe estar entre $0 < \rho < 1$. Después de realizar la evaporación, el siguiente paso es incrementar la feromona en las aristas utilizadas por las hormigas en sus recorridos.

5.3.4 Actualización de Feromona.

Después de disminuir el valor de la feromona en los caminos no utilizados, el siguiente paso es hacer un incremento del valor actual de feromonas por los puntos utilizados en los recorridos de las hormigas que dieron un buen resultado, con el objetivo de aumentar las probabilidades de que esos caminos sean utilizados posteriormente por otras hormigas. La

actualización de feromonas imita el proceso cuando las hormigas encuentran un camino favorable entre su nido y la fuente de alimento, como resultado recorren ese camino un mayor número de veces y al mismo tiempo intensifican el rastro de feromonas. Está actualización de feromonas está implementada por la **Ecuación 5.3**.

$$\tau_{ij} = \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \ , \ \forall (i, j) \in \mathrm{L}$$

Ecuación 5.3 Cálculo de la Actualización de Feromonas del Trayecto i a j.

En la Ecuación de arriba, $\Delta\tau_{ij}^{k}$ es el incremento de feromonas que se va aplicar en las aristas utilizadas en los recorridos de las hormigas, su valor está definido de la siguiente manera.

$$\Delta\tau_{ij}^{k} = \begin{cases} 1/\mathrm{L}^{k} & Si\,(i, j)\,\text{pertenecen a}\,\mathrm{T}^{k}. \\ 0 & Si\,no. \end{cases}$$

Ecuación 5.4 Cálculo del Incremento de Feromona en el Trayecto i a j.

En la **Ecuación 5.4** se establece la cantidad de feromona que se va agregar a las aristas, donde L^{k} es la longitud, del recorrido T^{k}, construido por la hormiga *k*, se calcula cómo la suma de las longitudes de las aristas que pertenecen al recorrido T^{k}. Esto significa que el incremento de feromonas es proporcional a la longitud del recorrido. Mientras que para un recorrido corto el valor de feromonas que se actualice en las aristas es mayor, para un recorrido muy largo el valor de feromonas que con el que se actualizarán las aristas que forman el recorrido será menor.

5.3.5 Nuevos Recorridos.

Una vez que todas las hormigas realizan un recorrido completo visitando todos los puntos del grafo, se realiza la evaporación de las feromonas en los caminos formados por las hormigas y se intensifica el rastro de feromonas para mejorar la construcción de nuevos recorridos. Una vez realizados estos pasos lo siguiente es organizar nuevamente a las hormigas para comenzar un nuevo recorrido.

A diferencia de la construcción del primer recorrido, en la construcción de los recorridos posteriores se debe tener en cuenta que, cómo las feromonas colocadas en el camino se calculan en base a la longitud del recorrido, las hormigas de forma probabilística comenzaran a recorrer las rutas con valores más altos de feromona, pero también de manera aleatoria algunas hormigas elegirán caminos alternativos.

De esta forma y dados los parámetros α y β para la construcción de mejores recorridos, la cantidad de feromona depositada será mayor en los mejores recorridos y disminuirá en los peores recorridos. Las hormigas realizaran un determinado número de recorridos, ya sea basándose en un máximo de recorridos o iteraciones establecidos en el algoritmo por el usuario, o después de un determinado lapso de tiempo en que el algoritmo no sea capaz de encontrar un mejor recorrido.

5.4 Algoritmos de Colonia de Hormigas Aplicado para Resolver el Problema de Ruteo Vehículos.

El problema de ruteo de vehículos o VRP es un problema enfocado a las tareas de distribución (Toth & Vigo, 2002.). Una variante del VRP clásico es el problema de ruteo de vehículos capacitado o CVRP en donde un número de clientes, representados por $(v_1,...v_n)$ tienen que ser atendidos por un depósito central, que normalmente se identifica

como el punto v_0. Cada cliente v_i maneja una cantidad de demanda q_i de un producto y por cada par de clientes (i, j) se maneja un costo de viaje d_{ij} entre los 2 clientes.

Los clientes son atendidos por una flota de vehículos con una capacidad Q. El objetivo del CVRP es encontrar un conjunto de rutas que reduzca el tiempo total de viaje respetando las siguientes restricciones.

1.-Cada cliente debe ser atendido una sola vez por un vehículo.

2.-Cada vehículo debe comenzar y terminar su ruta en el depósito.

3.-La demanda de los clientes cubiertos por un vehículo no debe superar su capacidad.

El CVRP contiene el problema del agente viajero o TSP como subproblema, por lo que se convierte en un problema del tipo NP-Duro. En la práctica, el VRP es mucho más difícil de resolver ya que está constituido por 2 problemas, el primero es un problema de tipo bin-packing cuyo objetivo es agrupar los clientes a visitar en un número desconocido de rutas. Después para cada una de las rutas creadas se tiene que buscar el recorrido más corto para visitar a todos los clientes asignados a cada ruta, significa resolver un problema del tipo TSP.

La importancia de estudiar el CVRP es muy importante ya que representa una oportunidad real para las empresas de reducir costos en áreas logísticas, la logística se puede describir como la entrega mercancía de parte de un proveedor a los consumidores. La gestión del transporte y la planeación de las rutas para los vehículos tienen un impacto económico considerable en todos los sistemas logísticos.

Debido a la naturaleza y complejidad del problema, no es viable utilizar métodos exactos para instancias grandes del CVRP. En su lugar algunos métodos de solución se han basado

en técnicas heurísticas desarrolladas específicamente para este problema y que proporcionan soluciones aproximadas, otra opción ha sido utilizar técnicas de optimización como la búsqueda tabú, recocido simulado, algoritmos genéticos o algoritmos basados en colonia de hormigas. Este trabajo tiene un interés en el uso de técnicas metaheurísticas utilizadas para resolver el CVRP y de manera más específica en el sistema de optimización basado en colonias de hormigas.

El primer algoritmo para resolver el CVRP que utilizó el método de optimización basado en colonia de hormigas se llamó Ant System (AS) y fue propuesto por (Bullnheimer, Hartl, & Strauss, Applying the Ant System to the Vehicle Routing Problem., 1997.); Donde el rastro de feromona y el valor heurístico del punto más cercano se utilizan para construir las rutas de los vehículos.

Para mejorar las rutas el Ant System se combinó con una función de búsqueda local llamada 2-Opt. Este algoritmo se probó en diferentes instancias de referencia y a pesar de que sus resultados fueron inferiores a los de un algoritmo utilizando búsqueda tabú, en términos de tiempo el Ant System genera resultados satisfactorios para convertirse en una metaheurística a considerarse.

Los mismos autores (Bullnheimer, Hartl, & Strauss, An Improved Ant System Algorithm for the Vehicle Routing Problem., 1999.) que presentaron el Ant System presentaron después una mejora que consistía en sustituir en la etapa de transición la heurística del vecino más cercano utilizada en el Ant System original por la heurística de Ahorro de (Paessens, 1988.), en la etapa de selección del siguiente punto a visitar se utilizó una heurística de búsqueda local, al final de cada iteración se aplicó una segunda heurística (2-opt) para mejorar la construcción de las rutas. Esta versión modificada dio mejores resultados a la original, sin embargo aún contaba con limitaciones y no siempre lograba los mejores resultados.

Otro algoritmo parecido al de Bullnheimer fue desarrollado por (Doerner, Gronalt, Hartl, Reimann, Strauss, & Stummer, 2002.), este algoritmo utilizaba como base el Ant System original y se utilizó la heurística de Ahorros por (Clarke & Wright, 1964.). Pero los resultados obtenidos por esta nueva versión no presentaron mejoras significativas a los anteriores.

5.5 Ventajas y Desventajas de un Algoritmo de Colonia de Hormigas.

La aplicación correcta de una metaheurística tiene como ventaja lograr obtener buenas soluciones que si bien no son la solución óptima del problema en cuestión, su calidad así como el tiempo de procesamiento que ahorran es motivo suficiente para ser una técnica a tomar en cuenta.

Además hablando específicamente de la optimización basada en colonia de hormigas, vemos en la función de evaporación del rastro de feromonas en los caminos recorridos por las hormigas artificiales un mecanismo que ayuda a evitar el estancamiento del algoritmo en resultados óptimos locales, ya que permite al algoritmo realizar una exploración más amplia de las posibles rutas que se puedan tomar e ir construyendo una mejor solución.

6. Unidades de Procesamiento de Gráficos (GPU´s) y la Arquitectura de Programación en Paralelo CUDA

6.1 Concepto de Unidad de Procesamiento de Gráficos.

La unidad de procesamiento de gráficos (graphics processing unit - GPU) es un coprocesador dedicado a realizar una enorme cantidad de operaciones de coma flotante, con el objetivo de apoyar a la unidad central de procesamiento (central processing unit - CPU) en aplicaciones como videojuegos, herramientas de diseño asistido por computadora, edición de video e imágenes y diseño en 3D.

A mediados de la década de los 90´s la demanda de gráficos por parte de los usuarios aumentó con la aparición de juegos cada vez más realistas, pero debido a la arquitectura de las CPU´s los resultados no eran satisfactorios, debido a esto, algunas empresas como NVIDIA, ATI Technologies y 3dfx Interactive iniciaron la comercialización de tarjetas aceleradoras gráficas, y así todas las tareas relacionadas con los gráficos y aplicaciones 3D

interactivas pasaron a ser exclusivamente de la GPU, permitiendo a la CPU dedicarse a otras operaciones.

Las primeras GPU estaban enfocadas en la aceleración 2D para sistemas de escritorio así como la resolución de monitores. Con el tiempo han surgido nuevas tecnologías como la aceleración 3D, funciones gráficas para los sistemas operativos como DirectX y OpenGL o motores para realizar cálculos físicos, diseñados para los juegos de computadora y la visualización de datos. El veloz ritmo de desarrollo de las GPU´s se debe a dos características principales. La primera es la alta especialización de las GPU´s ya que solamente están pensadas para realizar una sola tarea, las GPU´s actuales están optimizadas para realizar cálculos con valores de punto flotante (Floating Point - FP) que predominan en aplicaciones gráficas 3D. Por esta razón se ha dedicado más trabajo en mejorar sus diseños para que realicen su trabajo de manera más eficiente,

La segunda característica es que muchas aplicaciones graficas manejan un alto grado de paralelismo, ya que sus unidades de cálculo como texturas, vértices, iluminación y pixeles son completamente independientes. Por esto siempre se intenta utilizar todas las unidades de cálculo de la GPU para completar los cálculos más rápido y al mismo tiempo.

La GPU se ha convertido en una unidad poseedora de varios núcleos con funciones altamente paralelas, capaces de controlar múltiples hilos, con mayor cantidad de memoria con anchos de banda más veloces para almacenar y transmitir información. Además mientras que la frecuencia de reloj de las CPU´s actuales rondan los 3.8 – 4 GHz no significa que sean más eficientes que las GPU´s actuales con frecuencias de reloj entre los 600 GHz – 1Ghz, ya que gracias a su arquitectura en paralelo su potencia de cálculo es mucho mayor.

6.2 Historia de la GPU de Propósito General o GPGPU.

En sus inicios las GPU´s fueron diseñadas como procesadores gráficos, soportando únicamente funciones fijas. A finales de los años 90´s este hardware se hizo más sencillo de programar, y dio como resultado a la primera GPU de NVIDIA en 1999 (Historia de Nvidia: de la tarjeta gráfica al procesador móvil., 2014), a partir de ese momento artistas, desarrolladores de videojuegos así como científicos comenzaron a utilizar las GPU´s para acelerar sus aplicaciones aprovechando el enorme rendimiento en operaciones de punto flotante, y es aquí donde dio inicio el GPU de propósito general (General Purpose GPU - GPGPU) (Plataforma de Computacion Paralela CUDA.).

Pero el GPU de propósito general no era sencillo, inclusive para las personas que conocían el manejo de librerías graficas como OpenGL y CG, se encontraban con dificultades para programar las GPU´s y representar sus problemas con triángulos y polígonos, esta complejidad limitaba el acceso a la enorme capacidad de las GPU en el campo científico.

En el año 2003 un equipo de investigadores junto con Ian Buck dieron a conocer Brook (BrookGPU: Introduction, 2014), fue el primer modelo de programación ampliamente adoptado que extendía el lenguaje C con construcciones de datos en paralelo. Utilizando conceptos como streams, núcleos, operadores de reducción, el compilador Brook y el tiempo de ejecución del sistema permitieron utilizar a la GPU como un procesador de propósito general bajo un lenguaje de alto nivel. Lo más importante era que los programas de Brook no solo eran más sencillos de escribir que el código estándar de GPU, sino que eran mucho más rápidos que códigos similares ya existentes.

NVIDIA sabía que un hardware con tanta potencia tenía que hacerse acompañar de un software intuitivo de desarrollo, con este motivo invitó a Ian Buck para unirse a la empresa y trabajar en una solución para ejecutar sin problemas lenguaje C en la GPU, de esta manera se pudo unir el hardware con el software y en el año 2006 NVIDIA presentó

CUDA (Historia de Nvidia: de la tarjeta gráfica al procesador móvil., 2014), una solución para el desarrollo del GPU de propósito general.

6.3 CUDA.

CUDA es una interfaz de programación de aplicaciones (Application Programming Interface - API) creada por la Compañía NVIDIA para el cálculo de datos en paralelo que aprovecha el poder de las GPU´s para incrementar el rendimiento de los equipos de cómputo (Plataforma de Computacion Paralela CUDA.), algunas características de CUDA son su facilidad de uso ya que no es necesario tener conocimientos sobre las API´s gráficas, también es compatible con cualquier sistema operativo en donde se tenga instalada una GPU NVIDIA, es compatible con los compiladores de lenguaje C más populares como son el de la compañía Microsoft con Visual Studio y gcc.

La medición del rendimiento de los equipos de cómputo se hace por medio de las operaciones de coma flotante por segundo (Floating Point Operations Per Second - FLOPS) que son capaces de realizar.

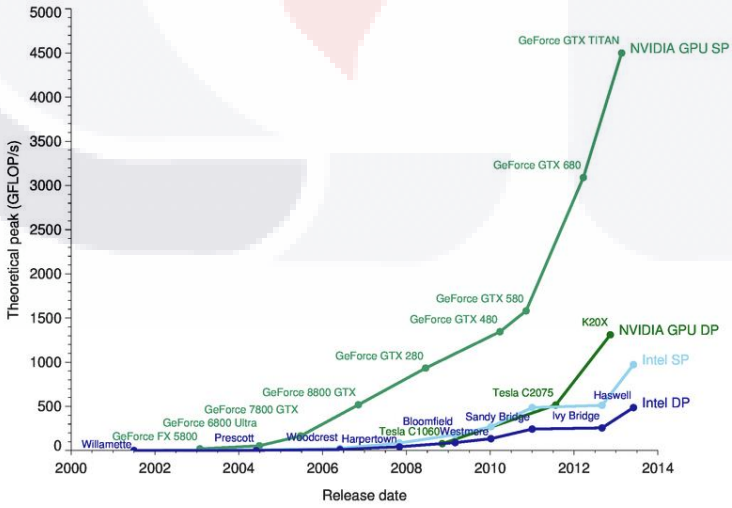


Figura 6.1 Incrementos del Número de Operaciones de Coma Flotante por Segundo.

En la **Figura 6.1** se muestran los modelos de CPU´s y GPU´s aparecidos entre los años 2000 y 2014 junto al número de FLOPS que pueden alcanzar, se pueden apreciar 2 cosas una es el notable incremento de FLOPS que han tenido las GPU´s y también la gran diferencia de FLOPS alcanzadas entre GPU´s y CPU´s tanto en operaciones de precisión sencilla (Single Precision - SP) y doble (Double Precision - DP).

La tecnología CUDA es utilizada por desarrolladores, científicos e investigadores y tiene aplicaciones prácticas en diversos campos como el procesamiento de video e imágenes, biología y química computacional, simulación de la dinámica de fluidos, reconstrucción de imágenes de tomografías computarizadas, análisis sísmicos o el trazado de rayos, análisis de grandes cantidades de información, algoritmos de búsqueda exhaustiva, entre muchas otras.

Es necesario un kit de desarrollo de software para trabajar en la plataforma de cálculo paralelo CUDA este kit se integra a entornos de desarrollo de lenguajes C y C++ e incluye un compilador, bibliotecas matemáticas, guías de programación, manuales para el usuario, herramientas para corregir y optimizar el rendimiento de aplicaciones, todo lo necesario para implementar el paralelismo en el procesamiento de instrucciones y datos a distintos niveles. Todo esto permite que los desarrolladores puedan implementar sus aplicaciones bajo la arquitectura CUDA utilizando diferentes lenguajes de programación como C, C++ y Fortran los cuales son soportados de manera oficial por NVIDIA y otros de manera no oficial como Python, CIL, Java y Matlab mediante software de terceros.

6.3.1 Requisitos de CUDA.

Para poder ejecutar aplicaciones CUDA en un sistema de cómputo se necesita contar con los siguientes requisitos de Hardware y Software.

Sistemas Operativos Windows:

- GPU con soporte para CUDA.
- Sistema Operativo Windows XP, Vista, 7, 8, Server 2003, Server 2008.
- Herramientas de Desarrollo CUDA.
- Compilador de lenguaje C incluido en Microsoft Visual Studio 2008, 2010 o 2012.

Sistemas Operativos Mac:

- GPU con soporte para CUDA.
- Sistema Operativo Mac OS X versión 10.7.5 o superior.
- Compiladores gcc o Clang. Y tener instalada la herramienta toolchain utilizando Xcode.
- Herramientas de Desarrollo CUDA.

Sistemas Operativos Linux:

- GPU con soporte para CUDA.
- Sistema Operativo Linux con soporte para CUDA.
- Compilador gcc y herramienta toolchain.
- Herramientas de Desarrollo CUDA.

6.3.2 Instalación y Configuración de CUDA en Sistemas Operativos Windows.

La instalación de las herramientas de desarrollo CUDA y su configuración en un equipo de cómputo con sistema operativo Windows, consiste en los siguientes pasos:

Paso 1: Verificar que el sistema posea una GPU que soporte CUDA https://developer.nvidia.com/cuda-gpus.

Paso 2: Instalar un compilador de lenguaje C, C++ o una interfaz de desarrollo integrado como Visual Studio 2010 o Eclipse.

Paso 3: Actualizar los controladores de la tarjeta gráfica NVIDIA http://www.nvidia.com/Download/index.aspx?lang=en-us.

Paso 4: Instalar las herramientas de desarrollo de CUDA https://developer.nvidia.com/cuda-downloads que permiten la compilación de los programas en CUDA y además contienen ejemplos para comenzar a programar.

Paso 5: Realizar una prueba de la instalación compilando y ejecutando alguno de los ejemplos de CUDA.

6.3.3 Organización en CUDA.

Para entender la programación bajo la arquitectura CUDA es necesario ver algunos conceptos como la capacidad computacional, hilos, bloques, mallas y núcleos.

La capacidad computacional (Compute Capability): es la que describe las características soportadas por el hardware CUDA. Las primeras series de GPU capaces de ejecutar instrucciones CUDA fueron los modelos GeForce 8800 GTX las cuales tenían una capacidad computacional de 1.0 algunas GPU más recientes tiene una capacidad computacional de 5.0.

Algunas de las características descritas a continuación están basadas en un hardware CUDA con capacidad computacional de 3.0.

Hilo (Thread): Es la unidad de trabajo básica de un núcleo y son asignados a núcleos de procesamiento de la GPU, cada hilo ejecuta el mismo código y opera sobre distintos valores. Es importante además conocer el término y la estructura de un “warp”, son las

unidades utilizadas en los sistemas de procesamiento en paralelo formados por grupos de 32 hilos y son útiles para efectos de rendimiento.

Bloque (Block): Es un arreglo de hilos los cuales trabajan de manera sincronizada y puede compartir información entre ellos mediante la memoria compartida. Los bloques se pueden formar de 1 a 3 dimensiones, manejar un máximo de 1024 hilos con un tamaño de x = 1024, y = 1024 y z = 64 por dimensión.

Malla (Grid): Es un arreglo bidimensional de bloques de hilos independientes, es el lugar de ejecución de un núcleo. Las mallas se manejan en 1 y 2 dimensiones con tamaños de x = 2147483647, y = 65535 y z = 65535 por dimensión.

Núcleo (Kernel): estas funciones especifican la parte de código que será ejecutada en la GPU de manera simultánea, al declarar un núcleo se debe especificar la malla esto es, el número de bloques e hilos que ejecutaran las mismas instrucciones sobre valores diferentes. Los núcleos están escritos en variantes de lenguaje C/C++.

6.3.4 Programa en CUDA.

Los pasos básicos de un programa en lenguaje C con funciones o núcleos en CUDA son los siguientes:

1. Almacenar la información del programa en la memoria principal del equipo de cómputo.
2. Copiar la información de la memoria principal del equipo de cómputo hacia la memoria de la GPU.
3. Realizar el procesamiento de la información en paralelo dentro de la GPU.
4. Copiar la información de la memoria de la GPU hacia la memoria principal del equipo de cómputo.
5. Mostrar los resultados de la información procesada.

Para entender el modelo de programación en CUDA, a continuación se muestra un código a manera de ejemplo, este código contiene las instrucciones para realizar el cálculo de la suma de dos arreglos unidimensionales utilizando la GPU y cómo almacena los resultados en un tercer arreglo.

Primero en el **Código 6.1** se declara una constante N con un valor de 10000 que servirá para definir la longitud de los arreglos a [ ] y b [ ] que contienen los valores a sumar y que se guardaran en un arreglo c [ ].

```
#define N   10000
```

Código 6.1 Declaración de una Constante N con Valor Entero 10000.

A continuación en el **Código 6.2** se declara una función llamada add que sumará los valores de los arreglos a [ ] y b [ ], se puede saber que se trata de una función que se ejecutará dentro de la GPU por el código __global__ agregado en la declaración de la función.

Esta función maneja una variable tid que es un índice para manejar la información de cada uno de los hilos. El objetivo de la función es sumar los valores de los arreglos a [tid] + b [tid] y guardarlos en c [tid].

```
__global__ void add( int *a, int *b, int *c ) {
   int tid = blockIdx.x;    // this thread handles the data at its thread id
   if (tid < N)
      c[tid] = a[tid] + b[tid];
}
```

Código 6.2 Kernel que Realiza la Suma de los Vectores a y b.

En el **Código 6.3** se muestra la función principal del programa, para iniciar se declaran los arreglos a [ ], b [ ] y c [ ] de tipo entero con una longitud N y también se declaran tres apuntadores llamados *dev_a, *dev_b y *dev_c para gestionar la información entre la CPU y GPU.

```
int main( void ) {
   int a[N], b[N], c[N];
   int *dev_a, *dev_b, *dev_c;
```

Código 6.3 Declaración de Arreglos y Apuntadores utilizados en la Aplicación.

El siguiente paso es realizar la asignación de espacios de memoria en la GPU utilizando la instrucción cudaMalloc mostrado en el **Código 6.4**.

```
   // allocate the memory on the GPU
   cudaMalloc( (void**)&dev_a, N * sizeof(int) );
   cudaMalloc( (void**)&dev_b, N * sizeof(int) );
   cudaMalloc( (void**)&dev_c, N * sizeof(int) );
```

Código 6.4 Asignación de Espacios de Memoria en la GPU.

En el **Código 6.5** se muestra un ciclo de 1 hasta N, que llenará los arreglos a [ ] con números negativos y b [ ] con los números elevados al cuadrado.

```
   // fill the arrays 'a' and 'b' on the CPU
   for (int i=0; i<N; i++) {
      a[i] = -i;
      b[i] = i * i;
   }
```

Código 6.5 Llenado de Vectores con Valores en la CPU.

El siguiente paso mostrado en el **Código 6.6** es copiar los arreglos a [ ] y b [ ] ya con los valores numéricos negativos y elevados al cuadrado, utilizando la instrucción cudaMemcpy y copiar los arreglos de la memoria principal o host hacia la memoria de la GPU o device.

```
// copy the arrays 'a' and 'b' to the GPU
cudaMemcpy( dev_a, a, N * sizeof(int), cudaMemcpyHostToDevice );
cudaMemcpy( dev_b, b, N * sizeof(int), cudaMemcpyHostToDevice );
```

Código 6.6 Copia de Vectores desde la CPU a la GPU.

Se hace la llamada de ejecución de la función en la GPU mediante add<<<N,1>>> siempre al llamar un núcleo de CUDA se deben especificar el tamaño de la malla con su número de bloques e hilos como se muestra en el **Código 6.7**, en este ejemplo se está especificando un tamaño de la malla de N (10000) y cada bloque estará formado por 1 hilo, por último se especifican los parámetros que serán enviados a la GPU.

```
add<<<N,1>>>( dev_a, dev_b, dev_c );
```

Código 6.7 Llamada de Ejecución del Kernel para Realizar la Suma de Vectores.

Una vez que terminó la ejecución de la función add, se utiliza otra instrucción llamada cudaMemcpy que ayuda a copiar de regreso el arreglo con los resultados de la suma, especificando que se copiará desde la memoria de la GPU o Device hacia la memoria principal o host, se muestra en el **Código 6.8**.

```
// copy the array 'c' back from the GPU to the CPU
cudaMemcpy( c, dev_c, N * sizeof(int), cudaMemcpyDeviceToHost );
```

Código 6.8 Copia de Vectores desde la GPU a la CPU.

Una vez que se tiene los resultados en el arreglo c [ ], se ejecuta un ciclo para imprimir los valores sumados y su resultado en pantalla como el mostrado en el **Código 6.9.**

```
// display the results
for (int i=0; i<N; i++) {
    printf( "%d + %d = %d\n", a[i], b[i], c[i] );
}
```

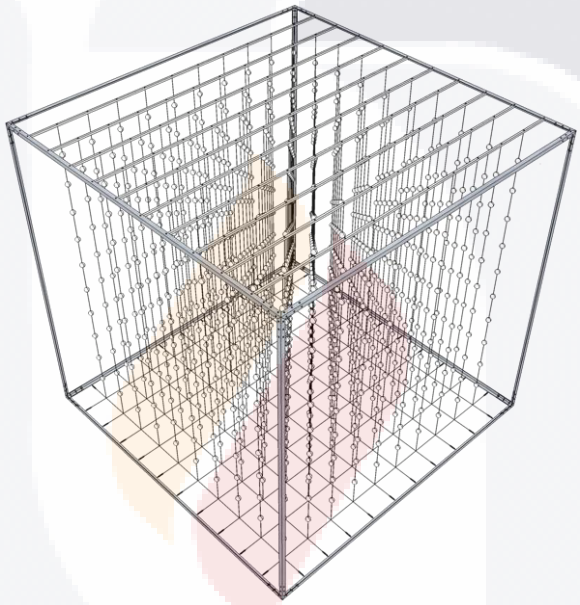
Código 6.9 Impresión de los Resultados de la Suma de Vectores.

Un paso importante es que se debe liberar los espacios de memoria reservados en la GPU hecho para los arreglos, mediante la instrucción cudaFree, esta acción se muestra en el **Código 6.10.**

```
// free the memory allocated on the GPU
cudaFree( dev_a );
cudaFree( dev_b );
cudaFree( dev_c );
getchar();
return 0;
}
```

Código 6.10 Liberación de Memoria de la GPU.

7.- Método de Solución para el Problema de Ruteo de Vehículos con Capacidad Limitada



Como resultado de los objetivos en este trabajo, se propone la implementación de un algoritmo en paralelo para solucionar problemas de ruteo de vehículos con capacidades limitadas o CVRP. En la **Figura 7.1** se puede observar la organización básica de las entradas, funciones y salida del algoritmo.

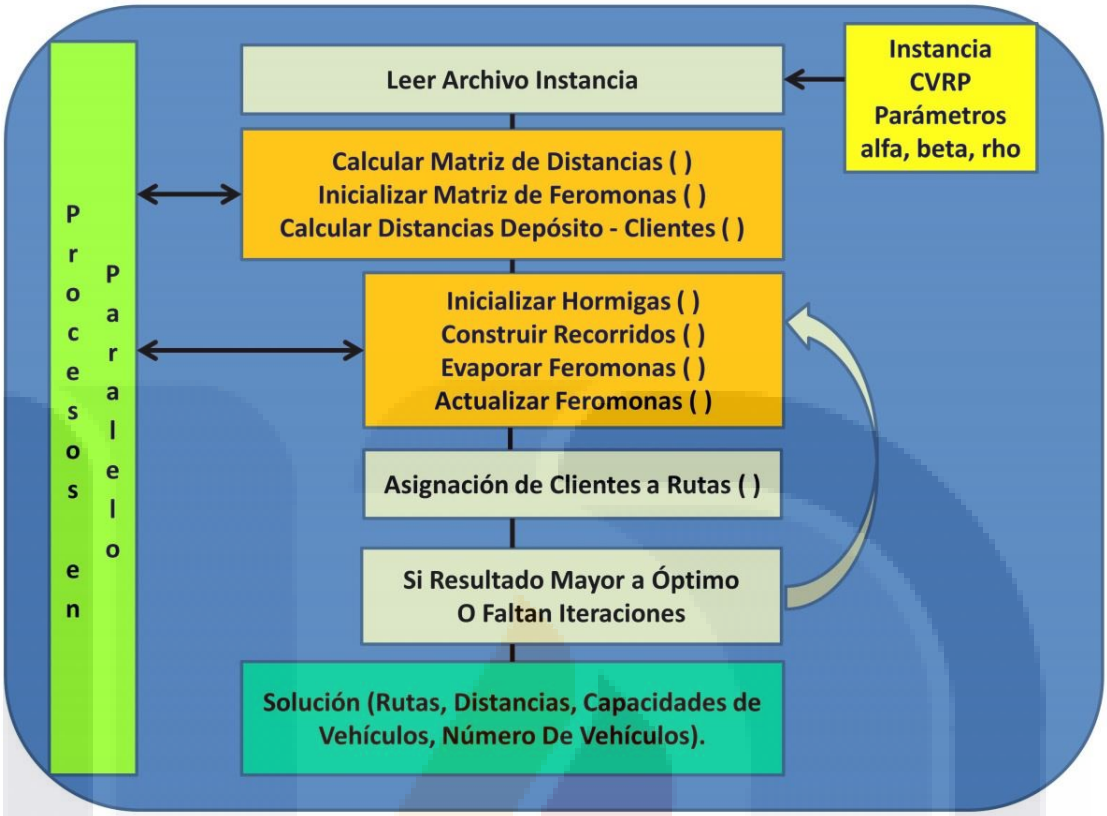


Figura 7.1 Estructura Básica del Algoritmo Implementado.

En este algoritmo se pueden resaltar 3 características principales:

1) La implementación a manera de núcleos de ejecución en paralelo de algunas funciones del algoritmo de colonia de hormigas como son el cálculo de la matriz de distancias entre los clientes, el cálculo de las distancias entre depósito y clientes, la inicialización de valores para la matriz de feromonas, la evaporación del rastro de feromonas e intensificación del rastro de feromonas, con el objetivo de aprovechar las capacidades de procesamiento de la GPU y reducir el tiempo de ejecución.

2) Se utiliza el algoritmo de colonia de hormigas como herramienta para generar varios recorridos y encontrar el recorrido con las distancias más cortas entre los clientes.

3) Dentro del Algoritmo de colonia de hormigas se implementó el método Rutear Primero – Asignar Después de Beasley (Beasley, 1983.) Utilizado para resolver problemas de ruteo de vehículos. Mediante el cual se analiza cada recorrido y se realiza la asignación de clientes a cada ruta de vehículos.

En la **Figura 7.2** se muestran las herramientas principales utilizadas en el método de solución. Teniendo como base la arquitectura de programación en paralelo, posteriormente el algoritmo de optimización y finalmente un método para asignar los clientes a las rutas.

Figura 7.2 Herramientas Utilizadas para el Método de Solución.

Teniendo los puntos anteriores como una visión general del método propuesto, la estructura general del algoritmo implementado para resolver problemas CVRP es la siguiente:

```
Leer Archivo Instancia
Generar Ruta de Menor Distancia mediante Algoritmo de Colonia de Hormigas
    Algoritmo de Hormigas
        Calcular Matriz_distancias ( ) CUDA
        Inicializar Matriz_feromonas ( ) CUDA
        Calcular Distancia_Deposito_Clientes ( ) CUDA
        Mientras (iteraciones del algoritmo de colonia de hormigas)
            Inicializar_Hormigas ( )
            Construir_Recorridos ( )
            Evaporar_Feromonas ( ) CUDA
            Actualizar_Feromonas ( ) CUDA
            Asignar_Clientes_Rutas ( )
        Fin Mientras
    Impresión de la Mejor Asignación de Clientes a Rutas
```

Pseudocódigo 7.1 Estructura General del Método de Solución.

El algoritmo propuesto, realiza en primer lugar la lectura de una instancia que contiene información sobre el problema cómo: autores, la solución óptima, el número de ciudades, la capacidad de cada vehículo, el número máximo de vehículos permitidos con los que se soluciona la instancia, en columnas se presentan las coordenadas de ubicación (x, y) de cada uno de los clientes y en otra la demanda de cada cliente.

La siguiente etapa del algoritmo es la generación de rutas con la menor distancia entre todos los clientes, para esto se utilizó un algoritmo de colonia de hormigas conformado por dos etapas principales, la primera es para calcular y almacenar información indispensable para efectuar los cálculos en la segunda etapa, en la cual se hacen los cálculos probabilísticos para ir construyendo las posibles soluciones.

Dentro de la primera etapa del algoritmo hay 3 funciones que se ejecutan dentro de la GPU de manera paralela, la primera función realiza el cálculo de las distancias euclidianas entre todos los clientes esto es, que para cada cliente calcula la distancia para llegar a los demás clientes de la instancia y estos resultados se almacenan en una matriz simétrica cuya diagonal principal es 0, la fórmula para calcular la distancia entre 2 puntos se muestra en la **Ecuación 7.1**.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Ecuación 7.1 Cálculo de la Distancia Euclidiana entre dos Puntos.

En operaciones posteriores esta información ya solamente es leída evitando así la realización de volver a realizar cálculos innecesarios que consuman tiempo.

```
__global__ void kernelPrecomputedistances(float *cities_x,
float *cities_y, double *precomputed_distance, int n)
{
      double dx, dy;
      int i = blockDim.x * blockIdx.x + threadIdx.x;
      int j = blockDim.y * blockIdx.y + threadIdx.y;
      int index= i * n + j;
      dx = abs(cities_x[i] - cities_x[j]);
      dy = abs(cities_y[i] - cities_y[j]);
      precomputed_distance[index]= sqrt((dx*dx)+(dy*dy));
}
```

Código 7.1 Kernel para Calcular la Distancia Euclidiana Entre los Clientes.

En seguida encontramos la función también paralela que inicializa la matriz que almacena los valores de feromona entre todas las ciudades.

```
__global__ void kernelPheromone(double *pheromone, int n){
      int i = blockDim.x * blockIdx.x + threadIdx.x;
      int j = blockDim.y * blockIdx.y + threadIdx.y;
      int index= i * n + j;
      pheromone[index] = BASE_PHEROMONE;
}
```

Código 7.2 Kernel para Inicializar el Nivel de Feromona.

La última función en esta etapa del algoritmo es el cálculo de distancias entre cada cliente y el depósito está información es necesaria al momento de asignar los clientes a cada vehículo, en el caso de que se trate del inicio de una ruta se agrega a la distancia recorrida por el vehículo la distancia del depósito al primer cliente o en caso de que sea el fin de la ruta se agrega la distancia del último cliente visitado hacia el depósito.

```
__global__ void kernelDepotdistances(float *x1, float *y1,
float x2, float y2, double *depot_distances)
{
      double dx, dy;
      int i = blockDim.x * blockIdx.x + threadIdx.x;
      dx = abs(x1[i] - x2);
      dy = abs(y1[i] - y2);
      depot_distances[i] = sqrt((dx*dx)+(dy*dy));
}
```

Código 7.3 Kernel para Calcular la Distancia Euclidiana entre el Depósito y los Clientes.

La segunda etapa son las iteraciones realizadas por el algoritmo de colonia de hormigas, dentro de este ciclo los agentes artificiales van realizando modificaciones en los valores almacenados en la matriz de feromonas, dependiendo de los recorridos que van formando.

La primera función es la inicialización de las hormigas artificiales, en este punto las hormigas se colocan aleatoriamente en todas las ciudades como puntos de inicio. La segunda función es la construcción, en esta parte cada una de las hormigas artificiales va construyendo su recorrido de puntos mediante fórmulas probabilísticas, tomando en cuenta los valores en la matriz de feromonas y el nivel de conveniencia de ir de un punto a otro, con esta información cada hormiga va decidiendo cuál será su próximo punto a visitar.

Una vez que los recorridos de las hormigas artificiales visitan todos los clientes, se ejecutan 2 tareas importantes la evaporación y la actualización del rastro de feromonas; la primera consiste en que para cada camino entre 2 clientes que no sea utilizado por las hormigas, a este camino se le aplica un decremento del valor de feromonas en la matriz correspondiente.

```
__global__ void kernelEvaporation(double *pheromone, double rho)
{
      int i = blockDim.x * blockIdx.x + threadIdx.x;
int j = blockDim.y * blockIdx.y + threadIdx.y;
int index = i * NUM_CITIES + j;
      pheromone[index] *= ( 1.0 - rho );
if (pheromone[index] < 0.000000)
            {
pheromone[index] = BASE_PHEROMONE;
}
}
```

Código 7.4 Kernel para Disminuir el Valor de la Matriz de Feromonas.

El segundo proceso es la actualización del valor de feromonas, consiste en que para todos los caminos entre 2 clientes utilizados por las hormigas se realizará un incremento en el valor de la matriz de feromonas, el incremento se hace en base a la longitud del camino, si la distancia del camino es muy larga su rastro será bajo, pero si la distancia del camino es corto el aumento del valor de feromona será alto.

```
__global__ void kernelIntensification(ANT_T *a, double
*pheromone, double rho, double qval, int cont)
{
      int from, to;
      int cx = blockDim.x * blockIdx.x + threadIdx.x;
      from=a[cont].tour[cx];
      to=a[cont].tour[(cx+1)%NUM_CITIES];
      if (cx<NUM_CITIES){
            pheromone[from * NUM_CITIES + to] += ((qval /
a[cont].tour_length) * rho);
            pheromone [to * NUM_CITIES + from] = pheromone[from
* NUM_CITIES + to];
      }
}
```

Código 7.5 Kernel para Aumentar el Valor de la Matriz de Feromonas.

En el momento en que todas las hormigas completan su recorrido (en el que visitan todos los clientes), se ejecuta la función para dividir los recorridos de cada hormiga y asignar los clientes a las rutas, asignando los clientes a los vehículos mientras éstos puedan satisfacer la demanda de los clientes siguientes, esta asignación está basada en el método Rutear Primero – Asignar Después.

Un ejemplo para explicar la división de los recorridos y cómo se asignan los clientes a las rutas de vehículos es el siguiente:

El problema consta de quince clientes para visitar, un depósito, el número máximo de vehículos permitidos son 8, la capacidad de cada vehículo es de 35 unidades y la distancia optima total, que es igual a la suma de las distancias recorridas por todos los vehículos es de 450.

El primer paso es generar un recorrido que visite todos los clientes (estos recorridos son generados por las hormigas artificiales), cada cliente tiene una cantidad de demanda asociada a él como se muestra en la **Figura 7.3**:

Clientes	15	12	10	2	7	9	13	8	3	1	6	5	14	4	11
Demanda	11	14	8	30	15	8	6	28	16	19	31	11	19	23	7

Figura 7.3 Recorrido de Clientes y sus Respectivas Cantidades de Demanda.

A partir del recorrido que se generó, se inicia la primera ruta partiendo del depósito hacia el primer cliente del recorrido en este ejemplo el número 15, el vehículo verifica si puede satisfacer la demanda del cliente si esto es posible, el cliente es asignado a la ruta y se le resta la demanda del cliente (demanda de 11 unidades) a la capacidad del vehículo quedando ahora con una capacidad de 24 unidades el resultado se muestra en la **Figura 7.4**.

Clientes	15	12	10	2	7	9	13	8	3	1	6	5	14	4	11
Demanda	11	14	8	30	15	8	6	28	16	19	31	11	19	23	7
Ruta 1	15					Capacidad Restante del Vehículo: 24									

Figura 7.4 Asignación del Cliente 15 a la Ruta 1.

El siguiente paso es ir al siguiente cliente el número 12, nuevamente se calcula si el vehículo cuenta con la capacidad suficiente para satisfacer la demanda del siguiente cliente, la capacidad del vehículo es de 24 unidades y la demanda es de 14 unidades por lo que sí es posible asignar el cliente a la ruta, esto es muestra en la **Figura 7.5**.

Clientes	15	12	10	2	7	9	13	8	3	1	6	5	14	4	11
Demanda	11	14	8	30	15	8	6	28	16	19	31	11	19	23	7
Ruta 1	15	12				Capacidad Restante del Vehículo: 10									

Figura 7.5 Asignación del Cliente 12 a la Ruta 1.

El vehículo en este punto tiene una capacidad restante de 10 unidades, y continúa con el cliente número 10, calcula si es posible satisfacer su demanda de 8, como esto es posible se asigna el cliente a la ruta, ver la **Figura 7.6**.

Clientes	15	12	10	2	7	9	13	8	3	1	6	5	14	4	11
Demanda	11	14	8	30	15	8	6	28	16	19	31	11	19	23	7
Ruta 1	15	12	10			Capacidad Restante del Vehículo: 2									

Figura 7.6 Asignación del Cliente 10 a la Ruta 1.

El vehículo continúa con el cliente número 2, calcula si puede satisfacer la demanda del cliente (demanda de 30 unidades), en este punto ya no es posible que la capacidad del vehículo pueda satisfacer su demanda, por lo que el siguiente paso es terminar la ruta actual regresando el vehículo al depósito.

A continuación se inicia una nueva ruta con otro vehículo con su capacidad de 35 unidades, partiendo del depósito hacia el cliente número 2, verifica si es posible satisfacer su demanda y lo agrega a la ruta número 2, se puede ver gráficamente en la **Figura 7.7**.

Clientes	15	12	10	2	7	9	13	8	3	1	6	5	14	4	11
Demanda	11	14	8	30	15	8	6	28	16	19	31	11	19	23	7
Ruta 1	15	12	10			Capacidad Restante del Vehículo: 5									
Ruta 2	2														

Figura 7.7 Asignación del Cliente 2 a la Ruta 2.

Este proceso continúa asignando los clientes a las rutas mientras los vehículos sean capaces de satisfacer su demanda, y se continúan iniciando y terminando rutas hasta llegar al final del recorrido, este ejemplo dio como resultado 8 rutas con sus respectivos clientes asignados como se puede ver en la **Figura 7.8**.

Clientes	15	12	10	2	7	9	13	8	3	1	6	5	14	4	11
Demanda	11	14	8	30	15	8	6	28	16	19	31	11	19	23	7
Ruta 1	15	12	10			Capacidad Restante del Vehículo: 5									
Ruta 2	2														
Ruta 3	7	9	13												
Ruta 4	8														
Ruta 5	3	1													
Ruta 6	6														
Ruta 7	5	14													
Ruta 8	4	11													

Figura 7.8 Fin de la Asignación de Clientes a las Rutas.

Una vez que se asignaron los clientes a cada ruta se realiza el cálculo de la distancia recorrida por cada ruta iniciando del depósito (indicado con 0) recorriendo los clientes y finalizando en el depósito nuevamente (en el método de Beasley (Beasley, 1983.), así como en las publicaciones en donde se muestran resultados y utilizan las mismas instancias de pruebas usadas en este trabajo, las distancias se redondearon al número entero más cercano, sin embargo también es posible utilizar valores reales si se necesita manejar mayor precisión), las distancias de cada ruta se suman para obtener la distancia total. En la **Figura 7.9** se muestra la distancia total de recorrido de todas las rutas es de 450.

						Distancia por Ruta
Ruta 1	0	15	12	10	0	67
Ruta 2	0	2	0			42
Ruta 3	0	7	9	13	0	68
Ruta 4	0	8	0			64
Ruta 5	0	3	1	0		66
Ruta 6	0	6	0			24
Ruta 7	0	5	14	0		62
Ruta 8	0	4	11	0		57
Distancia Total						450

Figura 7.9 Cálculo de la Distancia Total Recorrida por las Rutas.

El método original de Beasley solamente realizaba la división de clientes y su asignación a rutas para un solo recorrido generado de manera aleatoria, en el método que se está proponiendo se realizó una modificación en la que la división de clientes y su asignación a rutas se realiza para el mejor recorrido encontrado por cada una de las hormigas en cada iteración, además se le aplica una función de rotación para que la división y asignación de clientes tome como inicio cada uno de los clientes de la ruta, con el propósito de mejorar la calidad de la solución.

8.- Equipo de Cómputo e Instancias de Prueba Utilizadas en los Experimentos

8.1 Hardware y Software Utilizado en los Experimentos.

8.1.1 Workstation HP Z820.

Procesador: Intel Xeon E5-2609 @ 2.40GHz (2 Procesadores).

Memoria Instalada (RAM): 32.0 GB.

Almacenamiento: Disco duro de 1 TB**.**

Tarjetas Gráficas: NVIDIA QUADRO 6000, Memoria Total Dedicada 6 GB GDDR5, 448 núcleos CUDA, Capacidad computacional 2.0.

NVIDIA Tesla C2075, Memoria Total Dedicada 6 GB GDDR5, 448 núcleos CUDA, Capacidad computacional 2.0.

Sistema Operativo: Microsoft Windows 7 Professional de 64 bits con Service Pack 1.

Entorno de Desarrollo: Microsoft Visual Studio 2010 Profesional.

Herramientas de Compilación: NVIDIA CUDA SDK Versión 5.5.

Controladores De Video: NVIDIA Drivers Versión 333.11.

8.1.2 Portátil Lenovo IdeaPad Y400.

Procesador: Intel Core i7-3630QM CPU @ 2.40 GHz.

Memoria Instalada (RAM): 6.0 GB.

Almacenamiento. Disco duro de 1 TB a 5400 RPM.

Tarjeta Gráfica: NVIDIA GeForce GT 650M, 2 GB de Memoria, 384 núcleos CUDA, Capacidad computacional 3.0.

Sistema Operativo: Microsoft Windows 8 Single Language de 64 bits.

Entorno de Desarrollo: Microsoft Visual Studio 2010 Profesional.

Herramientas de Compilación: NVIDIA CUDA SDK Versión 5.5.

Controladores De Video: NVIDIA Drivers Versión 337.88.

8.2 Instancias de Prueba para los Experimentos.

Para las pruebas realizadas del método propuesto para solucionar problemas de ruteo de vehículos con capacidad limitada, se utilizaron benchmarks o instancias de prueba con el fin de poder verificar la calidad de los resultados obtenidos. Las instancias fueron obtenidas de 2 sitios de internet el primero se llama NEO Networking and Emerging Optimization (Dorronsoro, 2013) y el segundo sitio es de la Universitat Heildelberg (Reinelt) En estos sitios también se pueden encontrar instancias para otros problemas de optimización como son el problema del agente viajero o el problema de la mochila.

Las instancias utilizadas han sido publicadas en trabajos de los autores Augerat (Augerat, Belenguer, Benavent, Corberan, Naddef, & Rinaldi, 1998), Fisher (Fisher & Jaikumar R., 1981), Christofides y Eilon. Se decidió por utilizar algunas instancias de estos autores porque también se contaban con los resultados publicados, a fin de sustentar y comparar los resultados obtenidos con el método propuesto aquí. Estas instancias cuentan con dos archivos, un archivo con extensión .vrp que contiene la información del problema cómo: nombre de la instancia, autores, número máximo de vehículos que se pueden utilizar, la solución óptima de la distancia recorrida, el número de clientes, la capacidad de los vehículos, una sección con las coordenadas del depósito y los clientes, una sección con las demandas de los clientes, y una sección para indicar que punto será el depósito y su ubicación.

Ejemplo de la Estructura del Archivo .vrp.

```
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 41
9 57 58
10 62 42
11 42 57
12 27 68
13 43 67
14 58 48
15 58 27
16 37 69
DEMAND_SECTION
1 0
2 19
3 30
4 16
5 23
6 11
7 31
8 15
9 28
10 8
11 8
12 7
13 14
14 6
15 19
16 11
DEPOT_SECTION
1
-1
EOF
```

El segundo archivo tiene la extensión .opt y contiene la solución óptima del problema, en el caso del problema de ruteo de vehículos enlista las rutas de vehículos y los clientes que pertenecen a cada ruta, además contiene el valor óptimo de la distancia recorrida por todas las rutas.

Ejemplo de la Estructura del Archivo .opt.

```
Route #1: 2
Route #2: 6
Route #3: 8
Route #4: 15 12 10
Route #5: 14 5
Route #6: 13 9 7
Route #7: 11 4
Route #8: 3 1
cost 450
```

Las instancias de prueba utilizadas en los experimentos se enlistan en la **Tabla 8.1.**

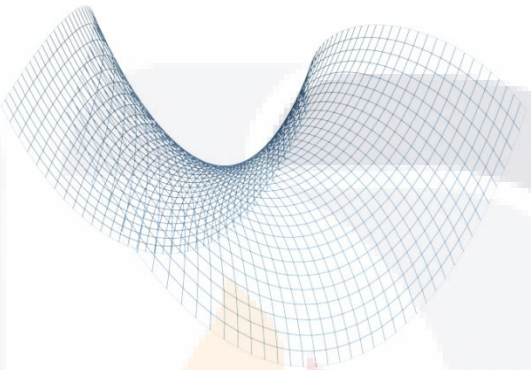
Se eligieron instancias con diferentes cantidades de clientes (de 15 a 100), diferentes limites en el número máximo de vehículos permitidos (de 2 a 10), y con diferentes capacidades para los vehículos (de 35 a 30000).

Tabla 8.1 Nombres y Propiedades de las Instancias para las Pruebas.

Nombre Instancia	Numero Clientes	Solucion Optima	Num Max de Vehiculos	Capacidad del Vehiculo
P-n16-k8.vrp	15	450	8	35
P-n19-k2.vrp	18	212	2	160
E-n23-k3.vrp	22	569	3	4500
A-n33-k5.vrp	32	661	5	100
F-n45-k4.vrp	44	724	4	2010
A-n54-k7.vrp	53	1167	7	100
A-n60-k9.vrp	59	1354	9	100
F-n72-k4.vrp	71	237	4	30000
E-n76-k10.vrp	75	830	10	140
M-n101-k10	100	820	10	200

En el siguiente capítulo se explican los pasos y los parámetros bajo los cuales se realizaron las pruebas con el algoritmo implementado, así como la realización de pruebas estadísticas de los resultados obtenidos.

9.- Pruebas y Análisis de los Resultados



Como se explicó anteriormente la estructura principal del método propuesto para resolver el problema del ruteo de vehículos con capacidad limitada está basada principalmente en un algoritmo de colonia de hormigas, este algoritmo maneja 3 parámetros alfa, beta y rho además de una cantidad de iteraciones durante las cuales el algoritmo busca los mejores recorridos. En la **Tabla 9.1** se muestran los valores de los parámetros utilizados en el algoritmo.

Tabla 9.1 Parámetros Considerados para el Algoritmo de Colonia de Hormigas.

alfa	beta	rho	Número de Iteraciones
1	1	0.7	100
3	3	0.9	
5	5		

Como los parámetros manejan diversos valores y son distintas instancias del problema CVRP, se realizó un experimento factorial, para esta prueba se consideraron todos los posibles niveles para alfa, beta y rho los cuales se muestran en la **Tabla 9.2.**

Tabla 9.2 Niveles Utilizados En las Pruebas de Diseño Factorial.

	Parámetros		
	alfa	beta	rho
nivel 1	1	1	0.7
nivel 2	1	1	0.9
nivel 3	1	3	0.7
nivel 4	1	3	0.9
nivel 5	1	5	0.7
nivel 6	1	5	0.9
nivel 7	3	1	0.7
nivel 8	3	1	0.9
nivel 9	3	3	0.7
nivel 10	3	3	0.9
nivel 11	3	5	0.7
nivel 12	3	5	0.9
nivel 13	5	1	0.7
nivel 14	5	1	0.9
nivel 15	5	3	0.7
nivel 16	5	3	0.9
nivel 17	5	5	0.7
nivel 18	5	5	0.9

Se realizaron 10 réplicas (ejecuciones) para cada uno de los niveles de la tabla anterior en cada una de las 10 instancias listadas en la **Tabla 8.1**, los resultados de las réplicas del algoritmo se encuentran en el **Anexo 3.**

Realizando un análisis empírico de los resultados obtenidos de las réplicas, se detectó que los parámetros del nivel 4 con alfa = 1, beta = 3 y rho = 0.9 generaron los mejores resultados en todas las instancias. Para ilustrar este análisis la siguiente tabla contiene las réplicas para cada instancia con los promedios más bajos.

Tabla 9.3 Réplicas de cada Instancia que Generaron los Promedios más Bajos.

	1	2	3	4	5	6	7	8	9	10	alfa	beta	rho	Promedio
P-n16-k8.vrp	450	450	450	450	450	452	450	450	450	450	1	1	0.7	450.200
P-n19-k2.vrp	212	212	212	212	212	212	212	212	218	212	1	3	0.9	212.600
E-n23-k3.vrp	571	571	576	576	576	576	574	576	576	576	1	3	0.9	574.800
A-n33-k5.vrp	674	677	677	671	671	675	675	671	677	673	1	3	0.7	674.100
F-n45-k4.vrp	766	754	743	734	769	767	769	756	758	759	1	3	0.9	757.500
A-n54-k7.vrp	1255	1252	1259	1264	1264	1256	1264	1261	1260	1261	1	5	0.9	1259.600
A-n60-k9.vrp	1472	1461	1427	1460	1459	1437	1463	1452	1461	1474	1	5	0.7	1456.600
F-n72-k4.vrp	272	272	271	275	269	270	271	271	268	270	1	3	0.7	270.900
E-n76-k10.vrp	913	914	910	908	915	907	911	918	905	916	1	5	0.7	911.700
M-n101-k10.vrp	882	903	884	898	896	894	876	900	879	887	1	5	0.9	889.900

Para respaldar este análisis empírico y la detección de un nivel de parámetros que generará los mejores resultados, se optó por realizar algunas pruebas estadísticas paramétricas con la finalidad de detectar si verdaderamente existe dentro de las muestras una relación entre los parámetros y los resultados obtenidos. Para poder realizar estas pruebas paramétricas son necesarias las siguientes condiciones:

1. **Independencia de las muestras.**

   Las muestras experimentales han de ser independientes. Se consigue esto si los sujetos son obtenidos aleatoriamente. Es decir, se consigue esta condición si los elementos de los diversos grupos han sido elegidos por muestreo aleatorio.

2. **Normalidad.**

   Se supone que los errores experimentales se distribuyen normalmente. Lo que supone que cada una de las puntuaciones se distribuirá normalmente. Para comprobarlo se puede aplicar una prueba de ajuste a la distribución normal como la de Kolmogorov - Smirnov.

3. **Homogeneidad de Varianzas (Homocedasticidad).**

   Esta prueba consiste en comprobar si dos características cualitativas están relacionadas entre sí. Este tipo de contrastes se aplica cuando se desea comparar una variable en situaciones o poblaciones diferentes.

En nuestro caso las muestras o resultados obtenidos en las réplicas si cumplen con la primera condición referente a la independencia de las muestras para realizar las pruebas estadísticas. Las siguientes dos condiciones para poder realizar pruebas estadísticas sobre las muestras son la distribución normal y la homogeneidad de las varianzas, para verificar el cumplimiento de ellas se utilizó el software estadístico SPSS de IBM el cual cuenta con diferentes pruebas estadísticas para evaluar la normalidad y homogeneidad.

Para revisar los contrastes de normalidad en los resultados de las réplicas la prueba más habitual es Kolmogorov – Smirnov. En esta prueba, si la significancia es (p-valor) > 0.05 la distribución es normal. Si la significancia es (p-valor) < 0.05 la distribución no es normal. Los resultados de la prueba Kolmogorov – Smirnov se realizaron para cada una de las 10 instancias y se pueden observar en el **Anexo 4.** Analizando los resultados de la prueba se puede ver que la significancia P < 0.05 por lo tanto no se tiene una distribución normal.

Para revisar la homogeneidad sobre los resultados de las réplicas la prueba utilizada para este propósito es Levene que se basa en comparar las medias de las poblaciones. De igual manera que el test Kolmogorov – Smirnov, si la significancia es (p-valor) > 0.05 las varianzas de los grupos son homogéneas. Si la significancia es (p-valor) < 0.05 las varianzas de los grupos son diferentes. Los resultados de la prueba de Levene se realizaron para cada una de las 10 instancias y se pueden observar en el **Anexo 4** analizando los resultados de la prueba se puede ver que la significancia P < 0.05 por lo tanto no se encontró homogeneidad en las varianzas.

Como solamente se cumplió la primera de las tres condiciones necesarias para realizar pruebas estadísticas paramétricas, tenemos la prueba no paramétrica de Kruskall - Wallis para determinar si los resultados de las réplicas están influidos por un parámetro o nivel de parámetros específicos en el algoritmo utilizado. Nuevamente para realizar la prueba de Kruskall – Wallis se utilizó el programa de pruebas estadísticas SPPS, y se realizó la prueba para las réplicas de cada instancia, como esta prueba funciona en base a categorías de las muestras se agregó una columna llamada categoría que representa a cada uno de los

niveles de parámetros en el algoritmo, por lo tanto se tienen 18 categorías. Los resultados de las pruebas Kruskall – Wallis obtenidas se pueden consultar en el **Anexo 4**. Como resultado del análisis de las pruebas tenemos que efectivamente todas las instancias provienen de niveles diferentes, esto significa que al menos una de las medianas es diferente a las del resto.

Analizando nuevamente los resultados de las réplicas, buscamos los niveles que dieron los mejores resultados para cada instancia, el resultado es la siguiente tabla.

Tabla 9.4 Mejores Promedios de las Réplicas en cada una de las Instancias.

	1	2	3	4	5	6	7	8	9	10	alfa	beta	rho	Mejor Promedio Encontrado
P-n16-k8.vrp	450	450	450	450	450	452	450	450	450	450	**1**	**1**	**0.7**	450.200
P-n19-k2.vrp	212	212	212	212	212	212	212	212	218	212	**1**	**3**	**0.9**	212.600
E-n23-k3.vrp	571	571	576	576	576	576	574	576	576	576	**1**	**3**	**0.9**	574.800
A-n33-k5.vrp	674	677	677	671	671	675	675	671	677	673	**1**	**3**	**0.7**	674.100
F-n45-k4.vrp	766	754	743	734	769	767	769	756	758	759	**1**	**3**	**0.9**	757.500
A-n54-k7.vrp	1255	1252	1259	1264	1264	1256	1264	1261	1260	1261	**1**	**5**	**0.9**	1259.600
A-n60-k9.vrp	1472	1461	1427	1460	1459	1437	1463	1452	1461	1474	**1**	**5**	**0.7**	1456.600
F-n72-k4.vrp	272	272	271	275	269	270	271	271	268	270	**1**	**3**	**0.7**	270.900
E-n76-k10.vrp	913	914	910	908	915	907	911	918	905	916	**1**	**5**	**0.7**	911.700
M-n101-k10.vrp	882	903	884	898	896	894	876	900	879	887	**1**	**5**	**0.9**	889.900

En ella se puede observar que el parámetro alfa = 1 es el único que aparece de manera constante en los mejores resultados y promedios de cada instancia. Y de igual manera se puede observar que el nivel 4 con los parámetros alfa = 1, beta = 3 y rho = 0.9 efectivamente es el nivel que aparece más veces en los mejores promedios.

Conclusiones

De manera general se puede decir que el método de solución ha sido satisfactorio, esto se debe a los resultados positivos obtenidos. Se logró la implementación en paralelo de las funciones para realizar el cálculo de las distancias euclidianas entre todos los clientes, para calcular las distancias entre los clientes y el depósito, relacionadas al manejo del valor de feromonas fueron la inicialización, la evaporación y la actualización.

En las pruebas de diseño factorial realizadas para determinar los parámetros que dieron los mejores resultados se obtuvieron para alfa = 1, beta = 3 y rho = 0.9, además las ejecuciones de las pruebas se realizaron con 500 iteraciones, si se aumentaba el número de iteraciones el algoritmo tardaba más tiempo y los resultados no mostraban gran mejoría.

En los resultados podemos ver que en dos de las diez instancias de pruebas en las que se trabajó se logró encontrar la solución óptima, se calculó un porcentaje de la diferencia entre la solución óptima de cada instancia y el mejor resultado obtenido por el algoritmo de solución propuesto, siendo la máxima diferencia de un 10.54%. Además el promedio de los porcentajes de diferencia entre soluciones óptimas y soluciones obtenidas fue de un 4.16%.

Desde las etapas de programación del algoritmo, la selección de los valores para los diferentes parámetros y la realización de las últimas pruebas con las instancias se tuvo que decidir entre 2 opciones involucrando los aspectos de calidad del resultado y tiempo, la primera opción era aumentar el número de iteraciones en el algoritmo para obtener resultados de mejor calidad, esta sería la opción más viable de no ser que el aumento en el número de iteraciones provoca aumentos considerables en la duración de cada ejecución y las mejoras en la calidad de las soluciones no es muy grande. La segunda opción fue por la que se optó en este proyecto, la de utilizar un número de iteraciones relativamente pequeño para tener tiempos de ejecución cortos pero que a su vez no disminuyera mucho la calidad de las soluciones respecto a las soluciones óptimas, se optó por esta forma de trabajo pensando que en la mayoría de situaciones y problemas reales en donde es posible la aplicación del problema de ruteo de vehículos es más importante obtener un resultado de buena calidad aun sin llegar al óptimo en poco tiempo.

Uno de los objetivos de este trabajo era la adaptación en paralelo del algoritmo de colonia de hormigas, de este objetivo se logró adaptar en la arquitectura en paralelo CUDA las funciones más importantes y que involucran la mayor carga de trabajo del mismo algoritmo, estas funciones fueron las del cálculo de las distancias euclidianas, la inicialización de feromonas, el cálculo de distancias entre clientes y el depósito, así como la evaporación y actualización de los valores de feromonas. Estas funciones fueron posibles adaptarse para su funcionamiento en paralelo ya que como se estudió en la bibliografía se tratan de procesos y ciclos que trabajan de manera independiente y que además manejan estructuras cíclicas y de datos como son los vectores, que se adecuan perfectamente para su tratamiento en paralelo mediante una tarjeta gráfica.

Además de las funciones implementadas en paralelo del algoritmo de colonia de hormigas, se intentó paralelizar la función de creación de recorridos y la asignación de los clientes a los recorridos, por una parte se consideró su implementación en paralelo ya que la información estaba almacenada en vectores y su funcionamiento era a base de ciclos For, pero los motivos por los que ésta función no era conveniente paralelizar eran que manejaba

diversas variables para controlar la capacidad disponible del vehículo, la lista y el orden de los clientes asignados a un recorrido, el número del vehículo de la ruta actual, para almacenar las distancias recorridas por cada ruta y de manera colectiva. Se analizaron todos estos puntos a favor y en contra, inclusive se comenzaron a realizar algunas pruebas de paralelización, pero se observó un incremento considerable de los periodos de ejecución del algoritmo, debido al tiempo extra que se necesita para enviar y recibir la información entre la memoria principal y la memoria de la tarjeta gráfica.

Una opción que se sugiere al implementar el método de rutear primero – asignar después, es la de generar soluciones o recorridos de los clientes de manera aleatoria y después aplicar un método de optimización local como son el 2-opt o el 3-opt, pero si se hubiera optado por aplicar uno de estos métodos se estaría perdiendo la ventaja de tiempo ganada con el uso de una metaheurística como es el algoritmo de colonia de hormigas, que va construyendo las rutas de clientes y al mismo tiempo va buscando las distancias más cortas entre ellos.

Debido a la naturaleza de los problemas de optimización combinatoria, en los cuales el espacio de búsqueda de una solución es muy extenso, esto implica que el encontrar la solución óptima tarde de unos cuantos minutos a horas o incluso días. Es por esta razón que en los resultados del método propuesto, conforme el tamaño de puntos o clientes de las instancias de prueba van incrementándose; de igual manera el tiempo para encontrar su solución aumenta de manera exponencial y es más complicado encontrar su solución óptima. Otro problema con el que se encontró al momento de buscar las instancias para realizar las pruebas fue la dificultad de encontrar instancias de prueba con números de clientes superiores y con una solución óptima cuyos resultados o aplicación pudieran ser consultados en la literatura o publicaciones serias.

Respecto al punto de comparar nuestro método de solución frente a otros métodos en cuanto a tiempos de resolución, no fue viable debido a que no era posible conseguir o duplicar el software y hardware empleado en las pruebas de las publicaciones de otros

autores. Otra razón de que no era posible la comparación era debido a la propuesta de implementar un método cuyas funciones trabajaran de manera paralela utilizando los núcleos de procesamiento y memoria de una tarjeta gráfica, cosa que en las publicaciones de resultados y pruebas de las instancias de pruebas no era posible debido a limitaciones en su tiempo de hardware y software.

Relacionado al empleo de la tecnología CUDA para implementar los algoritmos en paralelo es importante hacer dos observaciones, en primer lugar al principio del desarrollo del proyecto y debido al desconocimiento del funcionamiento de la tecnología CUDA se pretendía la implementación del algoritmo completo en paralelo, pero conforme se fue profundizando en esta nueva arquitectura se detectó que las partes con más posibilidades de implementarse en paralelo son estructuras cíclicas como el FOR y WHILE, así como funciones que manejan estructuras de datos y arreglos de 1, 2 y 3 dimensiones, por otra parte es importante identificar que estos ciclos y estructuras se manejen de manera independiente a funciones o instrucciones antecesoras para asegurar que los resultados obtenidos en el núcleo en paralelo puedan ser utilizados por las funciones e instrucciones sucesoras del algoritmo.

La segunda observación respecto a la arquitectura CUDA, es que aun cuando se detecta un ciclo o estructura de datos que sea posible paralelizar no es conveniente en todos los casos hacerlo, la experiencia fue en una función para rotar los valores de un arreglo unidimensional, se programaron 2 algoritmos uno con la función secuencial y otro con la misma función en paralelo, se esperaba una disminución de tiempo para el segundo algoritmo pero el resultado fue que se volvió más lento a pesar de que la función se realizaba en paralelo en la tarjeta gráfica, la razón por la que este algoritmo se volvió más lento fue debido a que el copiado de información entre la memoria principal y la memoria de la tarjeta gráfica involucra una cantidad de tiempo extra, añadiendo que esta función se ejecuta varias veces en cada iteración del algoritmo, por este motivo es necesario analizar las funciones que se implementen en paralelo y verificar si es conveniente o no hacerlo,

tomando en cuenta los cuellos de botella y el tiempo de enviar y recibir la información entre la memoria principal y la memoria de tarjeta gráfica.

Referente a la calidad de los resultados obtenidos con el método desarrollado y los parámetros utilizados en el algoritmo, mediante la realización de pruebas estadísticas se puede observar que el parámetro alfa = 1 es el único que aparece de manera constante en los mejores resultados y promedios de cada instancia. Y de igual manera se puede observar que el nivel 4 con los parámetros alfa = 1, beta = 3 y rho = 0.9 efectivamente es el nivel que aparece más veces en los mejores promedios. Así también analizando los resultados del experimento de diseño factorial se observó, que a medida que el número de clientes iba incrementando las soluciones obtenidas estaban más alejadas de la solución óptima, esto da pauta a realizar posteriormente nuevas pruebas con diferentes niveles de parámetros para cada instancia, con el fin de intentar encontrar mejores resultados para las instancias mayores.

Bibliografía

BrookGPU: Introduction. (31 de Julio de 2014). Recuperado el 14 de Julio de 2014, de http://graphics.stanford.edu/projects/brookgpu/intro.html

Historia de Nvidia: de la tarjeta gráfica al procesador móvil. (31 de Julio de 2014). Recuperado el 31 de Julio de 2014, de http://www.nvidia.es/object/corporate-timeline-es.html

Adam, E. E., & Ebert, R. J. (1978.). *Production and operations management.* Englewood Cliffs, NJ.: Prentice - Hall.

Alba, E. (2005). *Parallel Metaheuristics: A New Class of Algorithms.* Wiley.

Augerat, P., Belenguer, P., Benavent, E., Corberan, A., Naddef, D., & Rinaldi, E. (1998). Computacional Results with a Branch-And-Cut Code For the Capacitated Vehicle Routing Problem. *Research Report 949-M Universite Joseph Fourier. Grenoble, France.*

Bäck, T., Fogel, D., & Michalewicz, Z. (1997.). *Handbook of Evolutionary Computation.* New York.: Institute of Physics Publishing, Bristol and Oxford University Press.

Beasley, J. E. (1983.). Route First - Cluster Second Methods for the Vehicle Routing Problem. *Omega 11.*, págs. 403-408.

Bianco, L., Mingozzi, A., & Ricciardell, S. (1994.). Exact and Heuristic Procedures for The Travelling Salesman Problem with Precedence Constraints, Based on Dynamic Programming. *Infor.*, Vol. 32, pp 19-32.

Brownlee, J. (2011.). *Clever Algorithms: Nature-Inspired Programming Recipes.* Lulu Enterprises, January 2011.

Bullnheimer, B., Hartl, R. F., & Strauss, C. (1997.). Applying the Ant System to the Vehicle Routing Problem. *Paper presented at 2nd International Conference on Metaheuristics MIC'97.*

Bullnheimer, B., Hartl, R. F., & Strauss, C. (1999.). An Improved Ant System Algorithm for the Vehicle Routing Problem. *Annals of Operations Research. 89.*, 319-328.

Caric, T., & Gold, H. (2008.). *Vehicle Routing Problem.* InTech.

Chalasani, P., & Motwani, R. (1999.). Approximating Capacitated Routing and Delivery Problems. *SIAM Journal on Computing.*, Vol. 28. Pag 2133-2149.

Clarke, G., & Wright, J. (1964.). Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research.*, Vol 12., pp 568-581.

Cook, S. A. (1971.). The Complexity of Theorem-Prooving Procedures. *Proceedings of the Third Annual ACM Symposium on the Theory of Computing.*

Dantzig, G. B. (1959). The Truck Dispatching Problem. *Managment Science*, 6, 80-91.

Dantzig, G. B. (1959). The Truck Dispatching Problem. *Management Science*, 6, 80-91.

Dantzig, G. B., & Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science*, 6, 80-91.

Deneubourg, J. L., Aron, S., & Goss, S. (1990.). The Self-Organizing Exploratory Pattern of the Argentine Ant. *Journal of Insect Behavior.*, 159-169.

Denning, P. J. (2005.). Is Computer Science Science?. *Communications of the ACM.*, 48, 27-31.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 269-271.

Doerner, K., Gronalt, M., Hartl, R., Reimann, M., Strauss, C., & Stummer, M. (2002.). SavingAnts for the Vehicle Routing Problem. *Series Lecture Notes in Computer Science. Volume 1.*, 11-20.

Doerner, K., Hartl, R. F., & Reimann, M. (2011.). A hybrid ACO algorithm for the Full Truckload Transportation Problem. *SFB Adaptive Information Systems and Modelling in Economics and Management Science, WU Vienna University of Economics and Business.*

Dorigo, M., & Gambardella, L. M. (1997.). Ant Colonies for the Travelling Salesman Problem. *Biosystems.*, 39.

Dorigo, M., & Stützle, T. (2004.). *Ant Colony Optimization.* MIT Press.

Dorronsoro, B. (7 de Enero de 2013). *The VRP WEB.* Recuperado el 31 de Octubre de 2013, de Collaboration between AUREN and the Languages and Computation Sciences department of the University of Málaga.: http://neo.lcc.uma.es/vrp/

Feo, T., & Resende, M. G. (1989.). A Probabilistic Heuristic for a Computationally Dificult Covering Problem. *Operations Research Letters.*(8), págs. 67-71.

Fisher, M. L., & Jaikumar R. (1981). A Generalized Assignment Heuristic for Vehicle Routing. *Networks*, 109-124.

Floyd, R. W. (1962.). Algorithm 97: Shortest Path. *Communications of the ACM.*, 345.

Gambardella, L., Taillard, É., & Agazzi, G. (1999.). MACS-VRPTW: A multiple colony system for vehicle routing problems with time windows. *New Ideas in Optimization.*

Garey, M., & Johnson, D. (1979.). *Computers and Intractability - A guide to the Theory of NP-Completeness.* San Fransisco: Freeman.

Glover, F. (1977.). Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences., 8,* 156-166.

Glover, F. (1989.). Tabu Search, Part 1. *ORSA Journal on Computing., 1*(3), págs. pp. 190-206.

Glover, F. (1990.). Tabu Search, Part 2. *ORSA Journal on Computing., 2*(1), págs. pp. 4-32.

Glover, F., & Kochenberger, G. (2003.). *Handbook of Metaheuristics. (Hardcover).* Springer.

GPU-Accelerated Applications. (s.f.). Recuperado el 2 de Octubre de 2013, de http://www.nvidia.es/docs/IO/123576/nv-applications-catalog-lowres.pdf

Ho, S., & Haugland, D. (2004.). A Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows and Split Deliveries. *Computers & Operations Research.*, Volume 31, Pages 1947-1964.

Holland, J. H. (1992.). *Adaptation in Natural and Artificial Systems.* (M. press., Ed.) Ann Arbor, Michigan.: University of Michigan Press.

Hoos, H. H., & Stützle, T. (2004.). *Stochastic Local Search: Foundations & Applications.* Elsevier / Morgan Kaufmann.

Jones, M. T. (2010.). *Artificial Intelligence: A Systems Approach.* Jones & Bartlett Learning.

Karp, R. M. (1972). Reducibility Among Combinatorial Problems. *In R. E. Miller & J. W. Thatcher (ed.), Complexity of Computer Computations.*, 85-103.

Karp, R. M. (1986.). Combinatorics, Complexity, and Randomness. *Communications of the ACM.*, 29, 97-109.

Kennedy, J., & Eberhart, R. (1995.). Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks., IV*, págs. 1942-1948.

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

Laporte, G. (1992.). An Overview of Exact and Approximate Algorithms. *European Journal of Operational Research. 59.*, 231-247.

Larrañaga, P., & Lozano, J. A. (2002.). Estimation of distribution Algorithms. *A New Tool for Evolutionary Computation.*

Mañas, J. A. (Noviembre de 1997). *Analisis de Algoritmos: Complejidad.* Recuperado el 9 de Octubre. de 2013., de Analisis de Algoritmos: Complejidad.: http://www.lab.dit.upm.es/~lprg/material/apuntes/o/#s5

Martí, R. (2003). Procedimientos Metaheurísticos en Optimización Combinatoria. *Matemàtiques*, págs. 3-62.

Martínez, C., & Mota, E. (2000.). Del Poliedro del Agente Viajero Gráfico al de Rutas de Vehículos con Demanda Compartida. *Qüestiió.*, Vol. 24, Nº. 3, pag. 495-528.

Mladenovic, M., & Hansen, P. (1997.). Variable Neighborhood Search. *Computers and Operations Research., 24*, págs. 1097-1000.

Paessens, H. (1988.). The Savings Algorithm for the Vehicle Routing Problem. *European Journal of Operational Research. 34.*, 336-344.

Plataforma de Computacion Paralela CUDA. (s.f.). Recuperado el 29 de Agosto de 2012, de http://www.nvidia.com.mx/object/cuda_home_new_la.html

Reinelt, G. (s.f.). *Universität Heidelberg.* Recuperado el 3 de Marzo de 2014, de https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/vrp/

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.*, Vol. 16, No. 2, pág. 35 - 55.

Rosen, K. (2003.). *Discrete Mathematics and its Applications.* McGraw-Hill.

Sanders, J., & Kandrot, E. (2011). *CUDA by Example: An Introduction to General-Purpose GPU Programming.* Addison-Wesley Professional.

Tansini, L., Urquhart, M., & Viera, O. (2001.). *Comparing Assignment algorithms for the Multi-Depot VRP.* Montevideo, Uruguay.: Technical Report, University of Montevideo, Uruquay.

Toth, P., & Vigo, D. (2002.). *The Vehicle Routing Problem.* Philadelphia, USA.: Society for Industrial and Applied Mathematics (SIAM) Monographs on Discrete Mathematics and Applications.

Turing, A. M. (1936.). On Computable Numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society.*, 42, 230-265.

Wilf, H. S. (2002.). *Algorithms and Complexity.* A K Peters/CRC Press; 2 edition.

Anexos

Anexo 1

Trabajos Publicados
Trabajos Aceptados
Conferencias
Cursos o Talleres

Anexo 2

Publicación Relevante

Anexo 3

Resultados de las Pruebas de Diseño Factorial

Anexo 4

Resultados del Test Kolmogorov - Smirnov
Resultados del Test Levene
Resultados del Test Kruskal – Wallis

Anexo 1

Trabajos Publicados

Ponce, J. C., Padilla, A., Recio A., Hernández J. A. (2013). *Optimización de un Algoritmo de Colonia de Hormigas Mediante CUDA para Resolver el Problema del Agente Viajero*. 14° Seminario de Investigación. Universidad Autónoma de Aguascalientes.

Ponce, J. C., Padilla, A., Recio A., Hernández J. A. (2013). *Implementación de un Algoritmo de Hormigas en CUDA para Resolver el Problema del Agente Viajero*. Cuarto Congreso Internacional – La Investigación en el Posgrado. Universidad Autónoma de Aguascalientes.

Ponce, J.C. (2013). *Implementation of a Paralel Ant Colony Algorithm Using CUDA and GPUs to Solve Routings Problems*. The 12th Mexican International Conference on Artificial Intelligence (MICAI). The Mexican Society for Artificial Intelligence and the Autonomous Metropolitan University.

Trabajos Aceptados

Ponce, J. C., Padilla, A., Torres, A., Recio A. (2013). *Algoritmo de Colonia de Hormigas Paralelo para Resolver Problemas de Ruteo de Vehículos*. Quinto Congreso Internacional – La Investigación en el Posgrado. Universidad Autónoma de Aguascalientes.

Ponce, J. C., Recio A., Ochoa A., Hernández, J. A., Ornelas, F., Padilla, A., Torres, A. (2014). *Algoritmo de Colonia de Hormigas en CUDA para la Optimización de Rutas de Vehículos de Distribución*. Komputer Sapiens "Sistemas Híbridos Inteligentes y sus Aplicaciones" Año 6 Vol. 3. Ver **Anexo 2.**

Conferencias

Alfonso Recio Hernández. (2014). *Implementación de un Algoritmo de Colonia de Hormigas en Paralelo Aplicado al Problema del VRP*. Universidad autónoma del Estado de Morelos, Facultad de Contaduría, Administración e Informática. Cuernavaca, Morelos, México.

Cursos o Talleres

Taller "Programación en Paralelo", VIII Congreso de Ciencias Exactas del Departamento de ciencias Básicas, Universidad Autónoma de Aguascalientes del 06 al 10 de Octubre del 2014.

Anexo 2

Publicación Relevante

SOCIEDAD MEXICANA DE INTELIGENCIA ARTIFICIAL
REVISTA KOMPUTER SAPIENS

CARTA DE ACEPTACIÓN

México. D.F, 19 de Julio de 2014

Julio Cesar Ponce

Estimado colega

A nombre de la Sociedad Mexicana de Inteligencia Artificial (SMIA) tengo el placer de informarles que como resultado del proceso de arbitraje, su artículo

Algoritmo de colonia de hormigas en CUDA para la optimización de rutas de distribución

ha sido aceptado por el Comité Editorial para su publicación en la revista de divulgación Komputer Sapiens. El artículo será publicado en el año 6 volumen 3, correspondiente al periodo septiembre-diciembre de 2014, cuya temática es de "Sistemas Híbridos Inteligentes y sus Aplicaciones".

Para la preparación de su manuscrito, les pedimos sean tan amable de atender las instrucciones que hemos enviado por correo electrónico.

Agradecemos el tiempo dedicado a la preparación de su manuscrito y lo invitamos a seguir colaborando con esta revista.

Atentamente

Dra. Laura Cruz Reyes
Editor en Jefe

Dra. Elisa Schaeffer
Editora Científica

Revista Komputer Sapiens ISSN: 2007-0691
Sociedad Méxicana de Inteligencia Artificial, A.C. Nuevo León 125, 303, Col. Hipódromo de la Condesa, C.P. 06140, DF, México.
Tel. +52 (833) 357.48.20 ext. 3024, fax +52 (833) 215.85.44
komputersapiens@smia.org.mx www.komputersapiens.org

Algoritmo de Colonia de Hormigas en CUDA para la Optimización de Rutas de Distribución

Julio Cesar Ponce[1], Alfonso Recio[2], Alberto Ochoa[3], Alberto Hernandez[4], Francisco Ornelas[1], Alejandro Padilla[1], Aurora Torres[1]

[1] Departamento de Ciencias de la Computación de la Universidad Autónoma de Aguascalientes
jcponce@correo.uaa.mx, fjornel@ correo.uaa.mx, apadilla@ correo.uaa.mx,
[2] Maestría en Ciencias con opcion a la Computacion de la Universidad Autónoma de Aguascalientes
li.alfonso.rh@gmail.com
[3] Universidad Autónoma de Ciudad Juarez
alberto.ochoa@uacj.mx
[4]Faculta de Contaduría e Informática de la Universidad Autónoma del Estado de Morelos
jose_hernandez@uaem.mx

Resumen. Este trabajo muestra cómo pueden los algoritmos de colonias de hormigas resolver problemas de optimización de gran complejidad que requieren de grandes capacidades de computo (Espacio-Tiempo), los cuales pueden ser combinados con nuevas arquitecturas de programación en paralelo mediante el uso de tarjetas gráficas (GPU´s) para tratar de mejorar su rendimiento. En las diferentes secciones de este trabajo se explicarán que son los Algoritmos de Colonia de Hormigas, una introducción y las características del ambiente de programación en paralelo conocido como CUDA, como un algoritmo se puede adaptar a la arquitectura de CUDA. La finalidad es presentar cómo un algoritmo de colonia de hormigas en combinación con esta nueva arquitectura se pueden utilizar para resolver el Problemas de Rutas de Distribución en las empresas, para esto se resuelve el problema de Ruteo de Vehículos (VRP), se muestra los resultados obtenidos, y conclusiones sobre las mejoras de rendimiento respecto a un método de programación tradicional mediante la unidad de procesamiento central (CPU).

Palabras Clave. GPU´s, Problemas de Optimización, Algoritmos de Colonia de Hormigas, Problemas de Rutas de Distribución, Programación en Paralelo, CUDA.

1 Introducción

Actualmente podemos encontrar en cualquier lugar problemas de optimización, en la industria se tienen un gran número de procesos que se buscan mejorar y la mayoría de estos se pueden formular como problemas de optimización de recursos (personal, tiempo, espacio, dinero, distancias, etc.).

Muchos de los problemas de optimización son complejos y difíciles de solucionar mediante métodos exactos, en estos casos la mejor opción es utilizar algoritmos de aproximación los cuales intentan encontrar buenas soluciones con un margen de error mínimo.

El problema de ruteo de vehículos o VRP (Vehicule Routing Problem) es uno de los problemas de optimización combinatoria más difíciles de resolver, su definición data de los años 50's en un trabajo realizado por Dantzig y Ramser en donde establecieron la formulación matemática, sobre el problema de la entrega de combustible a las estaciones de servicio (Dantzig & Ramser, 1959).

Las principales aplicaciones del VRP son en logística y distribución en las empresas como: la entrega y recolección de materiales, el transporte de personal, distribución de correspondencia, etc.

Actualmente existen diversos algoritmos para resolver este problema, como son: Búsqueda Tabú (Glover, 1989.), Procedimiento de búsqueda Adaptativa Aleatoria Voraz (GRASP) (Feo & Resende, 1989.), Algoritmos Genéticos (Holland, 1992.). Recocido Simulado (Kirkpatrick, Gelatt, & Vecchi, 1983.),

Optimización por enjambre de Partículas (Kennedy & Eberhart, 1995.), Algoritmo de Estimación de Distribución (Larrañaga & Lozano, 2002.), Búsqueda Dispersa (Glover, 1977), Optimización con Colonia de hormigas (ACO) (Dorigo & Stüzle, Ant colony Optimization., 2004.), entre otras. Este último es un algoritmo bioinspirado en la manera de como las hormigas buscan alimento buscando los caminos más cortos, estas construyen sus rutas al depositar una sustancia química llamada feromona, los mejores caminos van acumulando una mayor concentración debido al paso más frecuente de las hormigas. En este algoritmo cada hormiga es vista como un agente artificial el cual va construyendo su ruta mediante una fórmula probabilística, intentando encontrar una solución óptima en un tiempo polinomial determinado (Dorigo & Stüzle, Ant colony Optimization., 2004.).

La computación de propósito general en las GPUs intenta aprovechar las capacidades de procesamiento actuales. En esta área se tiene la arquitectura de programación llamada CUDA desarrollada por la compañía NVIDIA, esta maneja extensiones en varios lenguajes de programación y permite la programación de funciones o núcleos que se pueden ejecutar de manera simultánea en la GPU obteniendo resultados de manera más rápida que si se hubiera ejecutado solamente en la CPU (Nvidia, 2014).

Los algoritmos ACO tiene diferentes variantes que han sido implementadas en diferentes trabajos de manera secuencial para resolver problemas de ruteo, llevando a cabo su ejecución mediante la CPU (Gambardella, Taillard, & Agazzi, 1999.) (Bullnheimer, Hartl, & Strauss, 1999.) (Dorigo & Gambardella, Ant Colonies for the Travelling Salesman Problem., 1997.). Actualmente existen otras arquitecturas como CUDA que permiten el procesamiento de operaciones en paralelo en la GPU para hacer más veloz su ejecución (Dawson & Stewart, 2013.). Este trabajo presenta un algoritmo de colonia de hormigas para solucionar el VRP con capacidad limitada, el cual está basado en un procedimiento conocido como Rutear Primero – Asignar Después, con funciones modificadas para ejecutarse de manera paralela en la GPU. Así como los resultados obtenidos utilizando benchmarks del estado del arte.

2 Algoritmo de Colonia de Hormigas

Los algoritmos ACO surgen como parte del trabajo de tesis de Marco Dorigo (Dorigo & Stüzle, Ant colony Optimization., 2004.), en el que presenta a las hormigas como insectos sociales que a pesar de tener limitaciones como la visión, son capaces de realizar tareas de manera cooperativa y efectiva, como es la búsqueda de alimento, de este modo al ir explorando caminos entre su hormiguero y la fuente de alimentos marcan la ruta con una sustancia química llamada feromona la cual ayuda a las otras hormigas a identificar los mejores caminos.

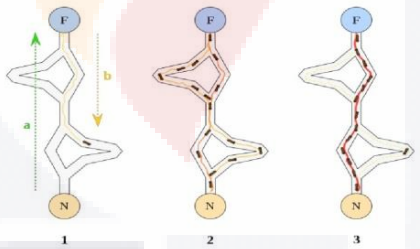


Figura 1. Ejemplo de una Colonia de Hormigas Creando una Ruta de Manera Natural.

En el algoritmo cada hormiga artificial es un agente que intentan imitar el comportamiento real en su tarea de buscar alimento.

La manera en que una hormiga selecciona el siguiente punto de su recorrido se basa en una sencilla ecuación. En la **Fórmula 1** se calcula la probabilidad de que una hormiga *k* seleccione moverse de un punto *i* a un punto *j*.

$$P_{ij}^{k} = \frac{(\tau_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta}}{\sum_{l \in N_i^k} (\tau_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta}} \quad , \quad \text{Si} \quad j \in N_i^k$$

Fórmula 1. Probabilidad de seleccionar el camino entre el punto *i* y el punto *j*.

La cantidad de feromona que existe entre los puntos *i* y j se conoce como rastro de feromona artificial, es el atractivo de visitar un punto *j* a partir del punto *i* y está representado por τ_{ij}. También se maneja un valor heurístico entre la posición actual *i* y la posición candidata *j* que está representado por η_{ij}, su valor es inversamente proporcional a la distancia entre 2 puntos y se calcula con $\eta_{ij} = 1/d_{ij}$. Otros parámetros utilizados son α y β que indican el nivel de importancia, el primero del rastro de feromona y el segundo de la distancia. N_i^k es el conjunto de puntos vecinos aun no visitados por una hormiga *k* que se encuentra en el punto *i*.

Una vez que todas las hormigas terminan de construir sus recorridos, se modifica el valor de la feromona de los recorridos. El primer paso es disminuir la cantidad de feromona en todas las aristas por un factor de evaporación y el siguiente paso es incrementar la feromona en las aristas utilizadas por las hormigas en su recorrido. Hay que resaltar que como se trata de un grupo colaborativo de hormigas artificiales en donde cada una va procesando y almacenando su información de los recorridos, puntos visitados y los no visitados, llevar un control en sus rastros de feromonas, se puede apreciar que el algoritmo ACO tiene características que lo hacen posible modificar y poder ser ejecutado en paralelo.

3 CUDA y la Programación en Paralelo

Actualmente se busca explotar la potencialidad de las GPUs en aplicaciones de uso intensivo de ciencia e ingeniería, la arquitectura CUDA(Arquitectura Unificada de Dispositivos de Computo) (Nvidia), hace referencia al conjunto de herramientas, un compilador y extensiones para lenguajes como: C, C++, Java, Python, Ruby, o Fortran, Que permiten a los programadores, desarrollar funciones que se ejecutan en la GPU aprovechando su alto nivel de paralelismo a comparación lo que hace que un programa en CUDA sea más veloz. La **Figura 2** muestra cómo ha aumentado la cantidad de operaciones con punto flotante tanto de precisión sencilla así como de doble precisión y la diferencia entre CPU´s y GPU´s.

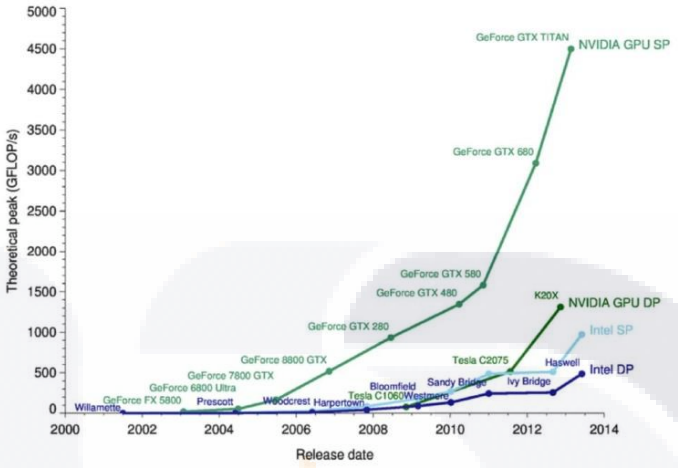


Figura 2. Cantidad de Operaciones de Punto Flotante de CPU´s y GPU.

Este aumento hace posible que una GPU pueda procesar una mayor cantidad de operaciones porque en su interior se pueden encontrar cientos o miles de unidades de procesamiento, menos potentes que las de una CPU pero que permiten realizar cálculos independientes en paralelo. Esto es lo que permite que las GPU´s puedan adaptarse a programas en donde se realicen operaciones de manera independiente, un ejemplo del uso de esta arquitectura sería aplicar un filtro sobre una imagen, una CPU realizaría esta función de manera lineal, en donde se aplicaría el filtro a cada pixel de la imagen de manera secuencial. De otra manera si se realizara mediante la GPU la imagen se puede dividir en partes y estas ser tratadas por las unidades de procesamiento de manera paralela e independiente. Algunos de los beneficios de usar aplicaciones en CUDA es la capacidad de realizar varias tareas en paralelo inclusive no estando relacionadas entre sí y ayudar a descargar a la CPU de trabajo permitiendo que este se encargue de otras tareas para dar mayor fluidez al sistema.

En el área de computación se pueden encontrar problemas con un grado de complejidad muy elevado, que requieren de equipos de cómputo con características muy altas y costosas. Este es el motivo por el cual se optó por implementar un algoritmo en CUDA para intentar solucionar uno de estos problemas complejos.

4 El Problema de Ruteo de Vehículos

Como se mencionó anteriormente el VRP data de los años 50's, la definición del problema consiste en encontrar el conjunto de rutas que minimicen el costo total de transportación para un número m de vehículos con capacidades uniformes que deben satisfacer las demandas de una n cantidad de clientes, iniciando y terminando cada ruta en un punto llamado depósito y respetando la restricción de que las demandas de los clientes asignados a una ruta no sobrepasen la capacidad del vehículo cómo se maneja en la variante del Problema de Ruteo de Vehículos con Capacidad Limitada (CVRP) (ver **Figura 3**), de acuerdo a las restricciones que maneje el problema existen otras variantes del VRP como problemas con ventanas de tiempo (VRPTW), con múltiples depósitos (MDVRP), con entregas y devoluciones (VRPPD), con entregas parciales (SDVRP), con valores al azar (SVRP), con planificación periódica (PVRP), con viajes de regreso (VRPB), con instalaciones satelitales (Toth & Vigo, 2002.).

Una solución es factible, si la suma de la demanda de los clientes asignados a cada ruta no excede la capacidad del vehículo. También existen instancias de prueba (Alba, 2014) (Reinelt, 2014) con la información del problema como el número, demanda y ubicación de clientes, así como la ubicación del depósito, con el objetivo de verificar el funcionamiento del algoritmo o método que se está utilizando.

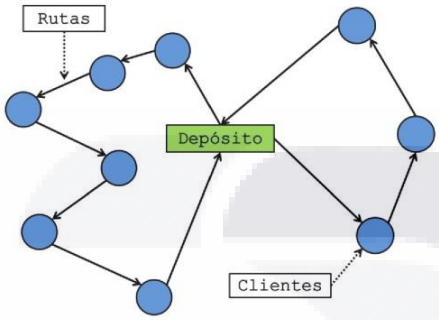


Figura 3. Representación Gráfica de un CVRP.

Los objetivos más comunes que se consideran en problemas de ruteo de vehículos son:

- Minimizar el costo total de transportación.
- Minimizar el número de vehículos requeridos para satisfacer la demanda.
- Manejar un número de clientes por vehículo equilibrado, para efectos de tiempos de los viajes y la carga de vehículos.
- Minimizar las penalizaciones asociadas con entregas parciales a los clientes.

Como parte del algoritmo que se propone en este trabajo se maneja el método rutear primero – asignar después utilizando un algoritmo ACO.

El método Rutear primero - Asignar después propuesto por Beasley con el objetivo de resolver el VRP en su trabajo (Beasley, 1983.), consiste en los siguientes pasos:

- Generar una ruta con la menor distancia.
- Avanzar en una dirección asignada por los clientes de la ruta generada partiendo del depósito, los clientes son asignados a un vehículo esto es, tomando en cuenta la capacidad del vehículo se verifica si la demanda del siguiente cliente a visitar se puede satisfacer por el vehículo, si esto es posible el cliente se asigna a ese vehículo y se avanza al siguiente cliente, en caso contrario si la capacidad del vehículo es insuficiente para satisfacer la demanda del siguiente cliente el vehículo regresa al depósito y el cliente se asigna al vehículo de la siguiente ruta
- El proceso de verificar si el vehículo puede o no satisfacer las demandas de los clientes del problema se repite hasta llegar al cliente final.

5 Implementación del Algoritmo de Colonia de Hormigas en CUDA

En este algoritmo se puede resaltar 3 características principales,

1) Utilizar el algoritmo ACO para encontrar la ruta con la distancia más corta entre los clientes.
2) La implementación en paralelo de algunas funciones del algoritmo ACO como son el cálculo de las distancias entre los clientes, el cálculo de las distancias entre depósito y clientes, la

inicialización de valores de feromonas, la evaporación de feromonas e intensificación del rastro de feromonas, Por último.

3) Se incluyó dentro del algoritmo una función de asignación de clientes a las rutas empleando el método Rutear Primero – Asignar Después.

La estructura general para el algoritmo implementado para resolver problemas de vehículos con capacidad limitada es la siguiente (Ponce et al., 2013):

Leer Archivo Instancia
Generar Ruta de Menor Distancia mediante Algoritmo de Colonia de Hormigas
Algoritmo de Hormigas
Calcular Matriz_distancias() **CUDA**
Inicializar Matriz_feromonas() **CUDA**
Calcular Distancia_Deposito_Clientes() **CUDA**
Mientras (iteraciones del algoritmo de colonia de hormigas)
Inicializar_Hormigas( )
Construir_Recorridos ( )
Evaporar_Feromonas ( ) **CUDA**
Actualizar_Feromonas ( ) **CUDA**
Asignar_Vehiculos_Rutas ( )
Fin Mientras
Impresión de la Mejor Asignación de Clientes a Rutas

El algoritmo propuesto, realiza en primer lugar la lectura de una instancia que contiene información sobre el problema (número de ciudades, capacidad del vehículo, número máximo de vehículos, posición de clientes).

La siguiente etapa del algoritmo es la generación de rutas con la menor distancia con el algoritmo base de colonia de hormigas.

Dentro de la primera etapa del algoritmo hay 3 núcleos que se ejecutan dentro de la GPU de manera paralela, la primera función es la que realiza el cálculo de las distancias euclidianas entre todos los clientes.

Pseudocódigo 1. Núcleo Para Calcular la Distancia Euclidiana Entre los Clientes.

```
__global__ void kernelPrecomputedistances(float *cities_x, float
*cities_y, double *precomputed_distance, int n)
{

      double dx, dy;
      int i = blockDim.x * blockIdx.x + threadIdx.x;
      int j = blockDim.y * blockIdx.y + threadIdx.y;
      int index= i * n + j;

      dx = abs(cities_x[i] - cities_x[j]);
      dy = abs(cities_y[i] - cities_y[j]);
      precomputed_distance[index]= sqrt((dx*dx)+(dy*dy));
}
```

En seguida encontramos la función también paralela que inicializa los valores de feromona entre 2 ciudades.

La última función en esta etapa del algoritmo es el cálculo de distancias entre cada cliente y el depósito está información es necesaria en el momento de asignar los clientes a cada vehículo en el caso de que se tratara del inicio o fin de una ruta se agrega a la distancia recorrida por el vehículo la distancia del depósito al primer cliente o del último cliente visitado hacia el depósito.

La primer función es la inicialización de las hormigas artificiales en este punto se colocan en diferentes ciudades como puntos de inicio. La segunda función es la construcción donde cada hormiga va construyendo

su recorrido mediante la fórmula de probabilidad tomando en cuenta la feromona y la distancia con esto va decidiendo cuál será su próxima ciudad a visitar.

Una vez que los recorridos de las hormigas recorren todos los clientes, se ejecutan 2 tareas fundamentales la evaporación y la actualización de feromonas.

En el momento en que todas las hormigas forman un recorrido que atraviesa todos los clientes, se ejecuta la función para crear las rutas, asignando los clientes a los vehículos mientras estos puedan satisfacer la demanda de los clientes siguiente.

6 Resultados

Para las pruebas del algoritmo propuesto para resolver el CVRP, se emplearon instancias de dos sitios web NEO y TSPLI (Alba, 2014) (Reinelt, 2014), en estos sitios se encuentran instancias que son archivos con la información de los problemas y la solución óptima.

Los valores utilizados en las pruebas son alfa=1.0, beta=1.0, para rho = 0.6, el número de hormigas = número de clientes de la instancia, y el número de iteraciones = 500.

En la **Tabla 1** se muestran las diferentes instancias utilizadas en el experimento las cuales manejan de 15 a 100 clientes y desde los 2 hasta los 14 vehículos permitidos por problema, también se tiene la solución óptima que es la suma de las distancias recorridas por todos los vehículos del problema, la siguiente columna con los mejores resultados obtenidos con el algoritmo del método propuesto, la diferencia entre la mejor solución encontrada por el algoritmo contra la solución óptima, y por último se obtuvo el porcentaje de diferencia existente entre la solución óptima y los mejores resultados obtenidos. Los colores indican, verde una diferencia del 0%, amarillo una diferencia entre el 0% y 5%, naranja para una diferencia entre el 5% y 10% y con rojo la diferencia mayor al 10%.

Tabla 1. Resultados de las pruebas realizadas con el algoritmo.

Nombre Instancia	Solucion Optima	Mejor	Diferencia Mejor y Optimo	% Diferencia
P-n16-k8.vrp	450	450	0	0
P-n19-k2.vrp	212	212	0	0
E-n23-k3.vrp	569	569	0	0
A-n33-k5.vrp	661	671	10	1.512859304
F-n45-k4.vrp	724	730	6	0.828729282
A-n54-k7.vrp	1167	1240	73	6.255355613
A-n60-k9.vrp	1354	1420	66	4.874446086
F-n72-k4.vrp	237	255	18	7.594936709
E-n76-k10.vrp	830	901	71	8.554216867
E-n101-k14.vrp	1071	1206	135	12.60504202
				0%
				Mayor a 0 % y Menor a 5 %
				Mayor a 5 % y Menor a 10 %
				Mayor a 10 %

7 Conclusiones y Trabajos Futuros

Se mostro un algoritmo de colonia de hormigas para resolver problemas de ruteo de vehículos con capacidad limitada, Se realizaron pruebas utilizando instancias, se han reducido tiempos de procesamiento gracias a la implementación de algunas de las funciones en paralelo, como se mencionó con anterioridad el

algoritmo tiene características que lo hacen un perfecto candidato para convertir algunas de sus funciones en paralelo con CUDA.

El algoritmo implementado encuentra fácilmente las soluciones óptimas para instancias pequeñas, pero conforme las instancias van aumentando la diferencia entre la solución óptima y la solución obtenida por el método también crecen.

Como trabajo futuro se ha pensado en algunas alternativas como, incrementar el número de iteraciones con el objetivo de que el tiempo de búsqueda de soluciones sea mayor, realizar cambios en los diferentes parámetros del algoritmo de colonia de hormigas utilizando experimentos factoriales para intentar encontrar una combinación de parámetros que de mejores resultados con instancias más grandes, aplicar algún algoritmo de búsqueda local en la mejor solución obtenida aunque esta alternativa resulte en tiempos de ejecución prolongados y se pierda el principal objetivo de encontrar una buena solución en cortos tiempos de ejecución.

Bibliografía

Alba, E. (2014). *NEO: Networking and Emerging Optimization.* Recuperado el 03 de Marzo de 2014, de http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances/

Beasley, J. E. (1983.). Route First - Cluster Second Methods for the Vehicle Routing Problem. *Omega 11.* , págs. 403-408.

Bullnheimer, B., Hartl, R. F., & Strauss, C. (1999.). An Improved Ant System Algorithm for the Vehicle Routing Problem. *Annals of Operations Research 89.* , 319-328.

Dantzig, G. B. (1959). The Truck Dispatching Problem. *Management Science* , 6, 80-91.

Dantzig, G. B., & Ramser, J. H. (1959). The Truck Dispatching Problem. *Managment Science* , 6, 80-91.

Dawson, L., & Stewart, I. A. (2013.). Improving Ant Colony Optimization performance on the GPU using CUDA. *IEEE Congress on Evolutionary Computation* , 1901-1908.

Dorigo, M., & Gambardella, L. M. (1997.). Ant Colonies for the Travelling Salesman Problem. *Biosystems.* , 39.

Dorigo, M., & Stüzle, T. (2004.). *Ant colony Optimization.* MIT. Press.

Feo, T., & Resende, M. G. (1989.). A Probabilistic Heuristic for a Computationally Dificult Covering Problem. *Operations Research Letters.* (8), págs. 67-71.

Gambardella, L., Taillard, É., & Agazzi, G. (1999.). MACS-VRPTW: A multiple colony system for vehicle routing problems with time windows. *New Ideas in Optimization.*

Glover, F. (1989.). Tabu Search, Part 1. *ORSA Journal on Computing. , 1* (3), págs. pp. 190-206.

Holland, J. H. (1992.). *Adaptation in Natural and Artificial Systems.* (M. press., Ed.) Ann Arbor, Michigan.: University of Michigan Press.

Kennedy, J., & Eberhart, R. (1995.). Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks. , IV*, págs. 1942-1948.

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

Larrañaga, P., & Lozano, J. A. (2002.). Estimation of distribution Algorithms. *A New Tool for Evolutionary Computation.*

Nvidia, C. (s.f.). *CUDA Computer Parallel Computing Platform*. Recuperado el 18 de Febrero de 2013, de 2006: https://developer.nvidia.com/what-cuda

Ponce, J. Hernández A., Ochoa A., Quezada F. Ponce de León E. Zavala C. (2013). Solving Routing Problems using Ant Colonies Optimization, and a Parallel Architecture with GPUs. Book Logistics: Perspectives, Approaches and Challenges. Editors: Jinghua Cheung and Huan Song. Nova publishers. pp. 141-156.

Reinelt, G. (2014). *Universität Heidelberg.* Recuperado el 3 de Marzo de 2014, de https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/vrp/

Toth, P., & Vigo, D. (2002.). *The Vehicle Routing Problem.* Philadelphia, USA.: Society of Industrial and Applied Mathematics (SIAM) Monographs on Discrete Mathematics and Applications.

Anexo 3

Resultados de las Pruebas de Diseño Factorial

Resultados obtenidos con las Pruebas de Diseño Factorial con el algoritmo de colonia de hormigas utilizando los parámetros:

alfa	beta	rho	Número de Iteraciones
1	1	0.7	100
3	3	0.9	
5	5		

Instancia	Réplicas										Parámetros		
P-n16-k8.vrp	1	2	3	4	5	6	7	8	9	10	alfa	beta	rho
Solución Óptima 450	450	450	450	450	450	452	450	450	450	450	1	1	0.7
	450	450	450	450	452	452	450	450	450	450	1	1	0.9
	457	450	455	450	455	452	450	450	456	455	1	3	0.7
	456	455	455	450	452	450	453	456	455	450	1	3	0.9
	452	456	450	461	450	450	450	455	450	450	1	5	0.7
	450	456	456	450	455	450	450	450	452	455	1	5	0.9
	472	458	461	455	453	472	463	470	478	463	3	1	0.7
	455	453	479	470	467	468	486	456	463	484	3	1	0.9
	470	456	468	471	467	468	456	450	461	477	3	3	0.7
	481	461	468	468	461	450	450	461	470	461	3	3	0.9
	461	470	478	450	467	467	465	452	461	450	3	5	0.7
	470	461	461	464	481	470	467	461	450	461	3	5	0.9
	488	466	480	464	460	459	478	463	485	487	5	1	0.7
	487	470	466	463	461	453	468	473	479	450	5	1	0.9
	456	456	468	463	484	468	472	472	481	464	5	3	0.7
	466	467	467	461	472	452	450	474	461	463	5	3	0.9
	478	463	467	467	465	463	459	478	456	463	5	5	0.7
	467	476	450	461	456	450	456	478	450	461	5	5	0.9

Instancia	Réplicas										Parámetros		
P-n19-k2.vrp	1	2	3	4	5	6	7	8	9	10	alfa	beta	rho
Solución Óptima	221	222	212	217	222	212	212	215	218	212	1	1	0.7
212	212	212	212	212	221	212	212	212	215	215	1	1	0.9
	212	212	212	212	212	215	212	212	215	212	1	3	0.7
	212	212	212	212	212	212	212	212	218	212	1	3	0.9
	212	212	212	217	217	219	212	212	212	212	1	5	0.7
	212	212	220	212	212	222	212	212	212	212	1	5	0.9
	230	227	250	254	226	228	240	243	241	222	3	1	0.7
	238	250	240	237	238	258	246	248	243	266	3	1	0.9
	231	227	220	221	226	222	225	212	221	225	3	3	0.7
	217	222	222	224	223	217	226	222	225	217	3	3	0.9
	237	225	231	212	212	229	227	218	212	222	3	5	0.7
	220	229	222	228	222	226	220	222	226	225	3	5	0.9
	267	232	242	237	247	242	253	249	231	228	5	1	0.7
	259	257	240	238	260	262	250	250	239	243	5	1	0.9
	228	221	231	233	231	228	228	215	228	222	5	3	0.7
	233	225	226	212	228	222	246	231	222	236	5	3	0.9
	226	220	234	222	224	220	222	222	226	220	5	5	0.7
	220	226	220	228	217	222	222	222	228	220	5	5	0.9

Instancia	Réplicas										Parámetros		
E-n23-k3.vrp	1	2	3	4	5	6	7	8	9	10	alfa	beta	rho
Solución Óptima	572	574	597	572	584	576	577	574	577	572	1	1	0.7
569	579	574	578	588	574	574	576	576	586	576	1	1	0.9
	576	578	576	574	578	576	576	576	577	574	1	3	0.7
	571	571	576	576	576	576	574	576	576	576	1	3	0.9
	576	576	576	577	577	576	577	580	576	576	1	5	0.7
	577	586	576	576	576	579	576	578	577	576	1	5	0.9
	632	628	690	604	609	593	657	612	654	655	3	1	0.7
	590	643	675	698	718	616	665	649	681	576	3	1	0.9
	588	597	597	585	579	600	609	586	578	577	3	3	0.7
	595	597	602	612	598	577	610	586	600	588	3	3	0.9
	578	597	579	597	602	588	576	595	595	588	3	5	0.7
	597	579	579	597	588	595	595	580	588	585	3	5	0.9
	640	640	613	612	720	662	697	732	643	673	5	1	0.7
	724	688	620	610	651	669	692	647	694	682	5	1	0.9
	602	595	612	576	618	626	609	572	606	592	5	3	0.7
	595	609	593	604	654	596	623	607	586	597	5	3	0.9
	597	600	602	604	592	601	606	601	594	597	5	5	0.7
	601	585	602	614	600	597	594	601	602	601	5	5	0.9

Instancia	Réplicas										Parámetros		
A-n33-k5.vrp	1	2	3	4	5	6	7	8	9	10	alfa	beta	rho
Solución Óptima	671	675	687	701	684	688	671	688	685	681	1	1	0.7
661	680	688	687	685	685	684	694	685	681	684	1	1	0.9
	674	677	677	671	671	675	675	671	677	673	1	3	0.7
	677	677	675	677	682	677	679	675	677	675	1	3	0.9
	675	684	675	671	671	685	675	685	677	675	1	5	0.7
	677	671	677	680	675	671	671	682	681	679	1	5	0.9
	711	703	734	712	743	713	723	714	737	727	3	1	0.7
	780	731	762	767	757	769	742	744	752	717	3	1	0.9
	713	682	677	694	692	674	694	685	677	688	3	3	0.7
	677	693	675	714	717	690	680	694	677	681	3	3	0.9
	688	684	675	708	688	688	688	675	671	681	3	5	0.7
	675	671	685	688	685	671	687	671	677	675	3	5	0.9
	725	730	752	729	795	781	754	715	744	783	5	1	0.7
	762	759	736	718	771	742	751	756	816	780	5	1	0.9
	694	702	710	692	699	700	685	720	697	706	5	3	0.7
	675	704	677	692	684	681	688	692	710	708	5	3	0.9
	692	677	675	681	688	680	692	685	692	694	5	5	0.7
	692	692	677	677	677	688	694	675	692	685	5	5	0.9

Instancia	Réplicas										Parámetros		
F-n45-k4.vrp	1	2	3	4	5	6	7	8	9	10	alfa	beta	rho
Solución Óptima	800	817	793	796	799	807	788	805	815	807	1	1	0.7
724	805	809	788	807	797	802	787	796	821	796	1	1	0.9
	765	749	762	775	774	765	743	765	766	767	1	3	0.7
	766	754	743	734	769	767	769	756	758	759	1	3	0.9
	755	768	766	764	758	768	755	747	765	762	1	5	0.7
	768	733	752	737	763	760	770	763	767	763	1	5	0.9
	804	821	812	804	744	812	782	781	804	787	3	1	0.7
	794	823	812	811	828	805	843	776	792	806	3	1	0.9
	777	767	789	756	764	762	764	774	760	787	3	3	0.7
	778	768	769	746	766	772	775	788	778	780	3	3	0.9
	771	757	785	765	771	766	772	761	798	773	3	5	0.7
	778	766	765	766	773	767	770	765	747	788	3	5	0.9
	802	867	840	812	866	928	876	838	813	816	5	1	0.7
	814	873	819	872	837	887	791	813	870	877	5	1	0.9
	794	781	787	772	787	795	785	774	770	789	5	3	0.7
	769	772	776	796	764	804	790	780	781	795	5	3	0.9
	785	787	785	778	784	774	768	779	791	782	5	5	0.7
	784	781	758	764	777	785	785	781	761	770	5	5	0.9

Instancia	Réplicas										Parámetros		
A-n54-k7.vrp	1	2	3	4	5	6	7	8	9	10	alfa	beta	rho
Solución Óptima	1317	1346	1344	1308	1331	1323	1324	1351	1336	1318	1	1	0.7
1167	1313	1358	1281	1322	1342	1338	1341	1312	1323	1339	1	1	0.9
	1264	1244	1267	1260	1275	1257	1276	1263	1259	1273	1	3	0.7
	1270	1280	1264	1267	1279	1270	1231	1255	1270	1268	1	3	0.9
	1246	1273	1254	1262	1262	1260	1279	1260	1259	1244	1	5	0.7
	1255	1252	1259	1264	1264	1256	1264	1261	1260	1261	1	5	0.9
	1322	1274	1313	1334	1267	1328	1340	1296	1306	1247	3	1	0.7
	1292	1337	1328	1330	1342	1349	1293	1309	1340	1303	3	1	0.9
	1313	1298	1283	1304	1263	1284	1268	1286	1270	1266	3	3	0.7
	1281	1261	1248	1280	1279	1242	1288	1262	1277	1270	3	3	0.9
	1301	1273	1263	1258	1276	1287	1289	1278	1256	1231	3	5	0.7
	1286	1249	1274	1283	1275	1247	1274	1245	1266	1262	3	5	0.9
	1386	1312	1311	1365	1310	1338	1355	1364	1322	1367	5	1	0.7
	1372	1390	1345	1354	1347	1301	1386	1364	1374	1385	5	1	0.9
	1294	1272	1312	1264	1274	1315	1318	1297	1297	1322	5	3	0.7
	1306	1259	1255	1278	1267	1253	1273	1281	1264	1271	5	3	0.9
	1255	1299	1266	1255	1301	1317	1286	1272	1310	1292	5	5	0.7
	1289	1277	1265	1273	1292	1272	1318	1281	1294	1308	5	5	0.9

Instancia	Réplicas										Parámetros		
A-n60-k9.vrp	1	2	3	4	5	6	7	8	9	10	alfa	beta	rho
Solución Óptima	1533	1554	1534	1543	1525	1518	1531	1550	1503	1547	1	1	0.7
1354	1517	1529	1523	1552	1512	1540	1505	1536	1558	1543	1	1	0.9
	1448	1463	1458	1459	1459	1459	1445	1461	1466	1462	1	3	0.7
	1465	1472	1455	1466	1464	1464	1470	1463	1445	1465	1	3	0.9
	1472	1461	1427	1460	1459	1437	1463	1452	1461	1474	1	5	0.7
	1459	1436	1463	1447	1463	1466	1465	1464	1457	1463	1	5	0.9
	1508	1508	1539	1508	1509	1495	1506	1511	1504	1501	3	1	0.7
	1521	1495	1474	1517	1496	1521	1536	1489	1517	1503	3	1	0.9
	1446	1464	1491	1454	1471	1456	1449	1485	1463	1463	3	3	0.7
	1472	1480	1495	1496	1478	1510	1493	1487	1461	1479	3	3	0.9
	1465	1485	1489	1474	1463	1498	1470	1487	1468	1464	3	5	0.7
	1483	1474	1492	1438	1469	1497	1477	1455	1486	1471	3	5	0.9
	1541	1525	1511	1545	1574	1542	1558	1492	1535	1481	5	1	0.7
	1571	1543	1559	1532	1543	1565	1538	1545	1490	1521	5	1	0.9
	1505	1472	1498	1486	1480	1479	1479	1448	1473	1493	5	3	0.7
	1519	1499	1465	1474	1468	1491	1494	1496	1495	1491	5	3	0.9
	1468	1501	1478	1485	1471	1458	1499	1498	1477	1503	5	5	0.7
	1494	1478	1506	1494	1481	1437	1464	1481	1458	1506	5	5	0.9

Instancia	Réplicas										Parámetros		
F-n72-k4.vrp	1	2	3	4	5	6	7	8	9	10	alfa	beta	rho
Solución Óptima	296	297	296	293	298	295	296	296	282	300	1	1	0.7
237	285	296	297	304	291	293	285	293	300	299	1	1	0.9
	272	272	271	275	269	270	271	271	268	270	1	3	0.7
	275	274	271	277	273	272	277	272	273	265	1	3	0.9
	274	269	272	274	275	278	273	274	269	272	1	5	0.7
	273	270	272	275	277	276	275	273	273	269	1	5	0.9
	288	274	286	269	302	294	282	301	285	278	3	1	0.7
	278	298	284	285	289	273	296	290	299	286	3	1	0.9
	277	272	279	282	273	267	275	284	275	284	3	3	0.7
	271	276	277	273	276	280	276	270	272	282	3	3	0.9
	278	266	273	272	273	278	283	281	269	276	3	5	0.7
	278	267	283	269	280	282	273	279	275	270	3	5	0.9
	295	310	288	292	297	285	303	286	290	306	5	1	0.7
	280	312	283	304	296	301	310	288	283	302	5	1	0.9
	286	279	283	283	282	286	282	284	283	285	5	3	0.7
	273	287	291	276	287	286	278	283	285	274	5	3	0.9
	278	273	285	271	281	283	282	283	275	275	5	5	0.7
	282	273	278	283	283	279	267	271	276	286	5	5	0.9

Instancia	Réplicas										Parámetros		
E-n76-k10.vrp	1	2	3	4	5	6	7	8	9	10	alfa	beta	rho
Solución Óptima	1021	980	998	1002	975	1008	1011	1007	996	998	1	1	0.7
830	1022	1006	1030	946	1001	1031	989	987	1000	987	1	1	0.9
	898	917	919	931	922	930	919	924	914	930	1	3	0.7
	931	926	923	933	915	919	932	911	918	913	1	3	0.9
	913	914	910	908	915	907	911	918	905	916	1	5	0.7
	913	918	909	918	907	916	915	921	910	914	1	5	0.9
	978	960	962	964	970	972	964	952	952	949	3	1	0.7
	952	959	991	957	973	958	958	967	955	941	3	1	0.9
	920	916	909	923	927	936	940	921	932	925	3	3	0.7
	928	916	917	929	919	933	902	920	921	914	3	3	0.9
	922	928	931	905	918	909	901	929	920	910	3	5	0.7
	914	910	921	912	925	914	899	909	908	922	3	5	0.9
	998	1020	1008	980	1004	1011	974	1002	1040	1003	5	1	0.7
	1018	1002	1019	999	992	1004	1004	984	968	992	5	1	0.9
	924	937	926	916	933	935	923	927	933	930	5	3	0.7
	923	927	920	921	925	916	920	917	932	914	5	3	0.9
	928	907	920	924	928	925	930	926	931	922	5	5	0.7
	908	927	940	930	919	911	923	934	926	934	5	5	0.9

Instancia	Réplicas										Parámetros		
M-n101-k10.vrp	1	2	3	4	5	6	7	8	9	10	alfa	beta	rho
Solución Óptima	968	994	990	982	989	975	989	977	980	981	1	1	0.7
820	995	986	987	998	999	985	1006	971	979	966	1	1	0.9
	904	909	908	907	899	906	902	905	898	897	1	3	0.7
	893	901	900	887	917	864	904	871	895	888	1	3	0.9
	890	896	897	892	897	901	892	890	889	890	1	5	0.7
	882	903	884	898	896	894	876	900	879	887	1	5	0.9
	936	906	938	936	950	934	948	930	963	963	3	1	0.7
	928	938	948	952	925	958	937	926	942	947	3	1	0.9
	927	909	918	902	930	908	918	916	904	912	3	3	0.7
	908	905	912	918	908	912	911	910	926	918	3	3	0.9
	911	912	921	904	909	924	917	907	922	902	3	5	0.7
	903	912	911	923	898	919	895	980	913	899	3	5	0.9
	957	974	950	955	966	955	980	993	940	959	5	1	0.7
	972	1005	984	976	987	938	975	979	950	978	5	1	0.9
	937	912	911	934	932	909	927	921	933	931	5	3	0.7
	884	925	931	943	929	942	914	935	922	949	5	3	0.9
	903	929	943	918	928	906	891	939	942	910	5	5	0.7
	900	918	901	918	912	928	913	917	918	918	5	5	0.9

Anexo 4

Resultados del Test Kolmogorov - Smirnov

1.- P-n16-k8.vrp

Prueba de Kolmogorov-Smirnov para una muestra

		resultado
N		180
Parámetros normales(a,b)	Media	460.92
	Desviación típica	10.217
Diferencias más extremas	Absoluta	.151
	Positiva	.151
	Negativa	-.143
Z de Kolmogorov-Smirnov		2.032
Sig. asintót. (bilateral)		.001

a La distribución de contraste es la Normal.
b Se han calculado a partir de los datos.

2.- P-n19-k2.vrp

Prueba de Kolmogorov-Smirnov para una muestra

		resultado
N		180
Parámetros normales(a,b)	Media	224.96
	Desviación típica	12.978
Diferencias más extremas	Absoluta	.159
	Positiva	.151
	Negativa	-.159
Z de Kolmogorov-Smirnov		2.133
Sig. asintót. (bilateral)		.000

a La distribución de contraste es la Normal.
b Se han calculado a partir de los datos.

3.- E-n23-k3.vrp

Prueba de Kolmogorov-Smirnov para una muestra

		resultado
N		180
Parámetros normales(a,b)	Media	602.58
	Desviación típica	35.358
Diferencias más extremas	Absoluta	.212
	Positiva	.212
	Negativa	-.186
Z de Kolmogorov-Smirnov		2.846
Sig. asintót. (bilateral)		.000

a La distribución de contraste es la Normal.
b Se han calculado a partir de los datos.

4.- A-n33-k5.vrp

Prueba de Kolmogorov-Smirnov para una muestra

		resultado
N		180
Parámetros normales(a,b)	Media	697.76
	Desviación típica	30.038
Diferencias más extremas	Absoluta	.244
	Positiva	.244
	Negativa	-.187
Z de Kolmogorov-Smirnov		3.276
Sig. asintót. (bilateral)		.000

a La distribución de contraste es la Normal.
b Se han calculado a partir de los datos.

5.- F-n45-k4.vrp

Prueba de Kolmogorov-Smirnov para una muestra

		resultado
N		180
Parámetros normales(a,b)	Media	785.67
	Desviación típica	30.555
Diferencias más extremas	Absoluta	.125
	Positiva	.125
	Negativa	-.091
Z de Kolmogorov-Smirnov		1.680
Sig. asintót. (bilateral)		.007

a La distribución de contraste es la Normal.
b Se han calculado a partir de los datos.

6.- A-n54-k7.vrp

Prueba de Kolmogorov-Smirnov para una muestra

		resultado
N		180
Parámetros normales(a,b)	Media	1292.46
	Desviación típica	35.797
Diferencias más extremas	Absoluta	.142
	Positiva	.142
	Negativa	-.071
Z de Kolmogorov-Smirnov		1.908
Sig. asintót. (bilateral)		.001

a La distribución de contraste es la Normal.
b Se han calculado a partir de los datos.

7.- A-n60-k9.vrp

Prueba de Kolmogorov-Smirnov para una muestra

		resultado
N		180
Parámetros normales(a,b)	Media	1490.11
	Desviación típica	31.843
Diferencias más extremas	Absoluta	.105
	Positiva	.105
	Negativa	-.057
Z de Kolmogorov-Smirnov		1.404
Sig. asintót. (bilateral)		.039

a La distribución de contraste es la Normal.
b Se han calculado a partir de los datos.

8.- F-n72-k4.vrp

Prueba de Kolmogorov-Smirnov para una muestra

		resultado
N		180
Parámetros normales(a,b)	Media	281.58
	Desviación típica	10.359
Diferencias más extremas	Absoluta	.118
	Positiva	.118
	Negativa	-.079
Z de Kolmogorov-Smirnov		1.589
Sig. asintót. (bilateral)		.013

a La distribución de contraste es la Normal.
b Se han calculado a partir de los datos.

9.- E-n76-k10.vrp

Prueba de Kolmogorov-Smirnov para una muestra

		resultado
N		180
Parámetros normales(a,b)	Media	942.67
	Desviación típica	35.545
Diferencias más extremas	Absoluta	.235
	Positiva	.235
	Negativa	-.124
Z de Kolmogorov-Smirnov		3.156
Sig. asintót. (bilateral)		.000

a La distribución de contraste es la Normal.
b Se han calculado a partir de los datos.

10.- M-n101-k10.vrp

Prueba de Kolmogorov-Smirnov para una muestra

		resultado
N		180
Parámetros normales(a,b)	Media	928.28
	Desviación típica	32.289
Diferencias más extremas	Absoluta	.130
	Positiva	.130
	Negativa	-.077
Z de Kolmogorov-Smirnov		1.750
Sig. asintót. (bilateral)		.004

a La distribución de contraste es la Normal.
b Se han calculado a partir de los datos.

Resultados del Test Levene

1.- P-n16-k8.vrp

Contraste de Levene sobre la igualdad de las varianzas error(a)

Variable dependiente: resultado

F	gl1	gl2	Significación
4.777	17	162	.000

Contrasta la hipótesis nula de que la varianza error de la variable dependiente es igual a lo largo de todos los grupos.
a Diseño: Intersección+alfa+beta+rho2+alfa * beta+alfa * rho2+beta * rho2+alfa * beta * rho2

2.- P-n19-k2.vrp

Contraste de Levene sobre la igualdad de las varianzas error(a)

Variable dependiente: resultado

F	gl1	gl2	Significación
5.798	17	162	.000

Contrasta la hipótesis nula de que la varianza error de la variable dependiente es igual a lo largo de todos los grupos.
a Diseño: Intersección+alfa+beta+rho2+alfa * beta+alfa * rho2+beta * rho2+alfa * beta * rho2

3.- E-n23-k3.vrp

Contraste de Levene sobre la igualdad de las varianzas error(a)

Variable dependiente: resultado

F	gl1	gl2	Significación
10.891	17	162	.000

Contrasta la hipótesis nula de que la varianza error de la variable dependiente es igual a lo largo de todos los grupos.
a Diseño: Intersección+alfa+beta+rho2+alfa * beta+alfa * rho2+beta * rho2+alfa * beta * rho2

4.- A-n33-k5.vrp

Contraste de Levene sobre la igualdad de las varianzas error(a)

Variable dependiente: resultado

F	gl1	gl2	Significación
5.526	17	162	.000

Contrasta la hipótesis nula de que la varianza error de la variable dependiente es igual a lo largo de todos los grupos.

a Diseño: Intersección+alfa+beta+rho2+alfa * beta+alfa * rho2+beta * rho2+alfa * beta * rho2

5.- F-n45-k4.vrp

Contraste de Levene sobre la igualdad de las varianzas error(a)

Variable dependiente: resultado

F	gl1	gl2	Significación
7.174	17	162	.000

Contrasta la hipótesis nula de que la varianza error de la variable dependiente es igual a lo largo de todos los grupos.

a Diseño: Intersección+alfa+beta+rho2+alfa * beta+alfa * rho2+beta * rho2+alfa * beta * rho2

6.- A-n54-k7.vrp

Contraste de Levene sobre la igualdad de las varianzas error(a)

Variable dependiente: resultado

F	gl1	gl2	Significación
3.218	17	162	.000

Contrasta la hipótesis nula de que la varianza error de la variable dependiente es igual a lo largo de todos los grupos.

a Diseño: Intersección+alfa+beta+rho2+alfa * beta+alfa * rho2+beta * rho2+alfa * beta * rho2

7.- A-n60-k9.vrp

Contraste de Levene sobre la igualdad de las varianzas error(a)

Variable dependiente: resultado

F	gl1	gl2	Significación
2.009	17	162	.013

Contrasta la hipótesis nula de que la varianza error de la variable dependiente es igual a lo largo de todos los grupos.

a Diseño: Intersección+alfa+beta+rho2+alfa * beta+alfa * rho2+beta * rho2+alfa * beta * rho2

8.- F-n72-k4.vrp

Contraste de Levene sobre la igualdad de las varianzas error(a)

Variable dependiente: resultado

F	gl1	gl2	Significación
5.055	17	162	.000

Contrasta la hipótesis nula de que la varianza error de la variable dependiente es igual a lo largo de todos los grupos.

a Diseño: Intersección+alfa+beta+rho2+alfa * beta+alfa * rho2+beta * rho2+alfa * beta * rho2

9.- E-n76-k10.vrp

Contraste de Levene sobre la igualdad de las varianzas error(a)

Variable dependiente: resultado

F	gl1	gl2	Significación
2.454	17	162	.002

Contrasta la hipótesis nula de que la varianza error de la variable dependiente es igual a lo largo de todos los grupos.

a Diseño: Intersección+alfa+beta+rho2+alfa * beta+alfa * rho2+beta * rho2+alfa * beta * rho2

10.- M-n101-k10.vrp

Contraste de Levene sobre la igualdad de las varianzas error(a)

Variable dependiente: resultado

F	gl1	gl2	Significación
2.014	17	162	.013

Contrasta la hipótesis nula de que la varianza error de la variable dependiente es igual a lo largo de todos los grupos.

a Diseño: Intersección+alfa+beta+rho2+alfa * beta+alfa * rho2+beta * rho2+alfa * beta * rho2

Resultados del Test Kruskal – Wallis

1.- P-n16-k9.vrp

Rangos

	categorías	N	Rango promedio
resultado	1	10	26.25
	2	10	29.00
	3	10	50.25
	4	10	52.90
	5	10	43.30
	6	10	45.35
	7	10	115.40
	8	10	125.00
	9	10	114.70
	10	10	103.95
	11	10	98.95
	12	10	113.40
	13	10	140.10
	14	10	120.80
	15	10	132.20
	16	10	108.30
	17	10	121.40
	18	10	87.75
	Total	180	

Estadísticos de contraste(a,b)

	resultado
Chi-cuadrado	93.045
Gl	17
Sig. asintót.	.000

a Prueba de Kruskal-Wallis
b Variable de agrupación: categorías

2.- P-n19-k2.vrp

Rangos

	categorías	N	Rango promedio
resultado	1	10	53.05
	2	10	35.25
	3	10	29.90
	4	10	28.35
	5	10	35.25
	6	10	35.75
	7	10	140.50
	8	10	161.90
	9	10	93.15
	10	10	87.50
	11	10	87.60
	12	10	101.25
	13	10	155.70
	14	10	166.40
	15	10	115.10
	16	10	114.80
	17	10	96.30
	18	10	91.25
	Total	180	

Estadísticos de contraste(a,b)

	resultado
Chi-cuadrado	139.310
gl	17
Sig. asintót.	.000

a Prueba de Kruskal-Wallis

b Variable de agrupación: categorías

3.- E-n23-k3.vrp

Rangos

	categorías	N	Rango promedio
resultado	1.00	10	33.75
	2.00	10	39.70
	3.00	10	33.30
	4.00	10	22.00
	5.00	10	39.30
	6.00	10	44.25
	7.00	10	145.50
	8.00	10	142.90
	9.00	10	85.85
	10.00	10	103.70
	11.00	10	83.80
	12.00	10	82.85
	13.00	10	161.20
	14.00	10	163.55
	15.00	10	104.55
	16.00	10	117.00
	17.00	10	112.55
	18.00	10	113.25
	Total	180	

Estadísticos de contraste(a,b)

	resultado
Chi-cuadrado	136.419
Gl	17
Sig. asintót.	.000

a Prueba de Kruskal-Wallis

b Variable de agrupación: categorías

4.- A-n33-k5.vrp

Rangos

	categorías	N	Rango promedio
resultado	1.00	10	69.40
	2.00	10	85.35
	3.00	10	24.90
	4.00	10	44.65
	5.00	10	41.75
	6.00	10	40.25
	7.00	10	146.30
	8.00	10	164.30
	9.00	10	86.55
	10.00	10	88.45
	11.00	10	74.15
	12.00	10	48.80
	13.00	10	162.50
	14.00	10	166.05
	15.00	10	124.85
	16.00	10	94.60
	17.00	10	84.80
	18.00	10	81.35
	Total	180	

Estadísticos de contraste(a,b)

	resultado
Chi-cuadrado	127.861
gl	17
Sig. asintót.	.000

a Prueba de Kruskal-Wallis

b Variable de agrupación: categorías

5.- F-n45-k4.vrp

Rangos

	categorías	N	Rango promedio
resultado	1.00	10	141.10
	2.00	10	137.30
	3.00	10	42.15
	4.00	10	29.90
	5.00	10	31.85
	6.00	10	30.50
	7.00	10	121.10
	8.00	10	143.65
	9.00	10	58.10
	10.00	10	71.45
	11.00	10	65.60
	12.00	10	58.25
	13.00	10	165.25
	14.00	10	165.20
	15.00	10	101.65
	16.00	10	96.35
	17.00	10	96.20
	18.00	10	73.40
	Total	180	

Estadísticos de contraste(a,b)

	resultado
Chi-cuadrado	133.703
Gl	17
Sig. asintót.	.000

a Prueba de Kruskal-Wallis

b Variable de agrupación: categorías

6.- A-n54-k7.vrp

Rangos

	categorías	N	Rango promedio
resultado	1.00	10	148.15
	2.00	10	143.85
	3.00	10	44.80
	4.00	10	53.30
	5.00	10	33.90
	6.00	10	30.55
	7.00	10	109.75
	8.00	10	138.75
	9.00	10	86.20
	10.00	10	59.20
	11.00	10	63.65
	12.00	10	53.20
	13.00	10	154.35
	14.00	10	167.00
	15.00	10	105.95
	16.00	10	58.45
	17.00	10	84.50
	18.00	10	93.45
	Total	180	

Estadísticos de contraste(a,b)

	resultado
Chi-cuadrado	123.378
gl	17
Sig. asintót.	.000

a Prueba de Kruskal-Wallis

b Variable de agrupación: categorías

7.- A-n60-k9.vrp

Rangos

	categorías	N	Rango promedio
resultado	1.00	10	156.70
	2.00	10	154.70
	3.00	10	26.15
	4.00	10	42.90
	5.00	10	30.30
	6.00	10	31.25
	7.00	10	132.85
	8.00	10	125.45
	9.00	10	42.25
	10.00	10	89.90
	11.00	10	72.00
	12.00	10	68.90
	13.00	10	148.65
	14.00	10	159.55
	15.00	10	83.50
	16.00	10	97.60
	17.00	10	86.95
	18.00	10	79.40
	Total	180	

Estadísticos de contraste(a,b)

	resultado
Chi-cuadrado	134.789
Gl	17
Sig. asintót.	.000

a Prueba de Kruskal-Wallis
b Variable de agrupación: categorías

8.- F-n72-k4.vrp

Rangos

	categorías	N	Rango promedio
resultado	1.00	10	152.55
	2.00	10	152.40
	3.00	10	23.65
	4.00	10	43.70
	5.00	10	42.90
	6.00	10	45.45
	7.00	10	114.00
	8.00	10	127.75
	9.00	10	70.60
	10.00	10	59.15
	11.00	10	57.15
	12.00	10	61.25
	13.00	10	153.30
	14.00	10	149.00
	15.00	10	113.70
	16.00	10	104.00
	17.00	10	81.60
	18.00	10	76.85
	Total	180	

Estadísticos de contraste(a,b)

	resultado
Chi-cuadrado	119.109
gl	17
Sig. asintót.	.000

a Prueba de Kruskal-Wallis

b Variable de agrupación: categorías

9.- E-n76-k10.vrp

Rangos

	categorías	N	Rango promedio
resultado	1.00	10	159.55
	2.00	10	158.40
	3.00	10	63.70
	4.00	10	67.05
	5.00	10	24.75
	6.00	10	33.75
	7.00	10	132.70
	8.00	10	131.70
	9.00	10	76.90
	10.00	10	60.05
	11.00	10	51.40
	12.00	10	34.80
	13.00	10	162.55
	14.00	10	158.10
	15.00	10	92.45
	16.00	10	65.35
	17.00	10	77.35
	18.00	10	78.45
	Total	180	

Estadísticos de contraste(a,b)

	resultado
Chi-cuadrado	140.486
Gl	17
Sig. asintót.	.000

a Prueba de Kruskal-Wallis

b Variable de agrupación: categorías

10.- M-n101-k10.vrp

Rangos

	categorías	N	Rango promedio
resultado	1.00	10	164.35
	2.00	10	167.45
	3.00	10	43.80
	4.00	10	25.55
	5.00	10	19.15
	6.00	10	17.15
	7.00	10	120.65
	8.00	10	123.30
	9.00	10	74.05
	10.00	10	71.10
	11.00	10	70.50
	12.00	10	68.70
	13.00	10	147.35
	14.00	10	155.65
	15.00	10	97.55
	16.00	10	101.50
	17.00	10	85.95
	18.00	10	75.25
	Total	180	

Estadísticos de contraste(a,b)

	resultado
Chi-cuadrado	146.097
gl	17
Sig. asintót.	.000

a Prueba de Kruskal-Wallis

b Variable de agrupación: categorías