ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE CIENCIAS NATURALES Y MATEMÁTICAS
DEPARTAMENTO DE MATEMÁTICAS

TESIS DE GRADO

PREVIO A LA OBTENCIÓN DEL TÍTULO DE:
"MAGÍSTER EN CONTROL DE OPERACIONES Y GESTIÓN LOGÍSTICA"

TEMA:
"DESARROLLO DE UN SISTEMA DE INFORMACIÓN DSS PARA EL RUTEO DE VEHÍCULOS"

AUTOR:
TANIA DENIS CASTRO RUIZ

GUAYAQUIL – ECUADOR
2014

DEDICATORIA

A Dios. A mi familia: mis padres Juan Castro Escudero y Roxana Ruiz de Castro, y mi hermana Ambar.

AGRADECIMIENTOS

A mis profesores por todas sus enseñanzas, en especial a mi Director de Tesis el profesor Carlos Martín. A todos mis amigos por su amistad, consejos y sugerencias.

DECLARACIÓN EXPRESA

La responsabilidad por los hechos y doctrinas expuestas en esta TESIS DE MAESTRÍA, me corresponde exclusivamente; el patrimonio intelectual del mismo, corresponde exclusivamente a la **Facultad de Ciencias Naturales y Matemáticas (FCNM), Departamento de Matemáticas** de la Escuela Superior Politécnica del Litoral (ESPOL).

(Reglamento de Graduación de la ESPOL)

_____

Tania Denis Castro Ruiz

TRIBUNAL DE GRADUACIÓN

_____

Doctor Peter Iza
PRESIDENTE DEL TRIBUNAL

_____

M. Sc. Carlos Martín B.
DIRECTOR DE TESIS

_____

M. Sc. Xavier Cabezas
VOCAL DEL TRIBUNAL

AUTOR DE LA TESIS

Tania Denis Castro Ruiz

RESUMEN

Las organizaciones cada vez más están tomando conciencia de la importancia de una buena gestión de su logística, y el enrutamiento de vehículos es una parte importante dentro de esta gran tarea. Con esa motivación, el objetivo principal de esta tesis es desarrollar un sistema de información DSS para el ruteo de vehículos, que además tenga la ventaja de operar en el web, y de ofrecer una interfaz gráfica de usuario muy amigable y fácil de usar.

Actualmente se conoce muchísimo respecto del enrutamiento de vehículos. Existen métodos exactos y métodos inexactos. El inconveniente de los primeros es que a pesar de que proporcionan la solución óptima del problema, cuando el número de puntos a visitar es del orden de las decenas o peor aún, de las centenas o más, la tecnología de los computadores actuales NO permite encontrar la ruta óptima en un tiempo razonable. Por el contrario, los métodos inexactos nos permiten obtener en un tiempo adecuado y viable, sino la solución óptima, una "buena" solución, pero son más complicados de implementar.

Existen muchos tipos de problemas relacionados con el ruteo de vehículos. El problema que se escogió para esta tesis es el CVRP simétrico. Se tienen n puntos distribuidos sobre un plano (por ejemplo ubicaciones en una ciudad). El problema a resolver consiste en determinar el orden en que se deben ir visitando cada uno de los n puntos de forma tal que se pueda minimizar, por ejemplo, la distancia recorrida o el tiempo de viaje.
El objetivo es que el tomador de decisiones de la empresa, por ejemplo un gerente de transporte, pueda administrar de manera óptima el enrutamiento de sus vehículos.

Se desea resolver el problema haciendo uso de heurísticas (algoritmos glotones) y metaheurísticas (algoritmos evolutivos). Una heurística importante a usar será la del "vecino más cercano". Para las metaheurísticas se desea considerar algoritmos evolutivos, más precisamente, el algoritmo de búsqueda dispersa (scatter search).

La única fuente de datos a la que se va a recurrir es la que proporciona GOOGLE EARTH en cuanto a las coordenadas geográficas de un punto terrestre, es decir, longitud y latitud de un punto cualquiera sobre el planeta tierra. Para medir la distancia entre puntos se usará la "métrica euclidiana" y la conocida "métrica del taxi", pero previamente se deben transformar las coordenadas geográficas a coordenadas UTM, antes de proceder a calcular la distancia entre dos puntos.

Existen muchas metaheurísticas, entre ellas: el algoritmo genético, la búsqueda tabú, el recocido simulado, GRASP, que pueden utilizarse, junto con heurísticas, para resolver problemas de optimización. "Búsqueda dispersa" es también un método metaheurístico que se puede emplear para resolver problemas de optimización y forma parte de los algoritmos evolutivos. Tiene sus orígenes en los años setenta, pero es en la última década cuando ha sido rediseñado y probado en muchos problemas con un alto grado de dificultad y con excelentes resultados. Esta es la principal razón por la cual se escogió el método de búsqueda dispersa para resolver el problema de transporte antes explicado. Otro objetivo de esta tesis es hacer un estudio profundo de cada una de las etapas de la metaheurística de búsqueda dispersa.

En el primer capítulo se presentan conceptos y definiciones que serán importantes para las cuestiones que se analizan y se discuten en los capítulos posteriores. Luego, en el segundo capítulo, se muestra en primer lugar cómo trabaja en general la metaheurística de búsqueda dispersa cuando intenta resolver un problema de optimización, para posteriormente explicar con todo detalle cómo se ha utilizado la búsqueda dispersa para resolver puntualmente el problema del CVRP simétrico. Se discuten asuntos de diseño e implementación. El tercer capítulo muestra el software, las opciones que presenta al usuario, y experimentos computacionales con datos reales tomados de la ciudad de Guayaquil. Se concluye este documento de tesis con las correspondientes conclusiones y recomendaciones sobre todo el trabajo realizado.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

CAPÍTULO 1

CONCEPTOS Y FUNDAMENTOS

En este capítulo se presentan todas las definiciones y conceptos importantes que se utilizan en este documento de tesis. El objetivo es proporcionar los fundamentos teóricos que se emplearán y que sirven de base en los capítulos posteriores.

1.1 SISTEMA DE INFORMACIÓN DSS

Un sistema de información DSS (Decision Support System) es un software de negocios cuya principal ventaja es proporcionar soporte para la toma de decisiones no rutinarias, es decir, decisiones que no están involucradas directamente con las operaciones diarias del negocio como la compra de insumos y la venta de productos finales. Para la toma de decisiones rutinarias existen otros tipos de sistemas de información como los MIS (Management Information System) y los TPS (Transaction Processing System) [1].

Dentro de la clasificación de los DSS existe uno en particular que Laudon [1] lo define como DSS basado en modelos matemáticos. Este tipo de DSS básicamente utiliza modelos estadísticos y de optimización para hacer cálculos matemáticos que generarán la información que servirá para la toma de decisiones en la empresa. A partir de este punto del documento, cuando se mencione DSS, se hará referencia a los DSS orientados a modelos matemáticos de optimización.

Por mencionar, en Laudon [1] se definen también DSS basados en datos. Este tipo de DSS se encarga de efectuar consultas sobre una base de datos que generalmente no usa el modelo relacional (modelo orientado principalmente a sistemas de información MIS y TPS). Se trata de una base de datos que emplea el esquema estrella, donde existen tablas de hechos y tablas de dimensiones. Estas bases de datos se caracterizan por almacenar grandes cantidades de datos. Por ejemplo, un hecho puede ser el *costo de producción* sujeto a las dimensiones de producto, fábrica, región y tiempo. Los DSS basados en datos efectúan las conocidas consultas multidimensionales [1].

1.2 SISTEMA DE INFORMACIÓN DISTRIBUIDO

En general, un sistema de información es un software de negocios, es decir, un programa de computadora que sirve de apoyo a las organizaciones. Se trata de tecnología que automatiza muchas de las operaciones de una compañía y permite la toma de decisiones a todo nivel. Actualmente, los sistemas de información operan de manera distribuida.

Un programa de computadora (software) es un conjunto de instrucciones que permiten realizar una tarea, pero el programa como tal "no tiene vida". Es el procesador de un equipo de cómputo (hardware) el que le da la vida en el momento en que se encarga de ejecutar dichas instrucciones. A un programa en ejecución se lo llama proceso [2].

Un sistema de información se dice que es distribuido cuando los diferentes procesos que lo conforman ejecutan en distintos computadores [2]. Durante la ejecución del sistema de información estos procesos se comunican, cooperan y coordinan para poder cumplir con las actividades que le corresponden al sistema de información. Aquí es importante indicar que los computadores donde ejecutan los procesos deberían estar enlazados a través de una *red de computadores* para que efectivamente pueda darse la comunicación entre los procesos. Cada vez que un proceso le envía un requerimiento a otro solicitándole algo (generalmente acceso a un recurso, como son los datos) juega el rol de *cliente*, por otro lado, el proceso que recibe una petición, y la atiende, juega el rol de *servidor*. El concepto cliente/servidor NO es hardware, es software [2]. Sin embargo, cliente/servidor sí involucra hardware.

El servidor es un proceso que administra un recurso (usualmente datos almacenados en algún repositorio). Normalmente muchos procesos clientes le envían peticiones a un servidor para tener acceso al recurso que dicho servidor gestiona. Entonces, debido a que el servidor puede recibir un sinnúmero de peticiones (pueden llegar a ser miles o millones, incluso de forma concurrente), necesita un entorno de hardware apropiado (número de procesadores, memoria principal y secundaria) para poder atender todas las peticiones con tiempos de respuesta aceptables. El cliente es un proceso que puede ejecutar en una computadora portátil, una computadora de escritorio o incluso en un

dispositivo móvil. Dado el avance tecnológico que vivimos actualmente, se puede afirmar que casi todos los sistemas de información son distribuidos.

Dentro de la terminología cliente/servidor existe un concepto importante que es el de *middleware*. Se entiende por middleware todo software que permite o que facilita la comunicación entre un proceso cliente y un proceso servidor. Normalmente dentro del middleware reposan los protocolos de comunicación. Con toda esta introducción se procede ahora a enunciar una definición de cliente/servidor.

"Cliente/Servidor es una arquitectura de diseño de software que es el resultado de la subdivisión de un sistema de información en procesos servidores que pueden ejecutar en variadas plataformas, que administran recursos, y que proveen servicios a un gran número de procesos clientes sobre diferentes plataformas, interconectados todos mediante redes de área local o de área extendida y utilizando uno o varios protocolos para comunicarse". En la figura 1.1 se ilustra la definición que se propone.

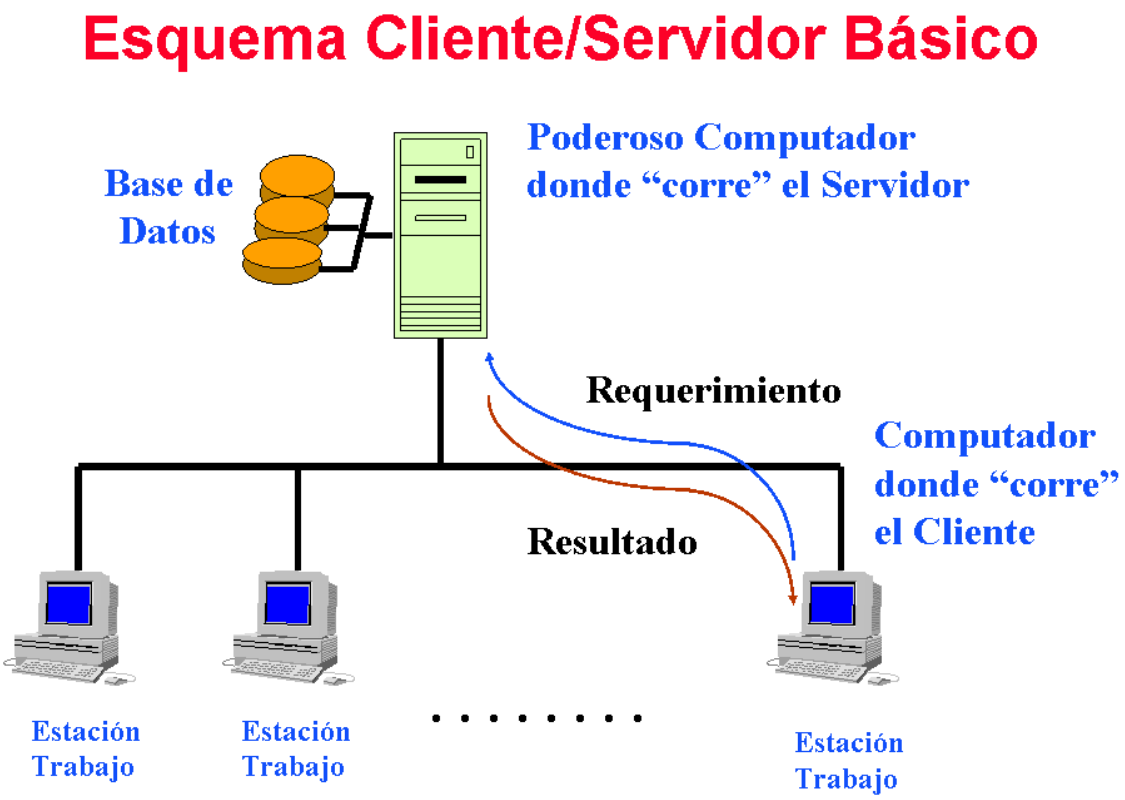

Figura # 1.1 ***Esquema Cliente/Servidor Básico***

1.3 TECNOLOGÍA DE LA INFORMACIÓN Y LAS COMUNICACIONES (TIC's)

TIC es un término que involucra toda forma de tecnología que permita recolectar, procesar, almacenar, generar, distribuir, intercambiar y usar información en todos sus formatos o presentaciones (datos, voz, video), y en él se incluye tanto a las ciencias de la computación como de las telecomunicaciones.

También se puede decir que TIC son todos los elementos de hardware y software, que en conjunto y con la interpretación que se le dan a los datos que en ellos se manipulan, permiten generar información y compartirla.

Muchas veces se hace una diferencia entre datos e información. Se dice que un dato es una cadena alfanumérica concatenada en bruto, es decir, simplemente una secuencia de números y letras que carecen de significado. Cuando un dato es interpretado o cuando a un dato se le añade significado, se dice que se tiene información. Por esta razón se dice que la información son datos con significado o datos procesados. Para procesar los datos necesitamos componentes de hardware y software.

El desarrollo de las telecomunicaciones ha sido muy importante y ha contribuido para facilitar el intercambio y la distribución de la información sin importar la geografía. Esta independencia de ubicación la vemos reflejada como un ejemplo palpable y real en la gran red de redes: Internet.

Se consideran componentes de la tecnología de la información y las comunicaciones, los siguientes:

- Todo tipo de computador o dispositivo de procesamiento que incluya una interfaz gráfica de usuario (es de observar que los teléfonos celulares y los dispositivos móviles actuales se consideran como TIC)

- Todo tipo de periférico que no pueda operar a menos que se encuentre conectado a un computador o a una red

- Todo tipo de red que soporte tráfico de voz, video, datos y también el equipo necesario para que dicha red pueda operar

- Todo tipo de sistema operativo moderno

- Todo tipo de software aplicativo (incluye los sistemas de información distribuidos)

- No debería considerarse como TIC ningún dispositivo que opere de forma completamente aislada o independiente sin ninguna capacidad de integración.

Por todo lo dicho anteriormente es difícil pensar que exista algún fabricante que construya un producto tecnológico sin la capacidad de integrarse. Estamos viviendo en plena era de la revolución de la información y las telecomunicaciones.

1.4 SISTEMA DE INFORMACIÓN LOGÍSTICA

Un *sistema de información logística* es un sistema de información que es parte de los sistemas de información de la empresa que proporcionan soporte en alguna de las fases o etapas de la cadena de suministro. La cadena de suministro son todas las actividades que van desde la adquisición de los insumos o materia prima con los proveedores, pasando por el manejo de inventarios, ubicación de las plantas de fabricación, elaboración de los productos finales, distribución a los clientes e incluso una retroalimentación posterior a la venta. Si un sistema de información está colaborando en alguna de estas actividades, entonces es un sistema de información logística. Se puede decir entonces que la gran mayoría de los sistemas de información de una empresa son logísticos, ya que un gran porcentaje de ellos participa directa o indirectamente en las actividades de la cadena de suministro.

Figura # 1.2 ***Sistemas de Información Logística***

Los sistemas de información como los ERP y los CRM son sistemas de información logística ya que dan soporte a las actividades de la cadena de suministro. Sistemas de información de compras, sistemas de información de ventas, sistemas de información de producción son ejemplos de sistemas de información logística. Los sistemas de información para el ruteo de vehículos que distribuyen los productos terminados de la compañía son sistemas de información logística. Incluso los sistemas de información logística más sofisticados pueden hacer proyecciones, pronósticos, presupuestos, análisis financieros de proyectos, identificación de patrones de consumo y estar basados fuertemente en modelos matemáticos. Se puede observar en la figura 1.2 los componentes principales que forman un sistema de información logística.

1.5 EL PROBLEMA DE RUTEO DE VEHÍCULOS (PRV)

Los problemas de enrutamiento de vehículos son aquellos que tienen que ver con la *distribución de bienes* desde los depósitos o almacenes donde se encuentran los productos terminados, hacia ubicaciones determinadas por los consumidores o clientes. Sin embargo, existen problemas *similares* que pueden ser resueltos usando los mismos algoritmos y modelos que resuelven problemas PRV, como por ejemplo: limpieza de

calles, ruteo de buses para estudiantes, planificación de visitas de vendedores, entre otros.

En un problema PRV se deben distribuir bienes ubicados en uno o más depósitos, en un tiempo dado, a un conjunto de clientes, utilizando un conjunto de vehículos, conducidos por un conjunto de choferes y que realizan sus movimientos o maniobras sobre una red de caminos traficada. Obviamente, la solución del problema involucra determinar el conjunto de rutas, cada una seguida por un único vehículo que comienza y termina en su propio almacén o depósito, de forma tal que todos los requerimientos de los clientes se satisfagan, cumpliéndose también los requerimientos o restricciones operacionales y, todo, a un costo mínimo.

La red de caminos usada para la transportación de los bienes es tradicionalmente representada por un grafo que podría ser dirigido, no dirigido o mixto. Los arcos y/o aristas representan las rutas mientras que los vértices representan depósitos y ubicaciones de clientes. Cada arco y/o arista tiene asociado un peso el cual podría representar la longitud y/o tiempo de viaje de la ruta lo que podría depender también del tipo de vehículo usado e incluso del horario en el cual se sigue la ruta respectiva. En general, en los problemas PRV, cuestiones a considerar son las características de los vehículos (costos, capacidad), los horarios de trabajo, la capacidad de planta y la distribución geográfica de los depósitos, así como también los horarios de trabajo, demandas, ventas y ubicaciones de los clientes.

A continuación se verá la formulación del problema PRV más básico de todos (PRV Capacitado), en su esquema asimétrico como en el modo simétrico, y se dará una breve explicación de cada uno de los componentes del modelo matemático.

- **PRV Capacitado (Modelo Asimétrico)**

El vértice 0 es para el único depósito y los vértices 1 hasta n son para los clientes.

Variables de Decisión:

x_{ij}, toma el valor de 1 si el vehículo pasa por el arco (i, j) y toma el valor de 0 si el vehículo no pasa por el arco (i, j)

Se trata de un problema de minimización de la función objetivo:

MIN $z = \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}$ donde c_{ij} representa el peso de cada arco

Sujeto a las restricciones:

(1) $\sum_{i \in V} x_{ij} = 1;\ j = 1,2,\text{K},n$ lo que quiere decir que exactamente un arco debe entrar en el vértice asociado a cada cliente

(2) $\sum_{j \in V} x_{ij} = 1;\ i = 1,2,\text{K},n$ lo que quiere decir que exactamente un arco debe salir del vértice asociado a cada cliente

(3) $\sum_{i \in V} x_{i0} = K$, donde K representa el número de vehículos. Esta restricción nos dice que K arcos deben entrar al depósito

(4) $\sum_{j \in V} x_{0j} = K$ lo que quiere decir que K arcos deben salir del único depósito

(5) $\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq r(S)$ donde $S \subseteq V - \{0\}$ y $S \neq \phi$. El conjunto S es un subconjunto arbitrario de clientes y $r(S)$ representa el mínimo número de vehículos que atiende a los clientes en S.

(6) Las variables de decisión son enteras binarias: $x_{ij} \in \{0,1\}\ \forall i, j \in V$

Esta es la formulación para un problema básico de PRV: el PRVC. Todos los clientes corresponden a entregas y las demandas son determinísticas, conocidas con anticipación y no pueden ser divididas o repartidas. Todos los vehículos son idénticos y deben partir de un único depósito; la única restricción impuesta para los vehículos es su capacidad.

El objetivo es minimizar el costo total que permita atender a todos los n clientes.

Un costo no negativo c_{ij} está asociado a cada arco (i, j) y representa el costo de desplazarse desde el vértice i hacia el vértice j. Se evitan lazos haciendo que $c_{ii} = +\infty$ $\forall i \in V$ donde V es el conjunto de vértices. Si el grafo G es dirigido se dice que se tiene el modelo PRVC asimétrico. Si G es un grafo no dirigido estamos ante el modelo PRVC simétrico por lo que $c_{ij} = c_{ji}$ $\forall i, j \in V$

El problema PRVC consiste en encontrar exactamente K circuitos simples, cada uno correspondiente a la ruta de un vehículo, con mínimo costo definido como la suma de los costos de los arcos que pertenecientes a los circuitos.

- **PRV Capacitado (Modelo Simétrico)**

El vértice 0 es para el único depósito y los vértices 1 hasta n son para los clientes. En este modelo las rutas son no orientadas, es decir, se utiliza un grafo no dirigido.

Variables de Decisión:
x_{ij}, toma el valor de 1 si el vehículo pasa por los vértices i y j, y toma el valor de 0 si el vehículo no pasa por los vértices i y j. Cabe indicar que $x_{ij} = x_{ji}$ ya que se emplean aristas en el grafo y no arcos.

Es un problema de minimización de la función objetivo:

MIN $z = \sum_{i \in V-\{n\}} \sum_{j>i} c_{ij} x_{ij}$ donde c_{ij} representa el peso de cada arista

Sujeto a las restricciones:

(1) $\sum_{h<i} x_{hi} + \sum_{j>i} x_{ij} = 2$; $i = 1,2,\text{K},n$ lo que quiere decir que exactamente dos aristas inciden en el vértice asociado a cada cliente

(2) $\sum_{j \in V-\{0\}} x_{0j} = 2K$, donde K representa el número de vehículos. Esta restricción nos dice que $2K$ arcos son incidentes en el vértice 0 del depósito

(3) $\sum_{i \in S} \sum_{\substack{h<i \\ h \notin S}} x_{hi} + \sum_{i \in S} \sum_{\substack{j>i \\ j \notin S}} x_{ij} \geq 2\, r(S)$ donde $S \subseteq V - \{0\}$ y $S \neq \phi$. El conjunto S es un subconjunto arbitrario de clientes y $r(S)$ representa el mínimo número de vehículos que atiende a los clientes en S.

(4) $x_{ij} \in \{0,1\} \ \forall i, j \in V - \{0\}$ con $i < j$

(5) $x_{0j} \in \{0,1,2\} \ \forall j \in V - \{0\}$

Ahora se van a explicar brevemente tres variantes al problema PRVC: PRV con ventanas de tiempo, PRV con particionamiento y PRV en *modo recoger y entregar*.

- **PRV con Ventanas de Tiempo**

Es un problema PRVC en el cual restricciones de capacidad son impuestas y cada cliente i está asociado con un intervalo $[a_i, b_i]$ llamado la ventana de tiempo de dicho cliente. El instante de tiempo en el cual los vehículos dejan el depósito, el tiempo de viaje y el tiempo que transcurre cuando se está brindando el servicio al cliente son considerados. El servicio de cada cliente debe comenzar dentro del intervalo de tiempo asociado y el vehículo debe detenerse en la ubicación del cliente el tiempo justo que permita proporcionar el servicio y terminar en un tiempo que se encuentre dentro del intervalo respectivo. Si el arribo del vehículo es muy temprano se permite incluso que el vehículo espere hasta que se encuentre con el punto izquierdo frontera del intervalo a_i y evitar así desincronizaciones. Debido a los problemas de tiempo involucrados normalmente este tipo de problema es modelado de forma asimétrica.
Como en las otras variaciones de problemas PRV se deben encontrar K simples circuitos a un mínimo costo. Consideraciones importantes son: cada circuito debe visitar el depósito, cada cliente es visitado por un circuito y solamente uno, la suma de las demandas de los vértices visitados por un circuito no puede exceder la capacidad C del vehículo que sigue dicha ruta, para cada cliente el servicio comienza y termina dentro de su ventana de tiempo correspondiente.

- **PRV con Particionamientos**

Es otra extensión del problema PRVC en donde el conjunto de clientes es dividido en dos subconjuntos. La idea es que los clientes del primer subconjunto tienen prioridad y requieren una cantidad dada del producto que se debe entregar, por otro lado, los clientes del segundo grupo tienen una prioridad menor en cuanto a la cantidad del producto que debe ser entregada. En esta variante existe una restricción especial:

siempre que una ruta sirva ambos tipos de clientes, todos los clientes del primer subconjunto deben ser atendidos antes que cualquiera de los clientes del segundo subconjunto. Esta restricción nos hace notar la precedencia o prioridad que tiene cualquier cliente del primer subconjunto respecto de cualquier cliente del segundo subconjunto.

Respecto de las consideraciones del modelo, adicionales a las mencionadas anteriormente, cabría indicar que cada circuito debe pasar por el depósito, cada cliente es visitado exactamente por un único circuito, las demandas totales de los clientes visitados por un circuito, ya sean del primero o segundo subconjunto, no deben exceder la capacidad C del vehículo y, en cada circuito, los clientes del primer subconjunto deben ser visitados primero que los clientes del segundo subconjunto. Debido a la precedencia entre los clientes y a la última restricción, es de observar que existen circuitos mixtos, es decir, vehículos que atienden a los dos tipos de clientes.

- **PRV en Modo *Recoger y Entregar***

En esta modalidad o esquema de PRV, cada cliente i tiene dos cantidades asociadas d_i y p_i, las cuales representan las demandas de entregar y de recoger, respectivamente. Se supone que, por cada locación de un cliente, la entrega es realizada antes que la recogida, por ende, la carga actual del vehículo antes de arribar a la ubicación donde se encuentra el cliente, está definida por la carga inicial del vehículo menos todas las demandas ya entregadas y más todas las recogidas ya realizadas. Igual que en las otras modalidades ya explicadas, se deben encontrar K circuitos simples con mínimo costo.

Las consideraciones son: cada circuito debe pasar por el depósito, cada cliente es visitado exactamente por un circuito, la carga actual de cada vehículo asociado a un circuito debe ser no negativa y nunca debe exceder la capacidad C del vehículo, para cada cliente es importante primero hacer la entrega y después la recogida en el mismo circuito. Es importante indicar que pueden, dentro de las variantes, implementar modelos híbridos, por ejemplo: PRV particionado y con ventanas de tiempo, PRV de *recoger y entregar* y particionado, PRV con ventanas de tiempo y en modalidad *recoger y entregar*, PRV particionado con ventanas de tiempo y en modo *recoger y entregar*. Es importante también decir que todas estar variantes híbridas soportan simetrías y asimetrías.

1.6 HEURÍSTICAS Y METAHEURÍSTICAS

La palabra heurística proviene de un término griego cuyo significado es *hallar*, *descubrir*. Se puede entonces definir una heurística como una técnica para resolver problemas donde el objetivo principal es buscar. Dicha búsqueda está basada en un conjunto de reglas, es decir, se trata de un procedimiento sistemático y organizado. Los pasos a seguir en una heurística nos dirán cómo proceder y qué inconvenientes evitar durante una búsqueda en particular [4] [5]. Por ejemplo, en los problemas de optimización se usan las heurísticas. Un modelo matemático típico de optimización es de la forma:

$$MAX \ o \ MIN \ f(X)$$
$$SUJETO \ A \ X \in \Omega$$

El objetivo en el problema de optimización es buscar aquel elemento $X^* \in \Omega$ que hace que la función f tome el valor más alto posible (*MAX*), o que permite que f tome el valor más bajo posible (*MIN*). La función f puede ser lineal o no lineal y se la conoce como la función objetivo. Al conjunto Ω se lo conoce como el conjunto de soluciones factibles o espacio de búsqueda [4] [5]. Existe un conjunto universo Ψ, tal que $\Omega \subseteq \Psi$. Cualquier elemento del conjunto $\Psi - \Omega$ es llamado una solución no factible. Si $\Omega = \Psi$ se dice que el problema no tiene restricciones, mientras que si $\Omega \subset \Psi$ se dice que se tiene un problema con restricciones. La función objetivo f es tal que:

$$f : \Psi \to R$$
$$X \ \alpha \ f(X)$$

Existen métodos exactos para resolver problemas de optimización, y como el nombre mismo lo dice, son métodos que nos permiten conocer la solución óptima, es decir, la solución exacta del problema de optimización. La función objetivo y sobre todo el tamaño del espacio de búsqueda pueden complicar el encontrar el óptimo de un problema de optimización. Al ejecutar un algoritmo que está intentando encontrar la solución de un problema de optimización, los recursos computacionales actuales, dígase principalmente memoria principal y el procesador, son también un factor importante que puede complicar muchísimo el encontrar el óptimo al menos en un tiempo prudencial [4] [5].

Las heurísticas son métodos inexactos para problemas de optimización, es decir, se trata de métodos que no ofrecen ningún tipo de garantía para encontrar la solución exacta, pero en la práctica son capaces de ofrecer muy buenas soluciones, es decir, soluciones viables en el mundo real que están *cerca* de la solución óptima del problema, y que esa solución cercana y buena la pueden encontrar en un tiempo computacional aceptable. Lo dicho anteriormente es la principal razón por la cual las heurísticas son muy populares y son la técnica preferida en muchos problemas de optimización que son muy complicados [4] [5]. En resumen, la heurística es un método que desea encontrar buenas soluciones (preferiblemente las óptimas) en buenos tiempos de ejecución.

El término heurística se usa también para referirse a algoritmos que resuelven problemas de propósito particular. Si se tiene un tipo de problema de optimización con ciertas características que lo hacen especial, distinto, diferente, de otros tipos de problemas de optimización, se puede diseñar una heurística, es decir, un algoritmo que resuelve ese tipo de problema puntual de optimización. Por ejemplo, es muy conocido el algoritmo de Dijkstra para encontrar el camino más corto entre dos vértices de un grafo.

Por el contrario, una metaheurística es un algoritmo que resuelve problemas generales de optimización, es decir, se le puede aplicar una misma metaheurística a muchos tipos de problemas de optimización sin importar lo diferente que sean. Una metaheurística se la puede ver también como una heurística para heurísticas, es decir, cuando se emplea una metaheurística para resolver un problema de optimización en particular, la metaheurística como parte de las tareas que debe realizar, se apoyará en heurísticas de propósito particular, para resolver el problema de optimización [4] [5].

Existen muchas metaheurísticas muy conocidas, entre ellas: el algoritmo genético, la búsqueda tabú, el recocido simulado, GRASP, que pueden utilizarse, junto con heurísticas, para resolver problemas de optimización. Scatter search es también un método metaheurístico que se puede emplear para resolver problemas de optimización y forma parte de la familia de los algoritmos evolutivos. Tiene sus orígenes en los años setenta, pero es en la última década cuando ha sido probado en muchos problemas con un alto grado de dificultad y con excelentes resultados [4] [5].

Un objetivo importante de este documento de tesis es el de explicar cada una de las fases de *scatter search* o en español *búsqueda dispersa*, es decir, indicar en detalle cómo trabaja internamente el algoritmo. Para tal efecto, se ha construido un software que resuelve el problema de ruteo de vehículos utilizando la metaheurística de *scatter search*. A medida que se explique cómo internamente hace su trabajo el algoritmo de búsqueda dispersa, se indicará también en qué momento utiliza heurísticas de propósito particular, y cuál es el trabajo que estas heurísticas realizan.

1.7 SISTEMAS DE INFORMACIÓN WEB

Un sistema de información web es un software distribuido que opera detrás de la interacción entre un web browser y un web server. El servidor web es un proceso servidor que administra un repositorio de documentos HTML, también conocidos como páginas web. El web browser es un proceso cliente que desea tener acceso a dichas páginas web y envía solicitudes al servidor web por ellas. Un web browser sin embargo, puede también enviar requerimientos a un servidor web para que ejecute programas. Un sistema de información web está constituido por todos aquellos programas que puede hacer ejecutar un servidor web debido a peticiones hechas por un web browser.

Figura # 1.3 "Interacción Web Browser y Web Server"

La forma de middleware que utilizan el web browser y el web server para comunicarse es el muy conocido protocolo de comunicación http, uno de los tantos protocolos de comunicación de la familia de protocolos TCP/IP. Internet, la gran red de redes, opera gracias a esta familia de protocolos. Por otro lado, HTML es un lenguaje que sirve para la creación de documentos web y es por este motivo que a las páginas web se las llama también documentos HTML.

Lenguajes de programación muy populares para desarrollar sistemas de información web son C# y Java. Otras tecnologías importantes para construir sistemas de información son AJAX, XML, JSON, entre otras. Cuando termina la ejecución de unos de los programas que *ejecuta* del lado del servidor web, la salida es normalmente un documento HTML que el servidor web debe enviar al web browser. Obviamente el web browser tiene la capacidad de interpretar el lenguaje HTML y finalmente le puede mostrar la página web al usuario. Entonces, durante la ejecución de un sistema de información web, los tomadores de decisiones de la empresa o los trabajadores operativos, lo que hacen es *navegar* entre varios documentos web que han sido generados en tiempo de ejecución (dinámicamente), como producto de las peticiones que el web browser le envía al web server por la ejecución de estos programas especiales.

Los sistemas de información actuales son distribuidos y ejecutan en entornos web. Las empresas se han dado cuenta de lo importante que es contar con aplicaciones de negocios que tengan capacidades de integración y que utilicen el Internet para coordinar sus actividades y tareas con otras aplicaciones que pueden estar ubicadas en cualquier parte del mundo. Además de esta independencia de ubicación es importante hacer notar que la independencia de plataforma es una característica deseada y es una realidad también, ya que lo vemos en Internet todos los días. Es decir, no importa las características del hardware ni el sistema operativo bajo el cual ejecutan los sistemas de información, la comunicación y el intercambio de información es siempre posible. Finalmente en la figura 1.3 se ilustra toda la explicación dada anteriormente sobre la interacción que ocurre en el web entre el cliente y el servidor.

CAPÍTULO 2

BÚSQUEDA DISPERSA PARA EL RUTEO DE VEHÍCULOS

En este capítulo se discute la utilización de la metaheurística *búsqueda dispersa* (scatter search en inglés) para resolver puntualmente el problema del PRVC, cuyo modelo matemático se encuentra en la sección 1.5 en la modalidad simétrica. Primeramente se va a explicar cómo trabaja la búsqueda dispersa para posteriormente indicar detalladamente cómo se la usó (diseño e implementación) para el problema del PRV simétrico capacitado. Durante el análisis se tratan los temas de convergencia a óptimos locales y globales, como también el consumo de recursos computacionales.

2.1 LA METAHEURÍSTICA DE BÚSQUEDA DISPERSA

La metaheurística de búsqueda dispersa se divide de manera general en cuatro etapas y utiliza dos conjuntos (P y R) cuando intenta resolver el problema de la sección 1.6 de optimización. Las cuatro etapas son las siguientes:

- Primera etapa: construcción del conjunto P
- Segunda etapa: construcción del conjunto R
- Tercera etapa: formación de grupos, combinación y selección
- Cuarta etapa: actualización del conjunto R

Se va a indicar de manera breve qué ocurre en cada una de las etapas. La metodología de *búsqueda dispersa* proporciona sugerencias y lineamientos generales cuando se pretende resolver un problema de optimización, y el objetivo en esta sección 2.1 es explicar esa filosofía de trabajo. En todas las cuatro etapas son muy importantes dos términos: diversidad y calidad de las soluciones. Al final de la ejecución del algoritmo, tomamos como la solución del problema el mejor elemento del conjunto R en la iteración número n. Una vez explicadas de forma resumida cada una de las etapas, se muestra un pequeño pseudocódigo de la metaheurística de scatter search.

2.1.1 CONSTRUCCIÓN DEL CONJUNTO P

El conjunto P tiene un tamaño definido a través de la variable $PSize$. Debemos tomar entonces $PSize$ elementos del conjunto Ω para *llenar* el conjunto P. ¿Cuáles elementos tomar del conjunto Ω?

El primer paso consiste en intentar tomar $PSize$ elementos de Ω de forma tal que se pueda *barrer o cubrir* todo el conjunto Ω, es decir, la idea es que haya variedad en estos $PSize$ elementos que se escogerán del conjunto Ω, que es el conjunto que contiene a todas las soluciones factibles del problema de optimización (sección 1.6). Una vez hecho esto, por cada elemento que hayamos tomado debemos estudiar su vecindad mediante una búsqueda local para saber si *cerca* de dicho elemento existe otro elemento *mejor* en términos de la función objetivo f. Si tal elemento mejor existe, entonces reemplazará al original. Podemos notar entonces que en esta primera búsqueda de elementos existe una mezcla adecuada de diversidad y calidad en los elementos que se toman del conjunto Ω.

2.1.2 CONSTRUCCIÓN DEL CONJUNTO R

El conjunto R tiene un tamaño definido por la variable $RSize$. En [4] [5] se recomienda que la relación entre los tamaños de los conjuntos P y R sea $PSize = 5RSize$, pero sólo es una sugerencia no una imposición. ¿Cómo llenar el conjunto R?

El conjunto R es construido a partir del conjunto P. Usualmente para construir el conjunto R nos preocupamos que la variable $RSize$, que define el tamaño de R, sea par, ya que la mitad de los elementos de R serán escogidos por calidad mientras que la otra mitad por diversificación, como ya se dijo, haciendo uso del conjunto P. Aquí es importante indicar que se puede parametrizar esto, es decir, que r_1 elementos sean escogidos por calidad y r_2 por diversidad teniendo presente que $r_1 + r_2 = RSize$. En general sin embargo, se toman entonces los $RSize/2$ *mejores* elementos del conjunto P y se incorporan al conjunto R. ¿Cómo escoger los elementos restantes?

De los elementos que quedan de P se deben tomar $RSize/2$ elementos más para completar el conjunto R, ¿cómo se lo hace para garantizar diversidad? Se calcula la distancia de cada elemento de P (ya no se consideran aquellos que se llevan a R) al

conjunto R con la fórmula: Sea $v \in P$, luego $d(v,R) = \underset{w \in R}{MIN} \| v - w \|$ y el elemento que se almacena en R es aquel elemento de P que dista más de R, es decir, el más lejano al conjunto R. La norma utilizada para calcular la distancia mencionada varía según el problema de optimización a resolver. En este momento, finalmente, se tiene ya construido totalmente el conjunto R con una mezcla muy buena y adecuada de calidad y diversidad. La figura 2.1 nos muestra el esquema general de la búsqueda dispersa.

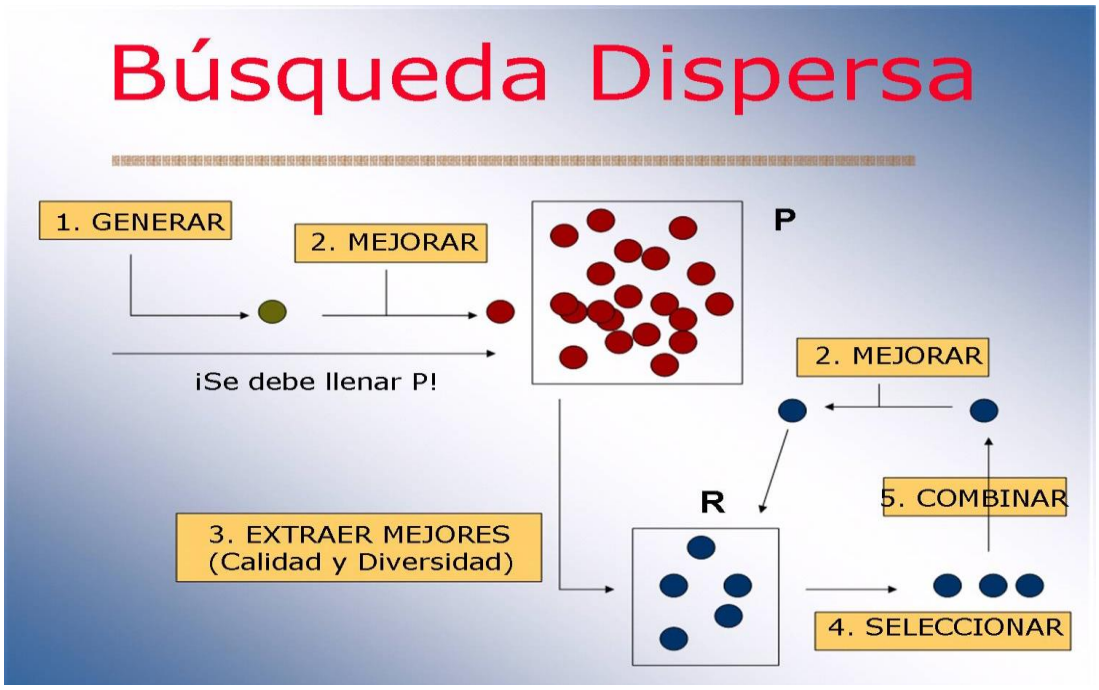

Figura # 2.1 ***Búsqueda Dispersa***

2.1.3 FORMACIÓN DE GRUPOS, COMBINACIÓN Y SELECCIÓN

Esta etapa es muy importante porque es la que se realiza de manera iterativa junto con la actualización del conjunto R. La formación de grupos consiste en tomar elementos del conjunto R para formar grupos de dos, tres o más elementos según el diseño que se quiera seguir, considerando que no haya repeticiones y que todos los elementos del conjunto R sean considerados. Una vez formados los grupos se tienen subconjuntos del conjunto R. Por cada grupo se efectúan combinaciones entre los elementos del grupo con la intención de generar nuevos elementos. En este punto se tienen entonces, además de los elementos del conjunto R, elementos adicionales que han sido generados como resultado de las combinaciones. De entre todos estos elementos, los del conjunto R y

los nuevos, se procede a seleccionar $RSize$ elementos que son los que formarán el nuevo conjunto R.

2.1.4 ACTUALIZACIÓN DEL CONJUNTO R

Una vez generados los nuevos elementos en la etapa anterior, y teniendo presente también a los elementos del conjunto R, la pregunta es ¿cómo actualizar el conjunto R? De entre todos los elementos tomamos r_1 elementos por calidad y los otros r_2 elementos pueden ser escogidos de forma aleatoria teniendo presente que $r_1 + r_2 = RSize$. Se pueden tomar también los $RSize$ mejores elementos o aplicar algún criterio de diversidad.

En cada una de las etapas se pueden hacer variaciones y tener distintas implementaciones que se pueden controlar mediante parámetros cuando se está diseñando el algoritmo de búsqueda dispersa. Cuando se termina la implementación del algoritmo en algún lenguaje de programación, es importante efectuar una calibración de los parámetros de entrada. Para el caso de búsqueda dispersa estos parámetros son normalmente $PSize$, $RSize$, n, r_1, r_2, entre otros. Los parámetros se ajustan mediante pruebas de *ensayo y error*, es decir, se prueban diferentes valores de los parámetros en ejecuciones del algoritmo hasta que se considere que se ha encontrado la estabilidad del algoritmo, tanto en los resultados que entrega como en el tiempo de ejecución.

2.1.5 PSEUDOCÓDIGO DE BÚSQUEDA DISPERSA

Se muestra un resumido pseudocódigo del algoritmo de búsqueda dispersa:

```
Sea n el número de iteraciones.
Construir el conjunto P
Construir el conjunto R
Para cada i desde 1 hasta n
     Formación de grupos, combinación y selección
     Actualización del conjunto R
Fin del "Para cada"
```

El número de iteraciones n, es decir, el número de veces que se actualiza el conjunto R, es también un parámetro del algoritmo. Tomamos como la *mejor solución encontrada* para el problema, el mejor elemento del conjunto R en la última iteración del algoritmo. Algo importante de mencionar es que al usar cualquier algoritmo metaheurístico estamos concientes que no existe garantía de encontrar el óptimo del problema, pero podemos encontrar una *buena solución* en un tiempo prudencial y con un costo computacional aceptable.

La metodología que propone búsqueda dispersa no deja todo al azar como ocurre en algunas implementaciones de algoritmos genéticos o de otras metaheurísticas, la aleatoriedad se uso de manera criteriosa. Existen reglas que guían la búsqueda de forma que se pueda explorar el espacio de soluciones eficientemente incorporando además mecanismos que intentan evitar caer en óptimos locales. Es decir, se trata de un procedimiento de búsqueda sistemático con moderados tintes aleatorios [4] [5]. Sin embargo, se reitera, siempre existe la posibilidad que el algoritmo se quede *atrapado* en un óptimo local pero se pueden incluir estrategias en el diseño para que eso no ocurra y se de la convergencia hacia el óptimo global del problema.

El análisis de convergencia para cualquier metaheurística es sumamente complicado debido a lo no determinista de su comportamiento. Si ejecutamos una metaheurística una y otra vez, con el mismo problema y con los mismos parámetros de entrada, se van a obtener resultados diferentes. El caer en un óptimo local es algo que puede ocurrir en general con cualquier metaheurística, no sólo con búsqueda dispersa [4] [5].

2.2 DISEÑO E IMPLEMENTACIÓN DEL ALGORITMO

En esta sección se explicará cómo se diseñó y se implementó la búsqueda dispersa para resolver el problema de ruteo de vehículos capacitado simétrico. Cabe indicar que se utilizó el lenguaje de programación C# y la tecnología .NET para programar el algoritmo de la búsqueda dispersa.

Se va a suponer que se tienen n puntos ubicados en una ciudad. Los números desde 1 hasta n indicarán estas ubicaciones. El punto dentro de la ciudad desde donde sale el vehículo para iniciar el recorrido, y donde debe finalmente regresar el vehículo una vez terminado el recorrido, lo vamos a representar con el 0. Se tienen entonces en total $n+1$ puntos. Para hacer la implementación lo más real posible, se han tomado $n+1$ de la ciudad de Guayaquil. Tenemos entonces un grafo no dirigido donde los nodos van a representar los puntos distribuidos por la ciudad. El peso en las aristas va a representar la distancia en kilómetros que existe entre los nodos. Se desea entonces, partiendo del vértice 0, conocer el orden en el cual se deben ir visitando cada uno de los n nodos, de forma tal que la distancia recorrida sea la mínima y considerando también que el vehículo debe finalizar el recorrido en el vértice 0 y que cada nodo se debe visitar una vez y sólo una. Este problema se conoce también como *el problema del agente viajero simétrico* y el número de soluciones factibles es $n!$ lo que hace que se trate de un problema difícil de resolver para valores de n muy grandes. Es más, dentro de la teoría de la complejidad computacional para resolver problemas de optimización combinatoria es considerado un problema NP duro.

2.2.1 COORDENADAS Y DISTANCIAS ENTRE NODOS

Cada nodo es representado por dos pares ordenados. El primer par ordenado representa al nodo en coordenadas geográficas. En estas coordenadas, el primer componente del par es la *longitud* y el segundo componente del par es la *latitud*. Este sistema de coordenadas ve a los puntos como realmente están ubicados, es decir, como puntos en una esfera (el planeta tierra). ¿Por qué esta representación? Para tener una idea exacta de la ubicación del punto en la ciudad de Guayaquil se ingresó a *Google Earth* y se obtuvo de forma muy precisa la longitud y la latitud de cada uno de los puntos que se podían considerar para un ruteo. Este sitio web muy popular proporciona las

coordenadas de los puntos en el globo terrestre usando el sistema de coordenadas geográficas. Pero para calcular las distancias entre los nodos fue necesario convertir de coordenadas geográficas a coordenadas UTM (Universal Transverse Mercator), ya que este otro sistema de coordenadas, por el contrario, ubica los puntos en un plano que es la figura geométrica más adecuada para medir distancias entre los nodos. Se muestra a continuación el método que hace la transformación de coordenadas geográficas a coordenadas UTM:

```
public double[] ConvertirGEO2UTM(double Longitud, double Latitud)
{
  double[] XY = new double[2];

  const double a = 6378388;
  const double b = 6356911.94613;
  double c = (a * a) / b;
  double e = Math.Sqrt(a * a - b * b) / b;
  int Huso = (int)Math.Floor((Longitud/6)+31);
  int LambdaInicial = Huso * 6 - 183;
  double DeltaLambda = (Longitud-LambdaInicial) * Math.PI / 180 ;
  double A = Math.Cos((Latitud * Math.PI) / 180) *
             Math.Sin(DeltaLambda);
  double Epsilon = 0.5 * Math.Log((1 + A) / (1 - A), Math.E);
  double Eta =
Math.Atan((Math.Tan((Latitud*Math.PI)/180))/Math.Cos(DeltaLambda))-
(Latitud*Math.PI)/180;
  double v = (c * 0.9996) /
Math.Sqrt((1 + e * e * Math.Pow(Math.Cos((Latitud * Math.PI) / 180),
2)));
  double si = e * e * 0.5 * Epsilon * Epsilon *
              Math.Pow(Math.Cos((Latitud * Math.PI) / 180), 2);
  double A1 = Math.Sin((2 * Latitud * Math.PI) / 180);
  double A2 = A1 * Math.Pow(Math.Cos((Latitud * Math.PI) / 180), 2);
  double J2 = ((Latitud * Math.PI) / 180) + 0.5 * A1;
  double J4 = 0.75 * J2 + 0.25 * A2;
  double J6 = (5 * J4 + A2 * Math.Pow(Math.Cos((Latitud * Math.PI) /
              180), 2)) / 3;
  double Alfa = 0.75 * e * e;
  double Beta = (5 * Alfa * Alfa) / 3;
  double Gamma = (35 * Alfa * Alfa * Alfa) / 27;
  double B0 = 0.9996 * c * (((Latitud * Math.PI) / 180) - Alfa * J2 +
              Beta * J4 - Gamma * J6);

  XY[0] = Epsilon * v * (1 + si / 3) + 500000;
  XY[1] = Eta * v * (1 + si) + B0 + 10000000;
  return XY;
}
```

Una vez que se tienen a la mano las coordenadas UTM, podemos usar una métrica para calcular los pesos de las aristas. Para la implementación se utilizaron dos métricas: la

métrica euclidiana y la *métrica del taxi* también conocida como *métrica de Manhattan*. A continuación se muestra la implementación:

```
public double Costo(DataRow NodoA, DataRow NodoB)
{
  double[] PuntoA =
this.ConvertirGEO2UTM((double)NodoA["CliLongitud"],
                      (double)NodoA["CliLatitud"]);
  double[] PuntoB =
this.ConvertirGEO2UTM((double)NodoB["CliLongitud"],
                      (double)NodoB["CliLatitud"]);

  if (this.Metrica == "E")
  {
    return Math.Sqrt(Math.Pow(PuntoA[0]-
           PuntoB[0], 2)+Math.Pow(PuntoA[1]-PuntoB[1], 2));
  }
  if (this.Metrica == "M")
  {
    return (double)(Math.Abs(PuntoA[0]-PuntoB[0])+
                    Math.Abs(PuntoA[1]-PuntoB[1]));
  }
  return 0;
}
```

El nombre del método es costo porque es el peso de cada arista, en otras palabras, representa el costo que se debe pagar para moverse entre nodos. Esto a su vez muestra lo flexible de la implementación. Si posteriormente se desea trabajar con una nueva métrica, simplemente se la implementa en el interior del método *Costo* sin afectar los demás métodos que componen el software.

2.2.2 DISTANCIA ENTRE SOLUCIONES FACTIBLES

Sean s_1 y s_2 dos soluciones factibles. Es decir, estas dos soluciones son permutaciones con los números de 1 hasta n. Suponga que s_1 y s_2 son de la forma:

$$s_1 = \left(\begin{smallmatrix}1\\ s_1\end{smallmatrix}\right)\left(\begin{smallmatrix}2\\ s_1\end{smallmatrix}\right)\left(\begin{smallmatrix}3\\ s_1\end{smallmatrix}\right)\Lambda\left(\begin{smallmatrix}n-1\\ s_1\end{smallmatrix}\right)\left(\begin{smallmatrix}n\\ s_1\end{smallmatrix}\right)$$

$$s_2 = \left(\begin{smallmatrix}1\\ s_2\end{smallmatrix}\right)\left(\begin{smallmatrix}2\\ s_2\end{smallmatrix}\right)\left(\begin{smallmatrix}3\\ s_2\end{smallmatrix}\right)\Lambda\left(\begin{smallmatrix}n-1\\ s_2\end{smallmatrix}\right)\left(\begin{smallmatrix}n\\ s_2\end{smallmatrix}\right)$$

En esta representación $\left(s_1^i\right)$ es el $i-ésimo$ componente de la permutación s_1 mientras que $\left(s_2^i\right)$ es el $i-ésimo$ componente de la permutación s_2 para $i=1,2,3,\Lambda\ ,n$.

Para la implementación que se propone en este documento de tesis se calcula la distancia entre s_1 y s_2 empleando el siguiente pseudocódigo:

```
Distancia = 0
Para cada i desde 1 hasta n
     Si (s1^i) ≠ (s2^i) entonces
          Distancia = Distancia + 1
     Fin del "Si"
Fin del "Para cada"
```

La variable `Distancia` es la que tendrá almacenada la *distancia* entre las dos soluciones factibles una vez finalizado el pseudocódigo. Se trata de una implementación sencilla. Existen otras implementaciones mucho más sofisticadas y por ende más complicadas de implementar. Una de ellas, por citar una, consiste en calcular el mínimo número de *cambios* que debe aplicarse a la solución s_1 para que se convierta en la solución s_2. Entiéndase por un *cambio* el intercambio de dos elementos consecutivos de la permutación cuando se escriben sus componentes de manera circular. Al final, la distancia es igual al número de cambios efectuados. Pero, ¿para qué necesitamos calcular la distancia entre dos soluciones factibles del problema?

En las secciones 2.1.2 y 2.1.4 que hablan sobre la construcción y la actualización, respectivamente, del conjunto R, para calcular los r_2 número de elementos que por diversidad alimentarán al nuevo conjunto R, se deben tomar aquellos elementos que se encuentre más lejos del actual conjunto R. Recordar que para calcular la distancia de un elemento al conjunto R, se calculan todas las distancias de dicho elemento a cada uno de los elementos que se encuentran en ese momento en R y, la más pequeña de esas distancias, se toma como la distancia del elemento en mención al conjunto R.

2.2.3 MÉTODO DE COMBINACIÓN

Para la implementación que se propone se formaron grupos de dos integrantes, es decir, parejas de elementos del conjunto R. Cada pareja va a generar dos nuevos elementos. Recordar que por cada iteración de scatter search se necesitan *nuevos elementos* con la intención de acercarnos cada vez al óptimo global del problema.

A los integrantes de la pareja se los llama indistintamente *Padre* y *Madre*. A los nuevos elementos *Hijo* e *Hija*. Para ilustrar el funcionamiento del método de combinación se usa ejemplo con $n = 9$. Suponga que *Padre* y *Madre* son:

Padre

8	***5***	***1***	***4***	***9***	***3***	***6***	***2***	***7***

Madre

3	***6***	***9***	***2***	***5***	***1***	***7***	***4***	***8***

El primer paso consiste en generar aleatoriamente un número k que representará una misma posición tanto para el *Padre* como para la *Madre*. Supongamos que $k = 3$. Entonces, se da un primer paso en la construcción de *Hijo* e *Hija* de la siguiente manera:

Hijo

3	***6***	***9***	**-**	**-**	**-**	**-**	**-**	**-**

Hija

8	***5***	***1***	**-**	**-**	**-**	**-**	**-**	**-**

Observe que *Hijo* tiene los k primeros componentes de la *Madre* y que *Hija* tiene los k primeros componentes del padre. Al generar aleatoriamente el número k se tiene presente que dicha variable debería como mínimo tomar el valor de 3 y como máximo el valor de $n - 3$.

Debido a que los k primeros componentes del *Hijo* fueron tomados de la *Madre*, ahora los $n-k$ restantes componentes del Hijo serán tomados del Padre, teniendo presente el no repetir los valores de los componentes y haciendo la lectura de los componentes del *Padre* de izquierda a derecha, de la siguiente manera:

Hijo

3	***6***	***9***	***8***	***5***	***1***	***4***	***2***	***7***

Para la *Hija* se trabaja de forma análoga, es decir, debido a que los k primeros componentes fueron tomados del *Padre*, los $n-k$ componentes restantes son tomados de la *Madre* haciendo la lectura de izquierda a derecha y sin repetir los componentes. Bajo este criterio, la *Hija* nos que de la siguiente forma:

Hija

8	***5***	***1***	***3***	***6***	***9***	***2***	***7***	***4***

El método de combinación juega un papel sumamente importante dentro de scatter search, ya que es la parte del algoritmo que se encarga de, usando elementos actuales, construir nuevas soluciones factibles que van a *entrar en el juego*. Puede ocurrir que al combinar dos elementos *buenos* obtengamos como resultado uno que sea *malo*, o que al combinar dos *malos* podamos generar uno bueno. Es la mezcla de calidad y diversidad lo que hará que el algoritmo presente convergencia al óptimo global después de un número determinado de iteraciones, por lo menos es lo que se espera. Recordemos que el empleo de heurísticas y metaheurísticas es simplemente una cuestión de fe.

Dado que el conjunto R tiene $RSize$ elementos, una alternativa es formar $RSize/2$ parejas sin repetir elementos de R. Si por cada pareja se construyen 2 elementos, entonces se tendrán en total $RSize$ elementos nuevos. Es decir que para la actividad de *selección* se tienen $2\,RSize$ elementos que serán considerados para que formen parte del nuevo conjunto R. Cabe indicar que se pueden hacer implementaciones donde se formen más parejas considerando repeticiones. La implementación que corresponde a esta tesis se apega al esquema sin repeticiones.

2.2.4 LA FUNCIÓN OBJETIVO

Sean s_1 y s_2 dos soluciones factibles del problema. Para la implementación de esta tesis, se dice que s_1 es una mejor solución que s_2 si $f(s_1) < f(s_2)$, donde f es la función objetivo, debido a que se trata de un problema de minimización.
Sea s una solución factible del problema, entonces s tiene la forma:

$$s = \left(\overset{1}{s}\right)\left(\overset{2}{s}\right)\left(\overset{3}{s}\right)\Lambda\left(\overset{n-1}{s}\right)\left(\overset{n}{s}\right)$$

El $i-ésimo$ componente de la permutación s tiene la notación $\left(\overset{i}{s}\right)$. Eso quiere decir que $\left(\overset{i}{s}\right)$ representa el nodo número i que será visitado por el vehículo. Se define entonces la función objetivo del problema como:

$$f(s) = C\left(\left(\overset{o}{s}\right),\left(\overset{1}{s}\right)\right) + C\left(\left(\overset{1}{s}\right),\left(\overset{2}{s}\right)\right) + C\left(\left(\overset{2}{s}\right),\left(\overset{3}{s}\right)\right) + \Lambda + C\left(\left(\overset{n-1}{s}\right),\left(\overset{n}{s}\right)\right) + C\left(\left(\overset{n}{s}\right),\left(\overset{0}{s}\right)\right)$$

Es decir:

$$f(s) = C\left(\left(\overset{n}{s}\right),\left(\overset{0}{s}\right)\right) + \sum_{i=0}^{n-1} C\left(\left(\overset{i}{s}\right),\left(\overset{i+1}{s}\right)\right)$$

La función $C(\bullet,\bullet)$ es la que entrega la distancia recorrida entre los nodos que recibe como argumentos. Note en la función objetivo que también consideramos la distancia del nodo desde donde parte el vehículo (nodo etiquetado con el 0) hasta el nodo 1 (primer nodo que visita el vehículo), y además la distancia del último nodo visitado (nodo n) al nodo 0. Se considera que el vehículo inicia y termina su recorrido en el mismo punto, es decir, el nodo 0.

La implementación en C# es la siguiente:

```
public double EvaluarElemento(DataRow[] Elemento)
{
  double CostoTotal = 0;
  for(int i = 0; i < Elemento.Length - 1; i++)
  {
    CostoTotal += this.Costo(Elemento[i], Elemento[i + 1]);
  }
  CostoTotal += this.Costo(Elemento[0],
                           Elemento[Elemento.Length - 1]);
  return CostoTotal;
}
```

2.2.5 LA BÚSQUEDA LOCAL

Dada una solución s factible del problema, el objetivo de la búsqueda local es estudiar la vecindad de s, con notación $N(s)$, para buscar si existe un elemento $s' \in N(s)$ tal que $f(s') < f(s)$. En otras palabras, buscamos *cerca* de la solución s si existe otra solución s' que sea mejor que s. La búsqueda local normalmente se implementa en un método llamado *método de mejora*, pues el objetivo es dada una solución, intentar reemplazarla por otra solución que sea mejor y que se encuentre algo *cerca* de la original. El término *cerca* depende del problema.

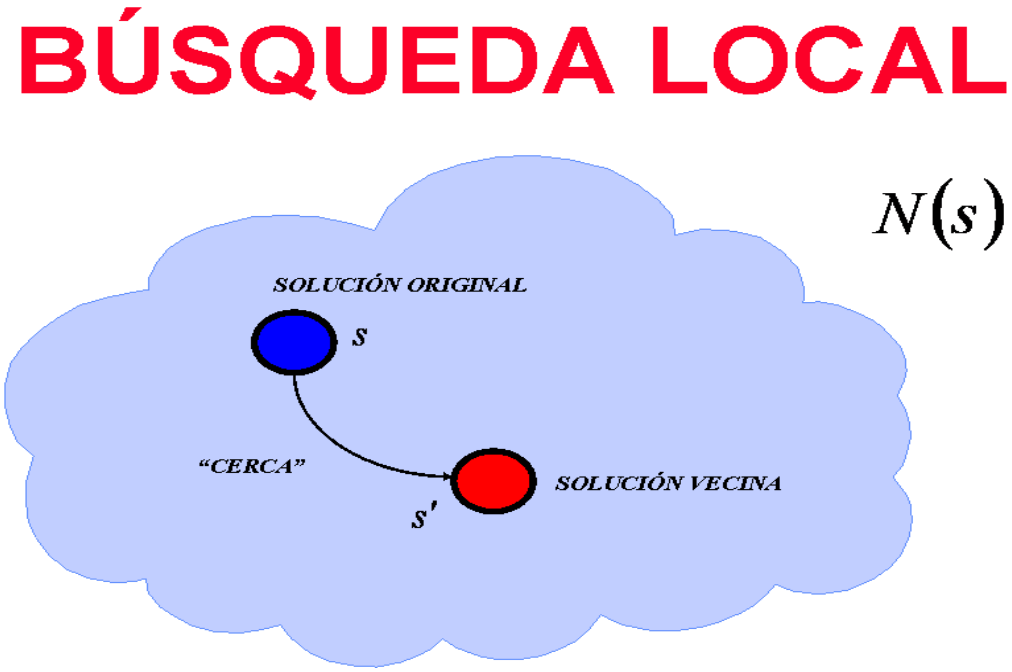

Figura # 2.2 *Búsqueda Local*

Para la implementación de esta tesis ilustraremos la búsqueda local implementada con un ejemplo. Supongamos que la solución original s es la siguiente:

s

8	***5***	***1***	***4***	***9***	***3***	***6***	***2***	***7***

El primer paso consiste en generar dos números aleatorios distintos entre 1 y n. Supongamos que dichos números son 2 y 6. Luego, debemos intercambiar los nodos que se encuentran en dichas posiciones. El resultado sería el vecino s', el cual es:

s'

8	***3***	***1***	***4***	***9***	***5***	***6***	***2***	***7***

De esta manera se ha obtenido un nuevo elemento simplemente intercambiando un par de componentes en el elemento original. Esta es la manera de construir un *vecino*. Si calculamos la distancia entre estos elementos el resultado es de dos unidades ya que ambas soluciones difieren, correspondientemente, en dos nodos. Es el valor más pequeño, según 2.2.2 para la distancia entre dos soluciones factibles, lo que nos dice que s y s' están muy cerca.

En el siguiente paso, utilizando la función objetivo de 2.2.4, debemos analizar si s' es mejor que s o no lo es. Si s' es mejor que s ahora centramos nuestra atención en s', caso contrario seguiremos enfocados en s. Mediante el parámetro *NumIntentos* del algoritmo de búsqueda dispersa podemos establecer, por cada búsqueda local, el número de intentos que haremos para encontrar un mejor vecino. Aquí se pueden tener dos implementaciones. La una consiste en, apenas se encuentra un vecino mejor, el método detiene la búsqueda aún cuando no se ha alcanzado el número de intentos establecido en el parámetro *NumIntentos*. La otra implementación consiste en *gastarnos* todos los intentos posibles y buscar el máximo número de veces que está permitido. Para la implementación de esta tesis se usó la segunda modalidad. La figura 2.2 nos ilustra la explicación efectuada sobre la búsqueda local que ocurre en búsqueda dispersa.

2.2.6 LA HEURÍSTICA DEL VECINO MÁS CERCANO

En la sección 2.1.1 se explicó en líneas generales como construir el conjunto P. En esta sección explicaremos en detalle la implementación que se ha seguido específicamente para llenar el conjunto P del algoritmo que se propone en esta tesis. Recordemos que el conjunto P debe tener un número de elementos igual a $PSize$.

El primer paso consiste en utilizar la heurística del *vecino más cercano*. El propósito es construir buenas soluciones en P, ya que esta heurística se basa en una estrategia *voraz* para efectuar su trabajo. Usando el *vecino más cercano* vamos a generar n buenas soluciones de la siguiente manera: La primera solución tendrá al 1 como primer nodo, el segundo nodo será aquel de los nodos restantes que se encuentre más cerca del nodo 1, el tercer nodo será aquel de los nodos restantes que se encuentre más cerca del segundo nodo, el cuarto nodo será aquel de los nodos restantes que se encuentre más cerca del tercer nodo y, así sucesivamente, hasta finalizar la construcción de la solución factible al problema. Esto debe repetirse $n-1$ veces más para generar las $n-1$ restantes soluciones factibles comenzando, respectivamente, con los nodos $2,3,\Lambda\ ,n$.

El segundo paso consiste en efectuar búsquedas locales a estos n elementos (es decir, analizar sus vecindades) para mejorarlos aún más. El tercer paso consiste en generar aleatoriamente las $PSize-n$ soluciones que faltan para finalizar la construcción del conjunto P. Para que la generación se haga teniendo presente la diversidad o variedad en las soluciones, se generan una misma cantidad de soluciones factibles que comienzan con el nodo 1 y otras que terminan con el nodo 1, luego igual cantidad de soluciones que comienzan con el nodo 2 y otras que terminan con el nodo 2, y así se continúa hasta generar una misma cantidad de soluciones que comienzan con el nodo n y otras que terminan con el nodo n. De esta manera se generan los $PSize-n$ elementos de P que nos hacían falta. Pero no olvidar que estos elementos no pasan directamente al conjunto P, ya que previamente, como cuarto y último paso, se les aplica el método de mejora mediante las búsquedas locales. A pesar de que en esta etapa del algoritmo se pueden realizar varias implementaciones, la implementación que se ha adoptado es la que se ha explicado.

2.2.7 MÉTODO PRINCIPAL DE BÚSQUEDA DISPERSA

A continuación se va a mostrar el método principal llamado *SS_VRP* que gobierna el comportamiento y la operación general del algoritmo de scatter search que se implementó en esta tesis. Todas las etapas o fases del algoritmo que se han explicado en secciones anteriores coordinan y se enlazan en el método principal. A continuación el método principal del algoritmo de búsqueda dispersa:

```
public double[] SS_VRP(out double MejorValor)
{
  ArrayList Pool = new ArrayList();
  ArrayList EvalPool = new ArrayList();
  DataRow[][] P = new double[this.PSize];
  double[] EvalP = new double[this.PSize];
  DataRow[][] R = new double[this.RSize];
  double[]EvalR = new double[this.RSize];

  double[] MejoresValores = new double[this.NumIteraciones];
  this.GenerarP(P, EvalP);
  this.ConstruirR(P, EvalP, R, EvalR);

  ArrayList SubSets = new ArrayList();
  int Iteraciones = 0;
  while (true)
  {
      Pool.Clear();
      EvalPool.Clear();
      this.GenerarSubSets(SubSets, R);
      while (SubSets.Count != 0)
      {
        double[][] SubSet = (double[][])SubSets[0];
        double[][] TrialSolutions = this.Combinar(SubSet);
        double ValorHijo = this.LS(TrialSolutions[0], 100);
        Pool.Add(TrialSolutions[0]);
        EvalPool.Add(ValorHijo);
        double ValorHija = this.LS(TrialSolutions[1], 100);
        Pool.Add(TrialSolutions[1]);
        EvalPool.Add(ValorHija);
        SubSets.Clear();
      }
      this.ActualizarR(R, EvalR, Pool, EvalPool);
      MejoresValores[Iteraciones] = EvalR[0];
      Iteraciones++;
      if (Iteraciones == this.NumIteraciones) break;
  }
  MejorValor = EvalR[0];
  return R[0];
}
```

CAPÍTULO 3

SOFTWARE DSS Y EXPERIMENTOS COMPUTACIONALES

En este capítulo se presenta el software que se ha construido para esta tesis. Se trata de un sistema DSS para el ruteo de vehículos que opera en el web, que es de naturaleza distribuida, y que es principalmente un sistema de información logística. Puede ser utilizado, por ejemplo, para recoger insumos de los proveedores o para distribuir bienes a los clientes. Se finaliza el capítulo mostrando algunos experimentos computacionales.

3.1 INICIO DEL SOFTWARE

Al tratarse de una aplicación web es necesario *instalar* el sistema en algún servidor web. Para la ejecución y los experimentos computacionales se utilizó una computadora portátil que cuenta con un servidor web para las pruebas. Una vez que iniciamos el sistema se puede observar la ventana principal, la cual se muestra a continuación:

Figura # 3.1 *Menú Principal Software DSS*

La ventana principal presenta dos opciones. La opción *Administración de Puntos para el Ruteo en Guayaquil* nos lleva a una parte del sistema donde se podrá ingresar, modificar o eliminar información de puntos ubicados en Guayaquil. La opción *Algoritmo Búsqueda Dispersa para el PRVC Simétrico* nos dirige a una parte del sistema desde donde se podrá invocar directamente al algoritmo implementado.

3.2 DISEÑO DE LA BASE DE DATOS

La base de datos que se emplea tiene tres tablas, las cuales se explicarán con todo detalle a continuación:

Tabla *Puntos*

Campo	**Tipo de Dato**
PuntoCod(CP)	Entero Largo Secuencial
PuntoDsc	Cadena de Caracteres [30]
PuntoDir	Cadena de Caracteres [50]
PuntoLongitud	Punto Flotante
PuntoLatitud	Punto Flotante

La tabla *Puntos* nos permite almacenar los distintos nodos de Guayaquil que pueden ser considerados para un ruteo en particular. El campo *PuntoCod* sirve como clave primaria de la tabla, es decir, se utiliza para identificar un registro de manera única en la tabla. Es un campo de tipo entero largo secuencial. El campo *PuntoDsc* permite almacenar una pequeña descripción para el punto, por ejemplo ESPOL o Biblioteca Municipal. El campo *PuntoDir* es para almacenar la dirección del punto, por ejemplo, para el caso de ESPOL se almacena como dirección KM 30.5 vía perimetral y para el caso de Biblioteca Municipal se almacena 10 de Agosto y Pedro Carbo como dirección. El máximo número de caracteres permitidos en *PuntoDsc* es de 20 y para el campo *PuntoDir* es de 50. Los campos *PuntoLongitud* y *PuntoLatitud* sirven para almacenar las coordenadas geográficas del punto. Recordar que en tiempo de ejecución estas coordenadas se convierten en coordenadas UTM, según la sección 2.2.1, además el grafo no dirigido para el ruteo también se construye dinámicamente (lo que obviamente incluye las distancias entre nodos, es decir, el peso de las aristas). Se podría considerar, sin embargo, almacenar las coordenadas UTM en la tabla *Puntos* para evitar el cálculo de transformación en futuras corridas.

Tabla *Rutas*

Campo	Tipo de Dato
RutaCod(CP)	Entero Largo Secuencial
RutaDsc	Cadena de Caracteres [30]

Tabla *RutasDetalle*

Campo	Tipo de Dato
RutaCod(CF)	Entero Largo Secuencial
PuntoCod	Cadena de Caracteres [30]

Si un ruteo en particular se desea ejecutar muchas veces, el sistema permite almacenar la información de los nodos del grafo usando las tablas *Rutas* y *Rutas Detalle* en una relación uno a varios. Por ejemplo, supongamos que para una corrida del algoritmo de búsqueda dispersa se desea trabajar con los siguientes puntos (es decir, con los siguientes nueve registros de la tabla *Puntos*):

PuntoCod	PuntoDsc
75	Mall del Sol
21	ESPOL
33	Hotel Ramada
49	Colegio Cristóbal Colón
5	Biblioteca Municipal
60	Policentro
17	Clínica Alborada
38	Puerto Marítimo
96	Colegio Espíritu Santo

A esta ruta de puntos la vamos a llamar *Ruta Alfa*. Entonces debemos agregar un registro en la tabla *Rutas*. El campo *RutaCod* es la clave primaria y es de tipo entero largo secuencial. El campo *RutaDsc* sirve para almacenar una breve descripción de la ruta, en este caso almacenamos *Ruta Alfa*. Luego, en la tabla *RutasDetalle* debemos almacenar tantos registros como nodos tenga la ruta. El campo *RutaCod* de la tabla *RutasDetalle* es la clave foránea que permite establecer la relación uno a varios entre las tablas *Rutas* y *RutasDetalle*. Esto tiene sentido ya que una ruta tiene varios nodos en

dicha ruta. El campo *PuntoCod* de la tabla *RutasDetalle* sirve para almacenar el código del nodo o punto que es parte de la ruta.

Esto permite que el software sea fácil de usar. Si se tiene una ruta con decenas de puntos, sería muy tedioso ingresar todos los puntos de la ruta por cada ejecución del algoritmo de búsqueda dispersa.

La base de datos se ha implementado usando *Microsoft Access*. El archivo *CVRPS.mdb* es el que almacena tanto la estructura de las tablas mencionadas como los datos que en ellas se encuentran. Al momento de escribir este documento de tesis se tienen 119 puntos almacenados y 12 rutas.

3.3 ADMINISTRACIÓN DE PUNTOS PARA EL RUTEO

Para administrar la información de los puntos se debe, en la página principal del software, escoger la opción *Administración de Puntos para el Ruteo en Guayaquil*. En ese momento el sistema nos dirige a la página web que se muestra a continuación:

Figura # 3.2 *Gestión de Puntos de Ruteo*

Podemos observar que se puede regresar al menú principal simplemente haciendo click en la parte superior izquierda usando un enlace del mismo nombre. Se aprecia que a la

izquierda de cada punto o nodo existe la opción de eliminar el nodo o de editar (actualizar) alguno de los campos del nodo. Si se desea ingresar un nuevo nodo, simplemente movemos la barra de la ventana, que se encuentre a la derecha, hacia abajo y veremos una pequeña opción del sistema para ingresar la información del nuevo nodo, lo que se muestra a continuación:

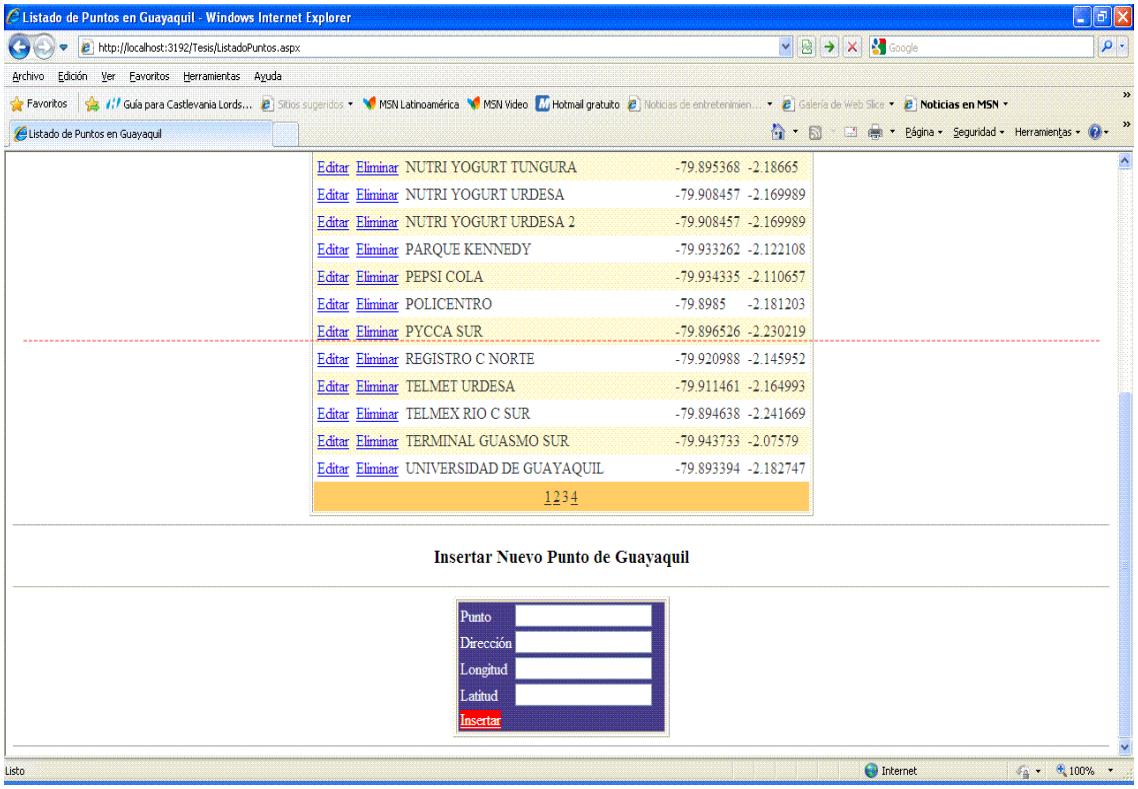

Figura # 3.3 ***Ingreso de Puntos de Ruteo***

Se puede observar en la figura 3.2 que también existe facilidad para la búsqueda de algún punto en particular si el número de puntos es extremadamente extenso. Supongamos que se desea buscar el punto etiquetado como *Policentro*. Se podría ingresar por ejemplo *Poli* en la caja de texto, luego hacer click en el botón *Consultar* que se encuentra a la derecha, y automáticamente el sistema filtrará a todos los puntos que comienzan con *Poli* en su descripción. Adicionalmente, el listado de los puntos se muestra mediante una técnica de programación web conocida como *paginación* que consiste en mostrar el listado por *páginas* de tamaño establecido y no el listado en su totalidad.

3.4 ALGORITMO BÚSQUEDA DISPERSA PARA EL PRVC SIMÉTRICO

Para ejecutar el algoritmo de búsqueda dispersa se debe escoger la opción *Algoritmo Búsqueda Dispersa para el PRVC Simétrico* A continuación se muestra la página web a la cual nos conduce la opción mencionada:

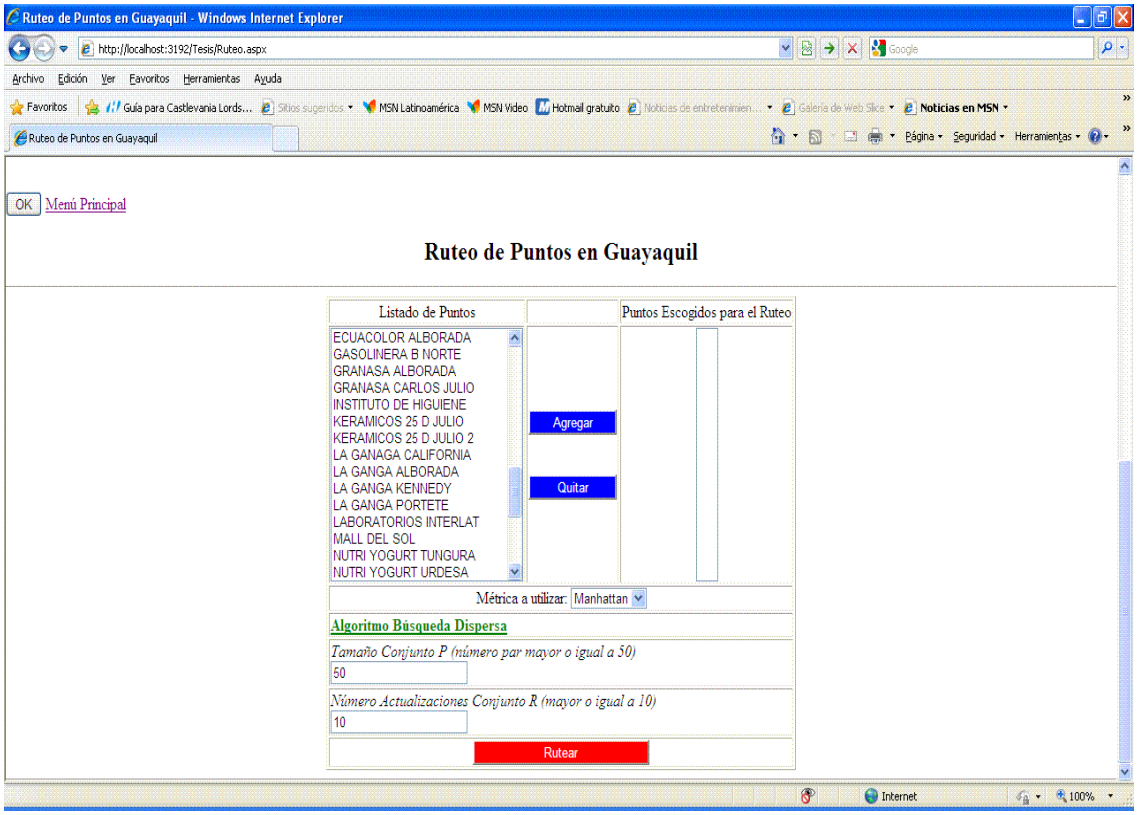

Figura # 3.4 *Algoritmo Búsqueda Dispersa PRVC Simétrico*

Podemos apreciar que del lado izquierdo tenemos el listado de puntos que podemos escoger para construir el ruteo. Si queremos considerar un punto para el ruteo simplemente lo seleccionamos del lado izquierdo, hacemos click en el botón *Agregar* y automáticamente ese punto pasa al lado derecho indicándonos así que es uno de los nodos que será considerado por el algoritmo de búsqueda dispersa. Si en algún momento no estamos de acuerdo con algún punto escogido para el ruteo, simplemente los escogemos del listado del lado derecho y presionamos el botón *Quitar*. La interfaz gráfica de usuario del sistema se diseñó de esta manera para que sea muy sencilla la manera de crear una ruta.

En la figura 3.4 podemos ver que la barra a la derecha de la ventana está situada completamente abajo. Lo que ocurre es que por encima de las opciones que permiten escoger los puntos para un ruteo, tenemos unas opciones que no se muestran. En estas opciones existe la alternativa de *Crear Ruta* si deseamos que todos los puntos escogidos más abajo para el ruteo, se almacenen como una ruta etiquetada a la que podamos cargar después de forma directa tal como se indicó en la sección 3.2, además se tiene también un listado de rutas y un botón *Cargar Ruta*. Si se desea cargar una de las rutas almacenadas en la base de datos, simplemente se accede al listado de rutas, se selecciona la que se desea y, finalmente, se hace click en el botón *Cargar Ruta*. Luego, automáticamente, los puntos o nodos de la ruta seleccionada aparecerán en el *Listado de Puntos Escogidos para el Ruteo*.

Podemos observar también que existe la opción de escoger la métrica con la cual se desea trabajar. Se puede escoger la *Métrica Euclidiana* o la *Métrica de Manhattan*. La opción por omisión o por *default* es la segunda métrica ya que refleja mejor la manera en que los vehículos se movilizan en una ciudad.

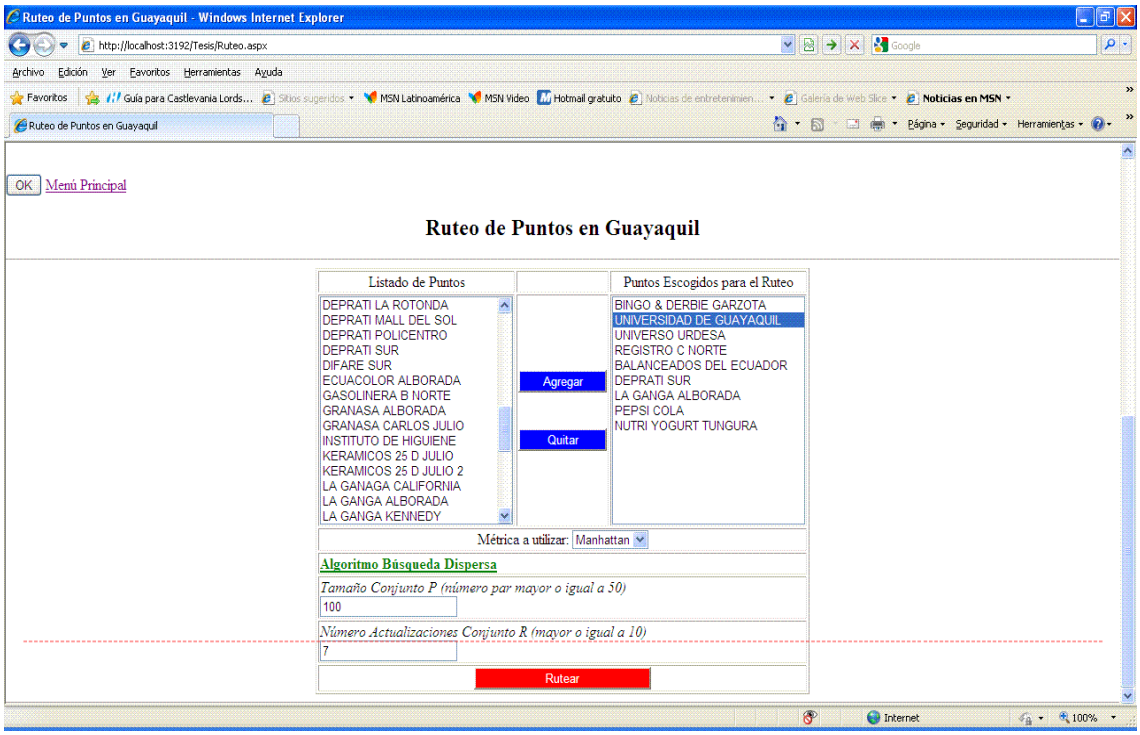

Figura # 3.5 *Puntos Escogidos para el Ruteo*

3.5 CONFIGURACIÓN DE PARÁMETROS

Finalmente, antes de ejecutar el algoritmo de búsqueda dispersa, es necesaria la configuración de ciertos parámetros. Un primer parámetro importante es el tamaño del conjunto P. En la figura 3.5 podemos ver que el valor de la variable $PSize$ se toma de una caja de texto etiquetada como *Tamaño Conjunto* P. Internamente el valor del parámetro $RSize$ se ajusta mediante la expresión $RSize = PSize/5$. Si $PSize$ no es divisible para 5 se escoge un valor par que se encuentre cerca de $PSize/5$ mediante un pequeño ajuste matemático. El tercer parámetro es el número de iteraciones. El valor para este parámetro se toma de la caja de texto etiquetada como *Número Actualizaciones Conjunto* R. Recordemos que una iteración equivale a una actualización del conjunto R. El parámetro *número de intentos* que nos dice cuántas veces debemos efectuar una búsqueda local como máximo en el método de mejora, no se lo pide mediante la interfaz gráfica de usuario pero sí en una especie de *variable global* dentro del código.

Antes de hacer *click* en el botón *Rutear*, que es el botón que inicia formalmente la ejecución del algoritmo, es importante que el usuario seleccione, dentro del listado de *Puntos Escogidos para el Ruteo*, el punto o nodo que indicará el inicio y el fin del ruteo del vehículo. Es decir, se debe seleccionar de la lista, aquel nodo desde el cual partirá el vehículo y donde finalmente llegará, una vez terminado el recorrido. En otras palabras, el usuario debe fijar el nodo 0 para el ruteo. Luego que el usuario hace click en el botón *Rutear* las líneas importantes de código en C# que se ejecutan son las siguientes:

```
double NumIntentos = 30;
Rutear obj = new Rutear(NumIntentos);
obj.EjecutarSS(int.Parse(txtPSize.Text),
               int.Parse(txtNumIter.Text),
               lstMetrica.SelectedValue, lstPuntosEscogidos)
```

La variable `NumIntentos` es el parámetro que se utiliza en la búsqueda local. Al interior del método `EjecutarSS` se invoca al método de la sección 2.2.7 llamado `SS_VRP`.

3.6 RESULTADOS COMPUTACIONALES

Para mostrar una ejecución del algoritmo se utilizaron los siguientes puntos (el orden en el cual se ingresaron los nodos al sistema es tal como se indica):

PuntoCod	PuntoDsc
66	Bingo & Derbie Garzota
13	Universidad de Guayaquil
42	Universo Urdesa
81	Registro Civil Norte
19	Balanceados del Ecuador
24	DePrati Sur
11	La Ganga Alborada
72	Pepsi Cola
92	Nutri Yogurt Tungurahua

Es decir, se trata de un ruteo de 9 puntos. Se escogió *19 – Balanceados del Ecuador* como el nodo especial 0. Los valores de los parámetros fueron fijados con los siguientes valores:

Parámetros de Ejecución

PSize	"100"
RSize	"20"
NumIter	"10"
NumIntentos	"30"
Métrica	"Manhattan"

El resultado de la ejecución se muestra a continuación, los puntos se muestran de abajo hacia arriba indicando el orden en el cual se deben ir visitando los nodos, partiendo y terminando el recorrido en *19 – Balanceados del Ecuador*:

PuntoCod	PuntoDsc
19	Balanceados del Ecuador
72	Pepsi Cola
81	Registro Civil Norte
66	Bingo & Derbie Garzota
11	La Ganga Alborada
42	Universo Urdesa
13	Universidad de Guayaquil
92	Nutri Yogurt Tungurahua
24	DePrati Sur
19	Balanceados del Ecuador

El sistema también nos muestra el resultado:

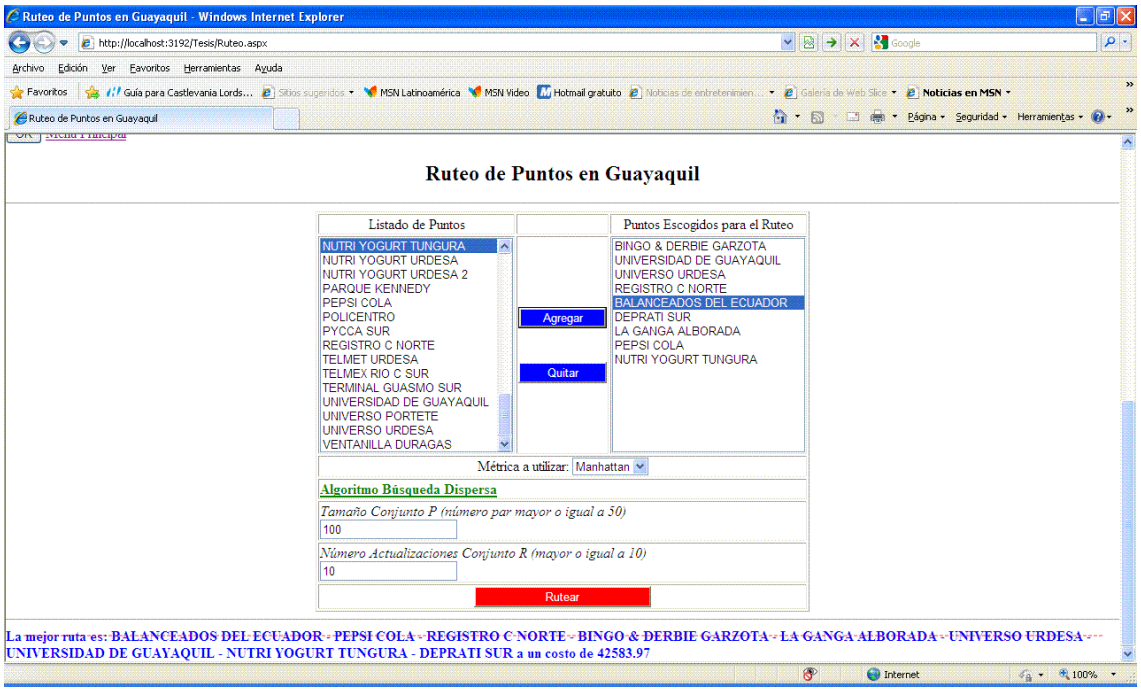

Figura # 3.6 *Resultados de Ejecución*

La ruta sugerida por el sistema nos dice que la distancia total recorrida por el vehículo será de aproximadamente 42 kilómetros y medio.

CONCLUSIONES

- Para problemas muy difíciles, debido principalmente a su tamaño, los métodos exactos no son adecuados. Por otro lado los métodos heurísticos y metaheurísticos no nos garantizan que podamos obtener la solución óptima del problema, pero sí nos permiten encontrar *buenas soluciones* con un apropiado consumo de recursos computacionales y en un tiempo de ejecución aceptable.

- Las metaheurísticas, como búsqueda dispersa, son métodos inexactos y no deterministas (debido al uso de la aleatoriedad), lo que hace muy complicado un análisis de convergencia. No existe garantía de encontrar el óptimo global del problema. Siempre, para cualquier metaheurística, está presente la posibilidad que quede atrapada en un óptimo local.

- La mezcla adecuada de *diversidad* y *calidad* que usa búsqueda dispersa, la convierte en una metaheurística atractiva para resolver problemas de optimización muy difíciles, en comparación con otras metaheurísticas que pueden llegar a abusar del azar y de la computación evolutiva.

- Las metaheurísticas, en general, son muy complicadas de implementar debido a la gran cantidad de actividades y tareas que realizan. Programar una metaheurística requiere mucha concentración, esfuerzo y dedicación.

- Las metaheurísticas se están usando actualmente para resolver numéricamente muchos problemas en áreas de las matemáticas como los sistemas dinámicos y las ecuaciones diferenciales. Existen metaheurísticas diseñadas para resolver sistemas de ecuaciones no lineales.

- Las empresas ecuatorianas necesitan sistemas de información logística. Es importante saber diseñar una metaheurística, pero es más importante aún tener la habilidad de construir software comercial que permita resolver problemas del mundo real usando metaheurísticas.

- Cualquier problema de transporte puede ser resuelto con metaheurísticas, pero mientras más requerimientos tenga (modalidades de *ventana de tiempo, entregar y recibir*, entre otras), más complicado será el diseño y la implementación.

- La computación y las matemáticas, en una mezcla adecuada, son la combinación perfecta que necesitan las empresas para una adecuada toma de decisiones.

- Para los problemas de transporte, mientras mayor es el número de puntos o nodos, más recursos computacionales consumirá la ejecución de la metaheurística y más tiempo de procesamiento estará involucrado

RECOMENDACIONES

- Sería importante, dada la experiencia del trabajo realizado, efectuar nuevos diseños e implementaciones para resolver otras modalidades más sofisticadas del problema de ruteo de vehículos, como son los esquemas de *ventanas de tiempo* y *entregar y recibir*. Se puede crear también un software DSS orientado al web que permita resolver muchos problemas diferentes de transporte.

- La implementación actual puede mejorarse si dentro de algunas partes del algoritmo de búsqueda dispersa, como en la *búsqueda local*, la *combinación*, la *formación de grupos*, entre otras tareas, se intentan manejar muchas de las variantes mencionadas haciendo uso de parámetros. Por ejemplo, si se tienen formas distintas de efectuar una *combinación* entre soluciones factibles, se puede parametrizar el hecho que, en la misma ejecución, algunos grupos usen un modo de combinación y otros grupos utilicen otro modo diferente de combinación. Manejar distintas variantes en los distintos métodos complicaría el diseño y la implementación final de la metaheurística, pero podría presentar mejores resultados.

- Se debe considerar el empleo de otras metaheurísticas, además de búsqueda dispersa, para resolver problemas de transporte. Se pueden implementar dos o tres metaheurísticas distintas para resolver un mismo problema de ruteo de vehículos. Posteriormente se puede hacer un estudio de los tiempos de ejecución, el consumo de recursos y la calidad de las soluciones que entregan.

- Al construir un software DSS que resuelve problemas de transporte, son importantes las características gráficas del sistema de información. Actualmente se pueden usar técnicas de programación que permiten el uso de mapas, como los de *Google Earth*, para mostrar los puntos del ruteo. Esto hace que el usuario pueda ubicar los nodos de manera visual y vuelve al software más amigable y fácil de usar.

REFERENCIAS BIBLIOGRÁFICAS

[1] Laudon K. C., Laudon J. P. (2012). "Sistemas de Información Gerencial", Edición 12, Editorial Pearson Education

[2] Orfali R., Harkey D., Edwards J. (2002). "La Guía de Supervivencia Cliente/Servidor", Edición III, Publicador John Wiley & Sons, Inc.

[3] Martín C. (2011) "Scatter Search y la Optimización Funcional", Revista Matemática, Volumen 9, Número 2, FCNM – ESPOL.

[4] Laguna M., Martí R. (2003). "Scatter Search: Methodology and Implementations in C", Kluwer Academic Publishers, Norwell Massachusetts

[5] Martí R., Laguna M. (2003). "Scatter Search: Diseño Básico y Estrategias Avanzadas", Universidad de Valencia, España

[6] Áslaug Sóley Bjarnadóttir (2004). "Solving The Vehicle Routing Problem with Genetic Algorithms", Informatics and Mathematical Modeling, IMM.

[7] Sitio Web "Microsoft Developer Network" (MSDN). ONLINE. http://msdn.microsoft.com

[8] Korte B., Vygen J. (2008). "Combinatorial Optimization, Theory and Algorithms", Edición 4, Springer Series

[9] Castillo E., Conejo A., Pedregal P., García R. (2002). "Formulación y Resolución de Modelos de Programación Matemática en Ingeniería y Ciencia", Editorial Wiley

[10] Dantzig G., Thapa M. (2001). "Linear Programming, Introduction", Springer Series in Operations Research

[11] Dantzig G., Thapa M. R. (2003). "Linear Programming, Theory and Extensions", Springer Series in Operations Research

[12] Conejo A., Castillo E., Minués R., (2006) "Decomposition Techniques in Mathematical Programming, Engineering and Science Applications", Springer Series

[13] Tang J., Zhang J., Pan Z. (2006) "A Scatter Search Algorithm for Solving Vehicle Routing Problem with Loading Cost", Journal Expert Systems with Applications, Elsevier Ltd, Volumen 37