UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA
CENTRO DE INVESTIGACIÓN EN INGENIERÍA
Y CIENCIAS APLICADAS

ALGORITMO DE AGRUPAMIENTO PARA EL PROBLEMA DEL TRANSPORTE

TESIS PROFESIONAL

PARA OBTENER EL GRADO DE
MAESTRO EN INGENIERIA Y CIENCIAS APLICADAS
OPCIÓN TERMINAL EN TECNLOGÍA ELÉCTRICA

P R E S E N T A :

I.S.C. ALFONSO D'GRANDA TREJO

ASESOR: DR. MARCO ANTONIO CRUZ CHÁVEZ

CUERNAVACA, MOR. DICIEMBRE 2013

DEDICATORIA

A mi esposa María Elena

Quien ha sido una influencia muy importante en mi formación académica, pero sobre todo, compañera de vida.

A mis hijos Mariel Alejandra y Alfonso Uriel.

AGRADECIMIENTOS

A CONACYT por brindarme el apoyo económico para hacer posible la realización de mis estudios de maestría.

A mi asesor Dr. Marco Antonio Cruz Chávez

A mi comité tutoral y revisores

Dra. Margarita Tecpoyotl T.

Dr. Martin Heriberto Cruz R.

Dr. Martin Martínez R.

RESUMEN

En esta tesis se presenta un algoritmo de agrupamiento para el problema del transporte VRP (por sus siglas en inglés), el cual genera soluciones factibles, de calidad y con número de rutas reducido. El algoritmo de agrupamiento propuesto es parte de una estrategia de dos fases “agrupar primero, rutear después”. Este trabajo se enfoca en la primera fase, en donde se genera una solución inicial que sirve como parámetro de entrada en la segunda. La Heurística desarrollada en el algoritmo propuesto hace una combinación de estrategias de heurísticas reportadas en la literatura. Se realizaron pruebas con benchmarks de Solomon para 100 clientes comparando contra otras heurísticas, y pruebas con benchmarks de Gehring & Homberger's para 1000 clientes comparando contra otras metaheurísticas; dando como resultado que el algoritmo propuesto tenga un excelente desempeño, logrando reducir el número de rutas en instancias de 100 en comparación con otras heurísticas e incluso con metaheurísticas, con instancias de 1000 clientes consigue igualar o reducir el número de rutas cuando la demanda de los clientes es alta.

ABSTRACT

This thesis presents a clustering algorithm for the vehicle routing problem (VRP), which generates viable solutions of quality and with a reduced number of routes. This clustering algorithm uses a strategy of two phases “group first, route later”. This paper focuses on the first phase, where an initial solution which serves as a parameter of entry to the second phase is generated. The Heuristic developed in the algorithm earlier proposed, makes a combination of strategies of heuristics reported in literature. Tests were conducted with Solomon’s benchmarks for 100 clients comparing against other heuristics and tests with Gehring and Homberger’s benchmarks for 1000 clients comparing them against other metaheuristics; giving as a result that the algorithm suggested successfully manages to reduce the number of routes in instances of 100 in comparison with other heuristics and in the comparison with metaheuristics with instances of 1000 clients it manages to equal or reduce the number of routes when the demand of the client is high.

CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

1. INTRODUCCIÓN

Todas las compañías tienen una forma de planear y coordinar las actividades necesarias para que un producto o servicio llegue al punto donde y cuando el cliente final lo requiera, sin importar el tamaño ni la complejidad de sus clientes, de modo que, los productos lleguen de manera exitosa al cliente. Dentro de la investigación este problema es conocido como el problema de ruteo vehicular (VRP) cuya problemática es definir el menor número de rutas de manera que se minimice el costo total del recorrido, y se puede definir de manera formal como el conjunto de N clientes que cuentan con una demanda d, la cual debe ser satisfecha por uno de k vehículos disponibles, de modo que la suma de la demanda de los clientes asignados a un vehículo k no exceda la capacidad máxima C del vehículo k; este problema tiene muchas variantes, de las que, se aborda la del problema transporte con restricciones de capacidad (CVPR Problema de Ruteo Vehicular con Restricción de Capacidad, de ahora en adelante). Para la solución de este problema existen diferentes métodos, entre ellos el método conocido como “Método de dos fases”, en él primero se busca generar grupos de clientes, también llamados clusters, que estarían en una misma ruta en la solución final; Luego, para cada cluster se crea una ruta que visite a todos sus clientes. Las restricciones de capacidad son consideradas en la primera etapa, asegurando que la demanda total de cada cluster no supere la capacidad del vehículo. En esta investigación solamente se trabaja con la primera fase, creando el conjunto de grupos con las características deseables para entregar una buena solución inicial a la segunda fase para su desarrollo.

1.2 DESCRIPCIÓN DEL PROBLEMA

La finalidad de los algoritmos de agrupamiento es agrupar los datos de forma que los objetos dentro de cada grupo sean más similares a los del mismo que a los de los otros grupos (Anderberg, 1973). Aplicado a la planificación de la flota de vehículos, este tipo de algoritmos se utiliza para atender a los clientes, asignado un conjunto de clientes a cada vehículo y esto genera una ruta. Los clientes que conforman cada grupo preferentemente deben estar cercanos geográficamente (esta cercanía es una característica de similaridad) para proporcionar soluciones factibles que satisfagan las restricciones del problema de agrupamiento y generen un número de rutas reducido.

1.3 JUSTIFICACIÓN

Los métodos de agrupamiento que se han propuesto y desarrollado en los últimos años son bastante numerosos y muy diversos en cuanto a su concepción, así entonces, existen métodos que crean agrupamientos esféricos, con base en densidades, en forma de pétalos, entre otros; la técnica a utilizar depende, sobre todo, de la naturaleza de los datos usados y de los objetivos finales perseguidos. En esta investigación se desarrolla un algoritmo de agrupamiento que no requiere especificar *a priori* los grupos que deben ser formados, además de genera grupos con formas irregulares; siendo éstas, algunas de las principales diferencias con respecto a los métodos existentes.

1.4 OBJETIVO GENERAL

Proponer un algoritmo de agrupamiento para el problema de CVRP que incorpore a los clientes en grupos que sirvan como una solución inicial a la generación de las rutas, respetando la restricción de capacidad propia del CVRP.

1.5 OBJETIVOS ESPECÍFICOS

- Incorporar estrategias de agrupamiento para mejorar el desempeño del algoritmo propuesto.
- Encontrar soluciones factibles y de calidad para problemas de grandes dimensiones.
- Reducir el número de rutas generadas con el algoritmo propuesto, en problemas de grandes dimensiones.
- Probar el algoritmo propuesto con otras heurísticas para evaluar su desempeño.

1.6 ALCANCE

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

1.7 ORGANIZACIÓN DE LA TESIS

El presente trabajo, se encuentra organizado de la forma siguiente; en el capítulo 1, se una introducción al tema y se describe el problema que se abordó en la investigación: sus características y restricciones principalmente, así como el objetivo general y los específicos. Para el capítulo 2 se presenta

el marco teórico que sustenta los métodos y procedimientos de trabajo además del estado del arte de las investigaciones realizadas al respecto. El planteamiento del problema se aborda en el capítulo 3 en donde se muestran los criterios para el desarrollo del modelo matemático, el cual se muestra también, la definición del espacio de soluciones y la complejidad del problema que justifica el uso de la optimización combinatoria. La propuesta de solución del problema presentado es presentada en el capítulo 4, en el cual se detallan las instancias de trabajo empleadas, los criterios más utilizados en investigaciones anteriores y la forma en que se realizaron los agrupamientos. En el capítulo 5, se presenta la evaluación de resultados y finalmente las conclusiones y recomendaciones se enumeran en el capítulo 6.

1.8 CONTRIBUCIÓN

Esta investigación aporta una nueva forma de realizar agrupamientos para el algoritmo de CVRP la cual consiste en generar el menor número de rutas de una instancia dada considerando el trayecto hacia el depósito para la integración de clientes a los grupos (rutas), sin afectar el costo del recorrido.

2. MARCO TEÓRICO

En este capítulo se describe el problema del transporte, las variantes y técnicas de solución que existen de él, haciendo énfasis en el problema de ruteo vehicular con restricción de capacidad y su solución en una estrategia de dos fases, la cual, es utilizada en esta investigación. Se proporciona una breve explicación de qué es agrupar y del uso de técnica heurísticas. Finalmente se presenta algunos de los algoritmos desarrollados para la solución del problema del transporte con restricción de capacidad.

2.1 PROBLEMA DEL TRANSPORTE

El problema de ruteo vehicular (VRP por sus siglas en inglés), también conocido como el problema del transporte, es un nombre genérico que se da a toda una clase de problemas en los que se debe determinar un conjunto de rutas para una flota de vehículos, que salen y regresan de uno o varios depósitos, para un número de ciudades o clientes geográficamente dispersos. El objetivo del VRP es asignar un conjunto de clientes con demandas conocidas en las rutas de vehículos, originando el mínimo costo por recorrido e inicien y terminen en un depósito. En la Figura 2.1 se ejemplifica una entrada típica para un problema VRP marcada con líneas delgadas y uno de sus posibles salidas marcada con líneas más gruesas:

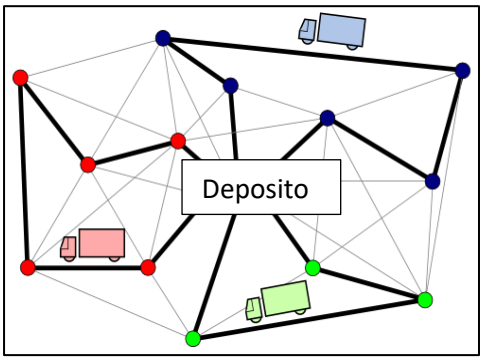


Figura 2. 1. Ejemplo de solución al VRP.

El VRP es un problema de programación entera bien conocido que cae en la categoría de problemas NP duros, lo que significa que el esfuerzo computacional requerido para resolver este problema aumenta exponencialmente con el tamaño del problema. Para este tipo de problemas que requieren un gran esfuerzo computacional, a menudo, es deseable buscar métodos que brinden soluciones aproximadas, que puedan encontrar lo suficientemente rápido y sean suficientemente precisos para el propósito. Por lo general, esta tarea se lleva a cabo mediante el uso de diversos métodos heurísticos, los cuales se basan en una idea de la naturaleza de un problema.

Este problema combinatorio difícil conceptualmente se encuentra en la intersección de estos dos problemas muy estudiados; la asignación de clientes a vehículos (Knapsack Problem) y establecer la secuencia adecuada de visita a los clientes que debe realizar un vendedor o vehículo (Traveling Salesman Problem o TSP).

El VRP surge originalmente (Dantzig & Ramser, 1959) como un problema central en los campos del transporte, la distribución y la logística. En algunos sectores del mercado, el transporte significa un alto porcentaje del valor agregado a los bienes. Por lo tanto, la utilización de métodos informatizados para el transporte a menudo resulta en ahorros significativos que van desde 5% a 20% en los costos totales, como se informa en (Toth & Vigo, 2002). Por lo general, en el mundo real VRP, aparecen muchas limitaciones. Algunas de las restricciones más importantes son:

- CVRP (Capacitated VRP), la variante más genérica y práctica, que introduce la restricción de capacidad de los vehículos, en la que la carga de las mercancías de una ruta no podrá exceder la capacidad fijada para cada vehículo de la flota.

- MDVRP (Multi Depot VRP), en la que se considera que existen dos o más depósitos desde los cuales servir a los clientes.
- PVRP (Periodic VRP), en el que la planificación se realiza para M días.
- SDVRP (Split Delivery VRP), simplificación de CVRP en la que se permite que un cliente pueda ser servido por varios vehículos.
- SVRP (Stochastic), en el que se consideran que uno o varios componentes del problema son aleatorios. Estos componentes pueden ser clientes, demandas y/o tiempo de servicio.
- VRPB (VRP with Backhauls), en la que un cliente puede devolver mercancía. - VRPPD (VRP with Pick-Up and Delivering), similar a VRPB salvo que la recolección de mercancías a devolver se realiza una vez concluida la entrega de todas las mercancías de esa ruta.
- VRPSF (VRP with Satellite Facilities), en la que se considera que un vehículo puede cargar mercancías en plena ruta, eliminando la necesidad de volver al depósito.
- VRPTW (VRP with Time Windows), en la que se considera que un cliente tiene asociada una ventana de tiempo durante la cual debe ser servido.

2.1.1 VRP CON RESTRICCIÓN DE CAPACIDAD (CVRP)

El problema de VRP con restricción de capacidad en su restricción (2.6) utiliza el algoritmo propuesto para determinar el valor de r(S) de esta restricción mostrada más adelante en el modelo matemático del VRP Figura 2. 2 . Este problema es una extensión del clásico 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 centro de depósito, “n” vehículos con capacidad definida 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 visitar a todos los clientes 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.

El VRP es una extensión del problema del múltiples agentes viajeros (m-TSP por sus siglas en inglés), en la cual, cada cliente tiene asociada una demanda y cada vehículo tiene una capacidad C (la flota es homogénea). En este problema la cantidad de rutas no es fijada de antemano como en el TSP y en el m-TSP.

Para un conjunto de clientes S, d(S) = $\sum_{i \in S} d_i$ es su demanda total y r(S) indica la mínima cantidad de vehículos necesarios servirlos a todos. En la formulación conocida con el nombre de flujo de vehículos de dos índices, se utilizan las variables binarias para determinar si el arco (i, j) se utiliza o no en la solución. El problema se formula de la siguiente manera (Toth & Vigo, 2000):

La función objetivo (2.1) es el costo total de la solución. Las restricciones (2.2) y (2.3) indican que m es la cantidad de vehículos utilizados en la solución y que todos los vehículos que parten del depósito deben regresar. Las restricciones (2.4) y (2.5) aseguran que todo cliente es un nodo intermedio de alguna ruta. Finalmente, la restricción (2.6) actúa como

restricción de eliminación de sub-tours y a la vez impone que la demanda total de los clientes visitados por un vehículo no puede superar la capacidad C. Determinar el valor de r(S) requiere establecer el número de rutas; este valor está sujeto a las restricciones del modelo matemático utilizado en el problema de agrupamiento. La ec. 2.6 determina el número mínimo de vehículos que se pueden tener y la 2.8 son las variables binarias.

Min $$Z = \sum_{(i,j)\in E} c_{ij} x_{ij} \qquad (2.1)$$

s. a.

$$\sum_{j\in\Delta+(0)} x_{0j} = m \qquad (2.2)$$

$$\sum_{i\in\Delta-(0)} x_{i0} = m \qquad (2.3)$$

$$\sum_{j\in\Delta+(i)} x_{ij} = 1 \quad \forall \quad i \in V \setminus \{0\} \qquad (2.4)$$

$$\sum_{i\in\Delta-(j)} x_{ij} = 1 \quad \forall \quad j \in V \setminus \{0\} \qquad (2.5)$$

$$\sum_{i\in S, j\in\Delta+(i)\setminus S} x_{ij} \geq r(S) \quad \forall \quad S \subset V \setminus \{0\} \qquad (2.6)$$

$$m \geq 1 \qquad (2.7)$$

$$x_{ij} \in \{0,1\} \quad \forall \ (i,j) \in E \qquad (2.8)$$

Figura 2. 2 Modelo matemático del VRP.

2.1.2 TÉCNICAS DE SOLUCIÓN PARA EL VRP

Las técnicas más utilizadas para la solución de problemas de ruteo de vehículos. Tienen que ver con el uso de heurísticas y metaheurísticas ya que ningún algoritmo exacto puede garantizar para encontrar recorridos óptimos dentro de tiempo de cálculo razonable cuando el número de ciudades es grande. Esto es debido a la complejidad NP-Duro del problema. En la fig. 2.3 se presenta una clasificación de las métodos de solución:

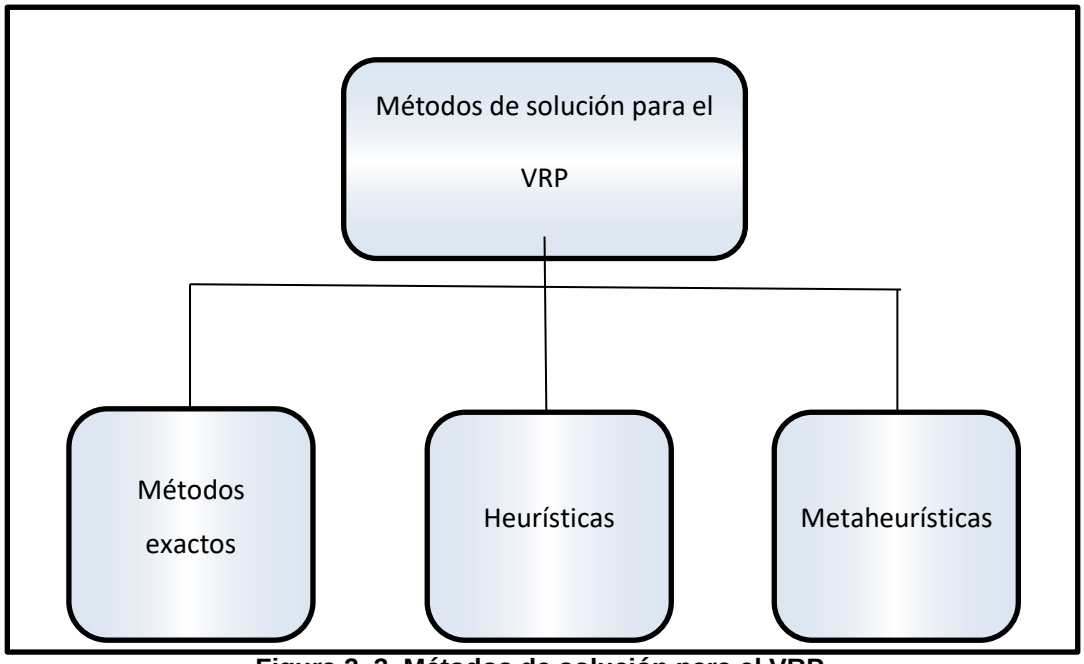


Figura 2. 3 Métodos de solución para el VRP.

Métodos exactos: Como su nombre lo indica, este enfoque propone para calcular todas las soluciones posibles hasta que se alcanza una de las soluciones óptimas.

Heurística: Los métodos heurísticos realizan una exploración relativamente limitada del espacio de búsqueda y por lo general producen soluciones de buena calidad dentro de los tiempos de computación modestos.

Métodos constructivos: construyen gradualmente una solución factible, manteniendo visto el precio de la solución, pero no contienen una fase de mejora.

2.2 HEURÍSTICAS Y METAHEURÍSTICAS

Los métodos heurísticos son aquellos que dan una solución factible al problema, sin asegurar que esa solución obtenida sea la óptima.

El término heurística proviene del vocablo griego heuriskein que significa encontrar, descubrir o hallar. Desde un punto de vista científico, el termino heurística se debe al matemático (Polya, 1971). Actualmente, existen muchas definiciones para la palabra heurística. Una de las definiciones más claras e intuitivas es la de procedimientos simples, generalmente basados en el sentido común, con el objetivo de obtener una buena solución (no necesariamente óptima) a problemas difíciles de un modo sencillo y rápido (Duarte Muñoz, Pantrigo Fernández, & Gallego Carrillo., 2007).". Un procedimiento se califica heurístico cuando se tiene un alto grado de confianza en que la soluciones "encontradas" son de alta calidad, es decir, con un tiempo de solución computacional razonable, aunque no garanticen una solución óptima. En los métodos heurísticos, la rapidez del proceso es tan importante como la calidad de la solución obtenida. Se usa el calificativo heurístico en contraposición a exacto (Diaz, Glover, & Ghaziri, 1996).

Se puede decir entonces que, los métodos heurísticos son procedimientos que se usan para resolver los problemas de optimización mediante una aproximación intuitiva, en la que la estructura del problema se utiliza para obtener una buena solución (Melían & Pérez, 2003).

Los procedimientos heurísticos son usados para resolver problemas de optimización cuando (Marti):

- No se conoce ningún método exacto que le brinde solución.
- Se conoce un método exacto que le brinde solución pero el tiempo computacional que requiere es demasiado largo.
- El método heurístico es más flexible que el método exacto, es decir, permite incorporar condiciones que son difíciles de modelar.

Una metaheurística es un proceso iterativo que hace uso de una heurística subordinada combinándola con diferentes conceptos de una manera inteligente con el fin de encontrar soluciones cercanas al óptimo de manera eficiente (Osman & Kelly, 1996).

2.3 CONCEPTO DE AGRUPAMIENTO

2.3.1 INTRODUCCIÓN

Agrupar tiene que ver con la división de los datos en subgrupos homogéneos. En los que se busca que: Los elementos dentro de un grupo deben ser similares entre ellos (similaridad inter-grupo), mientras que, los que estén dentro de los otros grupos deben ser diferentes (disimilaría intra-grupo) (Tan, Steinbach, & Kumar, 2006). En la Figura 2.4 se muestra un ejemplo de la forma de agrupar los datos.

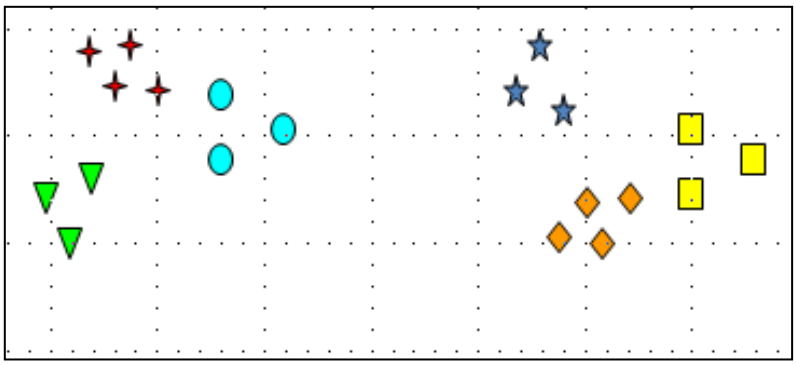


Figura 2. 4 Ejemplo de datos agrupados.

La aplicación de estas técnicas está asociada a la solución de problemas en una variedad de disciplinas que van desde la sociología y la psicología, con el comercio, biología y ciencias de la computación y algoritmos para hacer frente a ellos siguen siendo objeto de investigación activa.

Por consiguiente, existen una multitud de métodos de la agrupación, que no difieren sólo en los principios del algoritmo utilizado (que, por supuesto, determina el comportamiento de tiempo de ejecución y escalabilidad), sino también en muchas de sus propiedades más básicas, cómo manejan los datos (numérico vs categórica y datos de proximidad), los supuestos sobre la forma de los grupos (por ejemplo, de forma esférica), la forma de la partición final (asignaciones dura *vs* difusa) o los parámetros que tienen que ser proporcionados (por ejemplo, el número correcto de grupos) .

Es importante señalar la diferencia entre la clasificación de datos supervisada y la clasificación de datos no supervisada (Hansen & Jaumard, 2000):

- Clasificación de datos supervisada. En la clasificación, se conoce de antemano los valores de la membrecía o la pertenencia de un conjunto de objetos a los diferentes grupos establecidos. Lo que se busca es desarrollar mecanismos adecuados para clasificar nuevos objetos de acuerdo a los datos que ya se encuentran clasificados.
- Clasificación de datos no supervisada. En la clasificación, el valor de la membrecía que tienen los objetos en los distintos grupos no se conoce previamente. Por lo tanto, los grupos se forman de acuerdo a un criterio que determina el grado de similitud entre los objetos. Los objetos con alto grado de similitud establecen un mismo grupo. Se debe tratar que los objetos pertenecientes a distintos grupos sean en lo posible, lo más diferentes entre sí. La clasificación de datos no supervisada se conoce también como clustering.

Dentro del estado del arte, que ha sido el punto de partida para el desarrollo de la investigación propuesta. Se muestra una panorámica de los principales trabajos relacionados con el área. Dado que el objetivo de la tesis es proponer un algoritmo de agrupamiento que genere una solución inicial factible para el problema del transporte, se hace a continuación una revisión de las contribuciones más relevantes en estrategia de dos fases que consiste en asignar primero y rutear después (cluster first - route second) proceden en dos fases. Primero se busca generar grupos de clientes, también llamados cluster, que estarán en una misma ruta en la solución final. Luego, para cada cluster se crea una ruta que visite a todos sus clientes. Las restricciones de capacidad son consideradas en la primera etapa, asegurando que la demanda total de cada cluster no supere la capacidad del vehículo. Por lo tanto, construir las rutas para cada cluster es un TSP que, dependiendo de la cantidad de clientes en el cluster, se puede resolver en forma exacta o aproximada. Estos métodos dividen al problema en ruteo y asignación de

clientes a las rutas, el orden en que se realiza cada una de estas fases varía de acuerdo a la estrategia del método.

Entre los métodos de dos fases se encuentran los métodos de asignación elemental, el algoritmo de ramificación y acotamiento truncados, el Algoritmo de los pétalos, el método de rutear primero y asignar después y los procedimientos de búsqueda local. Como se puede ver en la Figura 2.4 se muestra la clasificación establecida a partir de los autores (Toth & Vigo, 2002) (Olivera, 2004) (Laporte, 1991). Posteriormente se presenta una descripción de cada uno.

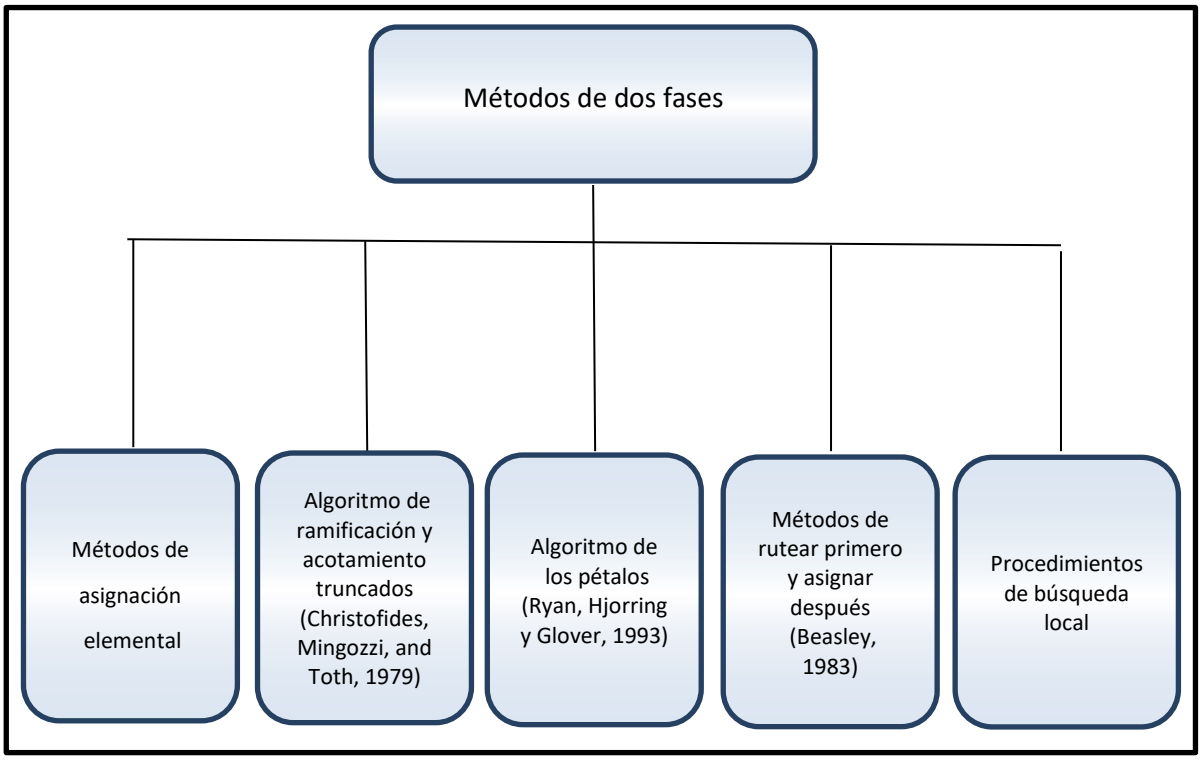


Figura 2. 4 Métodos de dos fases.

2.3.1 Métodos de asignación elemental (Elementary clustering methods)

En este tipo de métodos se encuentran el algoritmo de barrido (sweep algorithm), el algoritmo basado en asignación generalizada (generalizad-assignment-based algorithm) y la heurística basada en localización (location based heuristic) (Toth & Vigo, 2002) (Olivera, 2004). El algoritmo de Barrido (conocido también como sweep algorithm) consiste en formar inicialmente agrupamientos girando una semirrecta con origen en el depósito e incorporando los clientes hasta violar la restricción de capacidad. Una ruta de vehículos es obtenida para el cluster resolviendo un TSP. En algunos casos de implementación es necesaria una fase de post-optimización en la cual los vértices se intercambian entre clusters adyacentes y las rutas son re-optimizadas.

Por su parte, el algoritmo basado en asignación generalizada (generalizad-assignment-based algorithm) en lugar de utilizar un método geométrico para formar clusters utiliza un Problema de Asignación Generalizada (GAP por sus siglas en inglés). La primera fase de este algoritmo consiste en escoger los vértices semilla para construir los agrupamientos. En la segunda fase se asignan los vértices a cada agrupamiento sin violar la capacidad del vehículo resolviendo un GAP.

Por último en la heurística basada en localización (en inglés location based heuristic) las rutas iniciales (semillas) son establecidas como un problema de localización con capacidades y los vértices restantes son incluidos gradualmente en la ruta asignada en una segunda etapa. Para satisfacer las restricciones de capacidad y minimización de los costos se debe decidir sobre las semillas a colocar y los terminales a los cuales se van a conectar cada semilla. Las rutas de los vehículos se construyen entonces, insertando en cada paso, el cliente asignado para que las semillas tengan el menor costo de inserción.

2.3.2 Métodos de Ramificación y Acotamiento

Truncados (en inglés Truncated Branch and Bound). En este algoritmo el árbol de búsqueda tiene tantos niveles como rutas de vehículos y cada nivel contiene un conjunto de rutas de vehículos, para ello Christofides, Mingozzi y Toth proponen una implementación en la cual determinan una rama en cada nivel y una se descarta al paso de selección de la ruta. Se puede construir un árbol limitado manteniendo pocas rutas en cada nivel (Toth & Vigo, 2002).

2.3.3 Algoritmos de los Pétalos (en inglés Petal Algorithms).

Este algoritmo es una extensión del algoritmo de barrido y se utiliza para resolviendo un Set Partitioning Problem. Se dispone de un conjunto de rutas generar varias rutas llamadas pétalos con el fin de hacer una selección final R en la que cada cliente es visitado por varias rutas y se debe seleccionar un subconjunto de R que visite exactamente una vez cada cliente (Toth & Vigo, 2002) (Olivera, 2004) (Laporte, Reanud, & Boctor, 1996).

2.3.4 Métodos de Ruteo Primero y Asignación Después.

Este método consta de dos fases. En la primera se calcula una gran ruta que visita a todos los clientes resolviendo un TSP sin tener en cuenta las restricciones del problema. Luego en la segunda fase, esta ruta gigante se descompone en varias rutas factibles, es decir, teniendo en cuenta la solución de la primera fase se determina la mejor partición teniendo en cuenta la capacidad del vehículo (Toth & Vigo, 2002) (Olivera, 2004).

2.3.5 Procedimientos de Búsqueda Local.

Los procedimientos de búsqueda local se aplican para mejorar una solución ya obtenida. En estos procedimientos se define un conjunto de soluciones

vecinas y parte de una solución primaria para luego remplazarla por una solución vecina con menor costo. El procedimiento se repite hasta que no pueda mejorar la solución. Los diferentes procedimientos de búsqueda local, establecida a partir de los autores (Toth & Vigo, 2002) (Olivera, 2004) (Laporte, 1991).

Para problemas del VRPTW a gran escala (Mester & O.) hacen una combinación esta estrategia en la primera fase y y estrategias evolutivas en la segunda.

En el trabajo de Csiszár (Csiszár, 2007), en la primera fase se minimiza el número de rutas y en la segunda fase minimiza la distancia recorrida implementando una búsqueda tabú. Sus resultados fueron competitivos respecto de las mejores heurísticas publicadas en los últimos años (2003-2005). En otras investigaciones como (Taillard, Badeau, Gendreau, Geurtin, & Potvin, 1997) utilizan También búsqueda tabú.

Otros como Gehring y Homberger (Homberger & Gehring, 2005) en su primera fase minimizaron el número de vehículos y en la segunda, la distancia total recorrida, mostrando que el enfoque del método de las dos fases es muy competitivo.

Para la solución del CVRP (LIn, 2009) propuso un algoritmo híbrido de recocido simulado y búsqueda tabú, con muy buenos resultados en 8 instancias clásicas.

La propuesta de (Rochat & Taillar, 1995) es un algoritmo de búsqueda tabú que genera un número inicial de soluciones como entrada, de donde los recorridos son agregados a una memoria adaptativa.

2.3.6 Método de agrupamiento no supervisado para el CA-CVRP.

Otro algoritmo no contemplado en esta clasificación es: Método de agrupamiento no supervisado para el problema de ruteo vehicular (CA-VRP) (Cruz-Chavez & Martinez-Oropeza, 2011).

- Lee el benchmark de entrada.
- Hace cálculo de las distancias euclidianas correspondientes a los clientes y al depósito.
- Selecciona un centroide de forma aleatoria. (Cabe mencionar, que a diferencia de otros métodos de agrupamiento, para este algoritmo un centroide es un cliente, lo que evita el trabajo de recalcular distancias y centroides en cada iteración).
- Verifica que el cliente elegido no haya sido asignado a otra ruta, si esta condición se cumple.
- Toma el centroide y se inicia la búsqueda del cliente más cercano al centroide elegido y que no haya sido previamente asignado.

“Un punto importante a destacar, es que el algoritmo no sólo se enfoca en agrupar los clientes más cercanos a un centroide dado, sino que también evalúa la restricción correspondiente a la demanda de los clientes y capacidad máxima del vehículo, de modo que si el cliente seleccionado excede la capacidad máxima del vehículo actual, dicho cliente es descartado y se continúa en la búsqueda de otro cliente que pueda ser asignado al grupo, de lo contrario, se genera otra ruta, hasta que todos los clientes hayan sido asignados a un vehículo” (Cruz-Chavez & Martinez-Oropeza, 2011), el pseudocódigo de este algoritmo se muestra en la Figura 2.5.

```
1. Inicializar (N, Vh, C)
2. Leer Datos de Entrada
3. para i=0 : i <N
        para j=0 : j < N
            d[i][j] = √((x2 - x1)^2 + (y2 - y1)^2)
        fin-para
   fin-para
4. Inicializar Estructura Solution
            vect_tabu, index
5. Repetir
    centroide = 1+(rand()%N)
        si vect_tabu[centroide] == 1 entonces
           Repetir centroide = 1+(rand()%N)
           Hasta vect_tabu[centroide] == 0
        fin-si
        si-no
           vect_tabu[centroide] == 1
           Busca_cliente_mas_cercano_al_centroide
           si (di + Σ di) ≤ C entonces

              si vect_tabu[i] == 1 entonces
                 Repetir Busca_cliente_mas_cercano_al_centroide
                 Hasta vect_tabu[i] == 0
                 si (di + Σ di) > C entonces v++ fin-si
              fin-si
            fin-si
            si-no
              vect_tabu[i] == 1
              Solution[index] = i
         fin si-no
      fin si-no
   Hasta index == N-1
6. Imprime Agrupamiento
```

Figura 2. 5 Pseudocódigo CA-CVRP

Tomado de (Cruz-Chavez & Martinez-Oropeza, 2011).

3. PLANTEAMIENTO DEL PROBLEMA

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

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

3.1 CRITERIOS DE AGRUPAMIENTO

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

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

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

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

3.2 MODELO MATEMATICO

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

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

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

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

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

Figura 3. 1 Modelo matemático del problema de agrupamiento.

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

3.3 ESPACIO DE SOLUCIONES

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

3.4 COMPLEJIDAD DEL PROLEMA

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

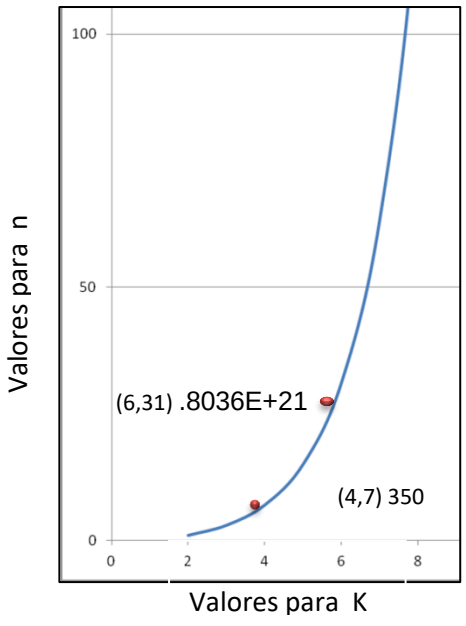


Figura 3.2 Números de Stirling 100 clientes.

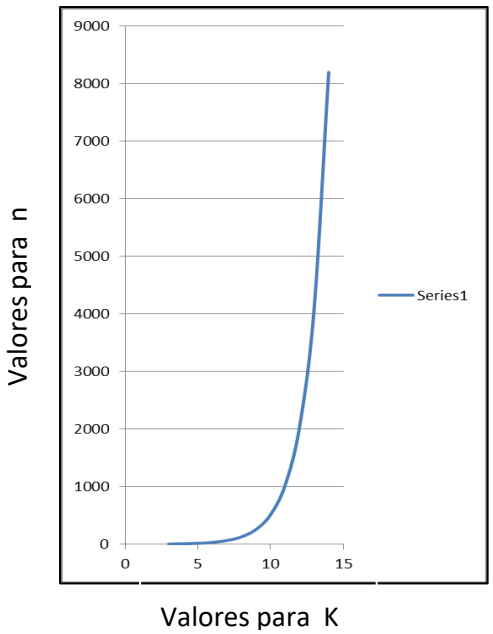


Figura 3.3 Números de Stirling 900 clientes.

En la Figura 3.2 se muestra gráficamente el crecimiento de la cantidad posible de permutaciones con n (clientes) hasta 100, K (grupos) de 8 y en la Figura 3.3 n (clientes) hasta 900, K (grupos) de 15. Se puede observar que el crecimiento es exponencial. En la gráfica que muestra hasta 100 clientes, puede notarse en la línea el rótulo que indica las coordenadas y el valor en ese punto, mientras que, en la gráfica que muestra hasta 900 clientes no están rotulados los valores en la línea por ser valores muy grandes, para ver la tabla que genera estos valores consultar (APENDICE B NUMEROS DE STIRLING DE SEGUNDA CLASE) página 59.

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

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

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

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

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

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

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

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

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

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

4. DESCRIPCIÓN DEL ALGORITMO DE AGRUPAMIENTO PROPUESTO

El algoritmo de agrupamiento desarrollado hace una combinación de estrategias de los algoritmos CA-VRP (Cruz-Chavez & Martinez-Oropeza, 2011) y el algoritmo de barrido (Sweep) que es una variante del algoritmo de pétalos. Incorporando del algoritmo CA-VRP el uso del primer cliente de cada grupo, seleccionado de forma aleatoria como centroide inicial, la incorporación sucesiva de clientes más cercanos al centroide (mientras se satisfaga la restricción de capacidad); y del segundo algoritmo, agregar al final de la formación de cada grupo, a clientes que se encuentren entre el ángulo menor y el ángulo mayor formados por la localización de los clientes de un grupo, teniendo como vértice de dicho ángulo la localización del depósito. En la Figura 4.1 se muestra el pseudocódigo del algoritmo de agrupamiento propuesto, del que se van describiendo en este capítulo cada uno de los puntos más importantes.

```
1.  Leer  Datos de entrada
2.  Calcular  distancias[i][j]=√(x2 – x1)2 + (y2 – y1)2
3.  Hacer
4.      Hacer
            centroide= Elemento de la pila de aleatorios
        Mientras  lista_tabu[centroide]==1
        lista_tabu[centroide]=1
        Vector_solucion[i]=centroide
        i++
5       Cliente_mas_cercano_al _centroide
        Mientras (d+∑ di ) < C
            lista_tabu[centroide]=1
            Vector_solucion[i]=centroide
            i++
            Cliente_mas_cercano_al _centroide
        Repetir
6       Cliente_cercano_dentro_angulo
        Mientras encuentre_cliente
            Si  (d+∑ di ) < C
                lista_tabu[centroide]=1
                Vector_solucion[i]=centroide
                i++
                Cliente_cercano_dentro_angulo
            Fin-si
        Repetir
        Index++
    Mientras  index <= N
7   Escribir  Archivos de Rutas generadas
```

Figura 4. 1 Pseudocódigo del algoritmo propuesto.

4.2 DATOS DE ENTRADA

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

4.2.1 INSTANCIAS

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

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

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

Los archivos de entrada tienen el formato mostrado en la Figura 4.2 en donde la primer línea es la capacidad de los vehículos, las siguientes líneas contienen en sus dos primeras columnas las coordenadas rectangulares que

determinan la posición del depósito y los clientes en plano cartesiano, la tercer columna es la demanda de cada cliente; la segunda línea contiene los datos del depósito, por eso, la demanda es 0 (cero). La cantidad de clientes es calculada cuando se lee el archivo de entrada, contando el número de líneas y restado las dos primeras; el obtener el número de clientes al leer el archivo de entrada evita que ese dato forme parte de una configuración inicial. Punto 1 de Figura 4.2.

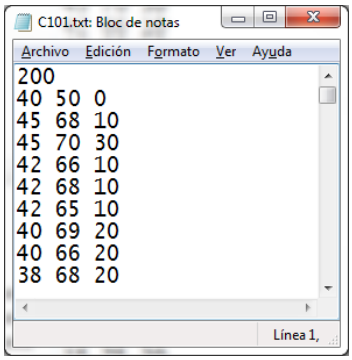


Figura 4. 2 Ejemplo del archivo de entrada.

En la Figura 4.3 se muestra la estructura que almacena una instancia (punto 2 de la Figura 4.3, en donde:

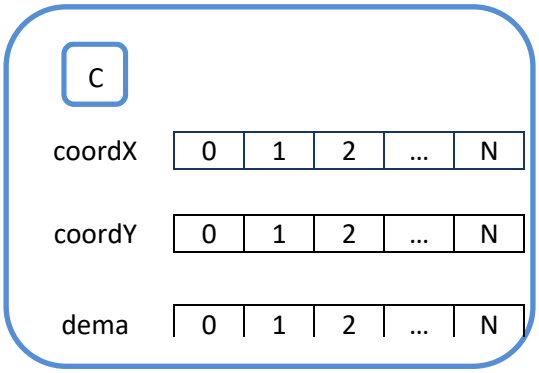


Figura 4. 3 Estructura de los datos de entrada.

C es la capacidad de los vehículos.

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

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

4.3 CRITERIOS PARA LA FORMACION DE LOS GRUPOS

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

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

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

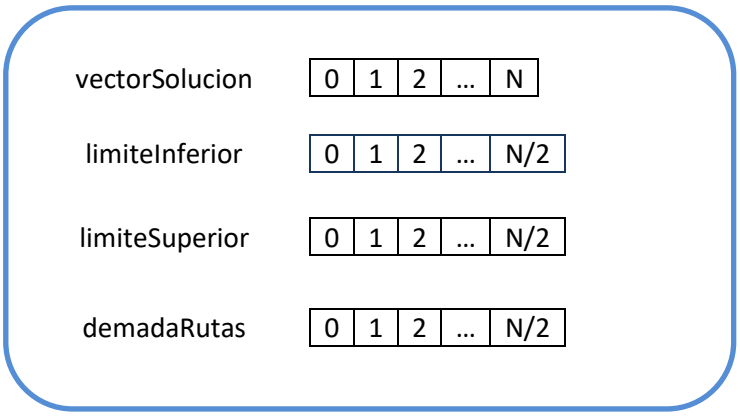


Figura 4. 4 Estructura de la solución.

4.3.1 FORMACIÓN DE LOS AGRUPAMIENTOS

La estrategia para formar los agrupamientos está dividida en tres etapas. la elección de los centroides, la integración de los clientes cercanos al centroide que cumplan con la restricción de capacidad y por último la integración de los

clientes dentro del ángulo; enseguida se explica cada una de las etapas mencionadas.

- Primera etapa.

Elección de los centreoides : Es el primer cliente de cada agrupamiento y se determina de forma aleatoria (punto 4 de la Figura 4.1, se toma del tope de la pila aleatorio, verificando que la bandera del cliente no esté en 1 en el vectorSolucion; esto determina que se trata de un cliente que no ha sido elegido como parte de la solución y por tanto, no es elemento de ningún agrupamiento.

- Segunda etapa.

Clientes cercanos al centroide: La incorporación de los clientes a un agrupamiento ya iniciado por un centroide (punto 5 de la Figura 4.1 se hace buscando en la matriz de distancias en que tenga la mínima con respecto a su centroide y es asignado al agrupamiento después de verificar que cumpla con la restricción de capacidad. Una representación gráfica de la formación de los agrupamientos se ve en la Figura 4.5, en donde los centroides son los centros de cada estrella, que es la forma de un conjunto clientes.

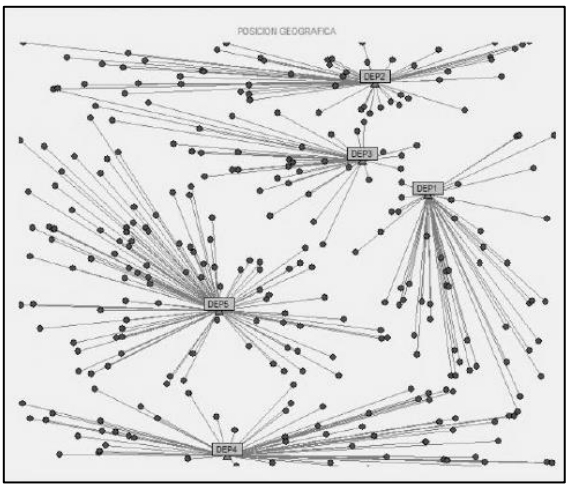


Figura 4. 5 Ejemplo de formación de agrupamientos.

- Tercera etapa.

Clientes dentro del ángulo: Es posible que un agrupamiento se cierre como ya formado porque el siguiente cliente cercano al centroide no cumple con la restricción de capacidad, pero la capacidad sobrante (la capacidad del vehículo menos la suma de las demandas de los clientes) puede permitir la incorporación de un cliente que no sea cercano al centroide (punto 6 de la Figura 4.1, pero se localice en dirección al depósito; esta condición hace que no afecte al costo del recorrido en la segunda fase (ruteo). Para asegurar que un cliente se encuentra en dirección al depósito con respecto al agrupamiento, cuando se incorpora cada cliente al agrupamiento en la primera etapa se van calculando tres valores:

a) AngIzq. Es el ángulo mayor de los clientes de un agrupamiento.

b) AngDer. Es el ángulo menor de los clientes de un agrupamiento.

c) MaximoDist . Es el índice del cliente del agrupamiento que tiene la mayor distancia con respecto al depósito.

Esto tres valores permiten identificar el área que forma un triángulo entre el depósito y el agrupamiento, en la que puede existir un cliente que tenga una demanda menor a la capacidad sobrante de la formación del agrupamiento y como ya se mencionó al estar en el área en dirección al depósito no afecta al costo de recorrido. Los ángulos y la formación del triángulo se muestran en la Figura 4.6 marcados con líneas rojas desde el depósito hasta los clientes con el ángulo menor y el ángulo mayor del agrupamiento como también el arco que marca la distancia del cliente más lejano con respecto al depósito.

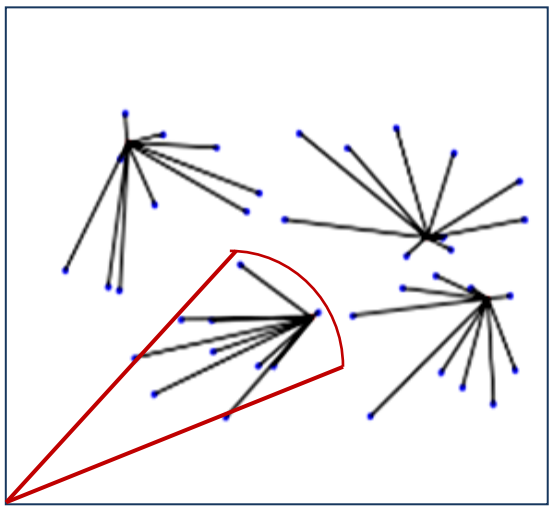


Figura 4. 6 Representación gráfica del ángulo izquierdo y derecho del agrupamiento.

Las tres etapas se repiten de forma iterativa hasta que no existan clientes por asignar a un agrupamiento. Como se observa en el punto 3 del pseudocódigo de la Figura 4.1

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

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

4.4 DATOS DE SALIDA

El resultado de la formación de los agrupamientos (punto 7 de la Figura 4.1, se imprime en un archivo de salida para su uso en la segunda fase (ruteo), con el número de agrupamientos en la primera línea y el índice de cada cliente en las líneas posteriores, comenzando y terminando la serie de números con 0 (cero) por ser el índice del depósito. Como se muestra en la d Figura 4.4

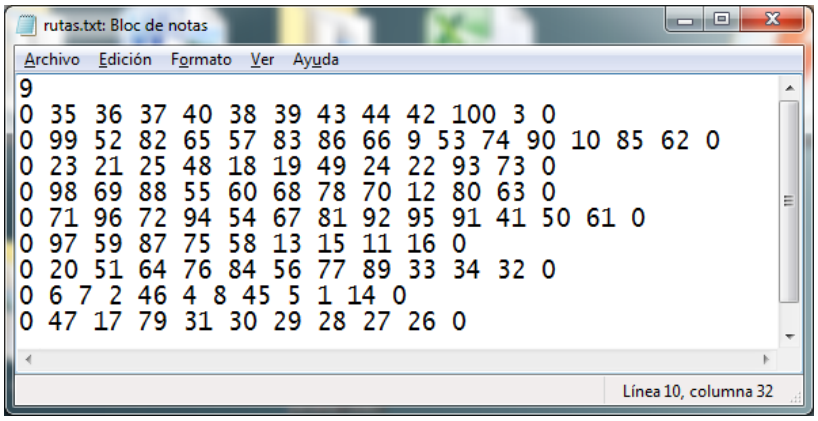


Figura 4.4 Ejemplo de archivo de salida.

Una solución inicial dada por el algoritmo propuesto en su representación gráfica se muestra en la Figura 4.5 en donde, los clientes de cada agrupamiento forman un conjunto de líneas, que inician del centriode al primer cliente incorporado, de ahí al siguiente cliente incorporado y así sucesivamente hasta el último cliente del grupo. En el archivo de salida se agrega al inicio y al final de cada agrupamiento el índice 0 (cero), que es el depósito, porque en la segunda fase del método de solución del VRP usado en esta investigación, los recorridos inician y terminan en el depósito como lo muestra las desigualdades 2.2 y 2.3.

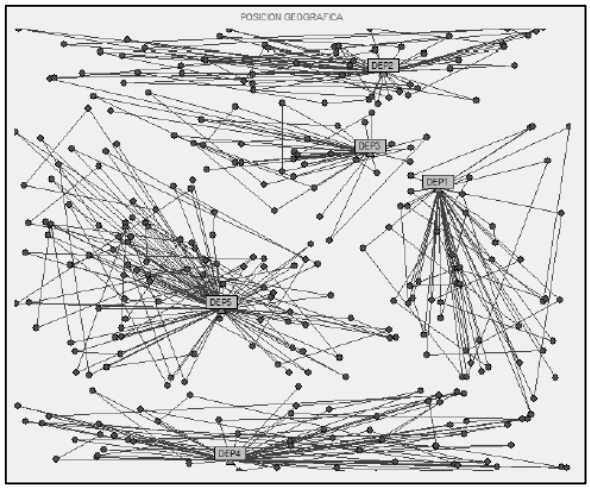


Figura 4.1 Ejemplo de solución inicial.

5. EVALUACIÓN DE RESULTADOS

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

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

5.1 TABLAS COMPARATIVAS

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

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

Tabla 5. 1 Número de rutas generado.

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

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

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

Tabla 5. 2 Instancias de Solomon R.

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

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

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

Tabla 5. 3 Instancias de Solomon C.

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

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

Tabla 5. 4 Instancias de Solomon RC.

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

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

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

Tabla 5. 5 Gehring & Homberger's c1_10.

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

Instancia	Vehículos	Propuesto	Autores	
c2_10_1	30	**28**	LL	
c2_10_2	29	**28**	BC4	05-apr-12
c2_10_3	28	**28**	BC4	05-apr-12
c2_10_4	28	**28**	VCGP	31-jul-12
c2_10_5	30	**28**	VCGP	31-jul-12
c2_10_6	29	**28**	BC4	05-apr-12
c2_10_7	29	**28**	BC4	05-apr-12
c2_10_8	28	**28**	VCGP	31-jul-12
c2_10_9	29	**28**	VCGP	31-jul-12
c2_10_10	28	**28**	VCGP	31-jul-12

Tabla 5. 6 Gehring & Homberger's c2_10.

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

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

Tabla 5. 7 Gehring & Homberger's r1_10.

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

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

Tabla 5. 8 Gehring & Homberger's r2_10.

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

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

Tabla 5. 9 Gehring & Homberger's rc1_10.

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

Instancia	Vehículos	Propuesto	Autores	
rc2_10_1	**20**	**18**	**BC4**	**12-apr-12**
rc2_10_2	18	**18**	VCGP	31-jul-12
rc2_10_3	**18**	**18**	**VCGP**	**31-jul-12**
rc2_10_4	18	**18**	VCGP	31-jul-12
rc2_10_5	**18**	**18**	**VCGP**	**31-jul-12**
rc2_10_6	18	**18**	BC4	05-apr-12
rc2_10_7	**18**	**18**	**VCGP**	**31-jul-12**
rc2_10_8	18	**18**	VCGP	31-jul-12
rc2_10_9	**18**	**18**	**VCGP**	**31-jul-12**
rc2_10_10	18	**18**	VCGP	31-jul-12

Tabla 5. 10 Gehring & Homberger's rc2_10

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

6. CONCLUSIONES

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

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

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

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

7. TRABAJOS FUTUROS

Como trabajo futuro se contempla :

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

REFERENCIAS

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

APENDICE A CÓDIGO

```
18  #include<stdlib.h>
19  #include<stdio.h>
20  #include<math.h>
21  #include<time.h>
23  #define LIMITE_MAX 6000
25  /***********PROTOTIPO DE FUNCIONES**********************/
26  void leeArchivos();
27  void distancias();
28  void aleatorio();
29  void demTotal();
30  void escribeArchivos(int,int);
31  int cliCerca(int, int);
32  int cliCercaAngulo(double, double, double);
34  /*************ESTRUCTURAS*********************************/
35  struct instancia
36  {
37      int C;     //capacidad de los vehiculos (homogenea)
38      int coordX[LIMITE_MAX+2];
39      int coordY[LIMITE_MAX+2];
40      int dema[LIMITE_MAX+2];
41  }datos;
43  struct datosRutas
44  {
45      int vectorSolucion[LIMITE_MAX+2];
46      int limiteInferior[(LIMITE_MAX/2)+2];
47      int limiteSuperior[(LIMITE_MAX/2)+2];
48      int demandaRuta[(LIMITE_MAX/2)+2];
49  }solucion;
50
51  /***********VARIABLES*************************************/
52  int listaTabu[LIMITE_MAX+2]={1,0}; //la posicion 0 esta marcada porque es el deposito
53  double mDistancias[LIMITE_MAX+2][LIMITE_MAX+2];
54  double angulos[LIMITE_MAX+2];
55  int visitados[LIMITE_MAX+2];
56  int pilaAleatoria[LIMITE_MAX+2];
57  int finPila = 1;
58  int cantidad;
59
60  /**************FUNCION PRINCIPAL*****************************/
61  int main(){
62      double angIzq = -400;
63      double angDer = 400;
64      int centroid=0;//primer cliente que forma el grupo
65      int sumDem=0;  //acumulador de las demandas
66      int cliente=0;
67      int vh =0;       //vehiculo
68      int index=0;   //clientes
69      int k=1;       // es el indece que maneja el vectorSolucion[]
70      int z=1;       // es el indece que maneja el limiteInferior[]
71      int auxCliente=0;
72      float totalito = 0;
73      int alto = -100;
74      int bajo = 10000;
75      time_t pi;
```

```
76    time_t pf;
77    time_t totalit = 0;
78
79    leeArchivos();
80    //demTotal();
81    distancias();
82
88    time_t ini = clock();
89    aleatorio();
90    do{
91       //busca un centriode q no este marcado
92       do{
93          finPila--;
94          centroid = pilaAleatoria[finPila];
95       }while(listaTabu[centroid]==1);
96       // guardar en listaTabu los grupos en lugar de los limites
97
98       //asigna el centroide encontrado
99       listaTabu[centroid]=1;  //lo marca
100       solucion.vectorSolucion[k]=centroid;
101       k++;
102       solucion.limiteInferior[z]=centroid;
103       index++;
104       sumDem=0;              //reinicia la sumatoria de la demanda
105       sumDem+=datos.dema[centroid];
106
107       pi = clock();
108       cliente=cliCerca(centroid, 0);
109       pf = clock();
110       totalit += pf - pi;
111       if(cliente != 0){
112          double maximoDist = -1;
113          while((sumDem + datos.dema[cliente]) < datos.C && cliente != 0){
114             listaTabu[cliente]=1;  //marca al nuevo cliente
115             solucion.vectorSolucion[k]=cliente;
116             k++;
117             auxCliente=cliente;
118             if(angIzq < angulos[cliente])
119                angIzq = angulos[cliente];
120             if(angDer > angulos[cliente])
121                angDer = angulos[cliente];
122             if(maximoDist < mDistancias[0][cliente])
123                maximoDist = mDistancias[0][cliente];
124             index++;
125             sumDem += datos.dema[cliente];
126             pi = clock();
127             cliente=cliCerca(centroid, sumDem);
128             pf = clock();
129             totalit += pf - pi;
130          }
131          cliente = cliCercaAngulo(angIzq, angDer, maximoDist);
132          while(cliente != 0){
133             if(sumDem + datos.dema[cliente] < datos.C){
134                listaTabu[cliente]=1;
135                solucion.vectorSolucion[k] = cliente;
136                k++;
137                auxCliente = cliente;
138                index++;
139                sumDem += datos.dema[cliente];
```

```
140                 cliente = cliCercaAngulo(angIzq, angDer, maximoDist);
141             }else
142                 cliente = 0;
143         }
144     }else{
145         auxCliente = centroid;
146     }
147     vh++;
148     solucion.limiteSuperior[z]=auxCliente;
149     solucion.demandaRuta[z]=sumDem;
150     if(sumDem < bajo)
151         bajo = sumDem;
152     if(alto < sumDem)
153         alto = sumDem;
154     totalito += sumDem;
155       //printf(" \n%d",sumDem);
156     z++;
157     //printf("\nindex %d",index);
158   }while(index < cantidad);
159   time_t fin = clock();
160   //printf("\nTiempo total [%f]\n", (float) (fin-ini)/CLOCKS_PER_SEC);
161   //printf("Tiempo de nada [%f]\n", (float) totalit/CLOCKS_PER_SEC);
162   //printf("%d\n", vh);
163   //printf("Promedio [%f]\n", (float) (totalito / vh));
164   //printf("Mayor [%d]\nMenor [%d]\n", alto, bajo);
165
166         /*for(int indice=0, indice2=0;indice<=index;indice++)
167         {printf("%d ",solucion.vectorSolucion[indice]);
168             if(solucion.vectorSolucion[indice]==solucion.limiteSuperior[indice2])
169             {printf("\n");
170              indice2++;
171             }
172         }*/
173 escribeArchivos(vh,index);
174   return 0;
175 }
176
177 ////////////////FUNCIONES////////////////////////////////////
178
179 /*****    LEE ARCHIVOS DE ENTRADA    ******/
180 //los arreglos los comienzo en uno igual  los indices
181 //la posicion  cero es del deposito
182 char archivo[25];
183 void leeArchivos(){
184   FILE *archDatos;
185
186   //printf("Escriba el nombre  del archivo que quiere abrir ");
187   scanf("%s",archivo);
188   if((archDatos = fopen(archivo,"rt"))==NULL){
189     printf("No existe el archivo de datos");
190   }else{
191     //printf(" El archivo es %s\n",archivo);
192     fscanf(archDatos," %d\t ",&datos.C);
193     do{
194       fscanf(archDatos," %d\t ",&datos.coordX[cantidad]);
195       fscanf(archDatos," %d\t ",&datos.coordY[cantidad]);
196       fscanf(archDatos," %d\t ",&datos.dema[cantidad]);
197       cantidad++;
198     }while(!feof(archDatos));
```

```
199         /*for(i=0;i<=2500;i++){
200             fscanf(archDatos," %d\t ",&datos.coordX[i]);
201             fscanf(archDatos," %d\t ",&datos.coordY[i]);
202             fscanf(archDatos," %d\t ",&datos.dema[i]);
203         }*/
204         cantidad--;
205         //printf("%d\n", cantidad);
206         fclose(archDatos);
207     }
208 }
209
210 /*************** MATRIZ DE DISTANCIAS   *************/
211 void distancias(){      //cálculo de las distancias euclidianas
212     int i,j;
213     double xx,yy;
214     for(i=0;i<=cantidad;i++){
215         for(j=i+1;j<=cantidad;j++){
216             xx=(double)(datos.coordX[i]-datos.coordX[j])*(datos.coordX[i]-datos.coordX[j]);
217             yy=(double)(datos.coordY[i]-datos.coordY[j])*(datos.coordY[i]-datos.coordY[j]);
218
219             mDistancias[j][i] = mDistancias[i][j] = sqrt(xx + yy );
220         }
221         if(i != 0){
222             angulos[i] = (180/3.14159)*asin((datos.coordY[i]-datos.coordY[0])/mDistancias[0][i]);
223             if(angulos[i] < 0)
224                 angulos[i] += 360;
225         }
226     }
227 }
228
229 /*************** ALEATORIO  *************/
230 void aleatorio(){
231     int total = 0;
232     int actual = -1;
233     srand(time(NULL));
234     while(total < cantidad){
235         actual = (1+(rand()%cantidad)); // creamos un numero aleatorio
236         if(visitados[actual] == 0){ // si no existe entonces
237             pilaAleatoria[finPila] = actual; // guardamos
238             finPila++; // aumentamos la pila
239             visitados[actual] = 1; // lo visitamos
240             total++; // aumentamos el total de visitados
241         }
242     }
243 }
244
245 /***************BUSCA CLIENTE MAS CERCANO    *************/
246 int cliCerca(int centroid, int acumulado){
247     double menor=1000;
248     double menormenor=1000;
249     int posmm = 0;
250     int posm = 0;
251
252     int min=0;      //primer minimo no marcado en lista tabu
253     double minimo;     //valor de la distancia minima entre el centriode y  un cliente no marcado
254     int i=0;
255     int posicion=0;    // indice del  valor minimo encontrado
256     // el min es el minimo inicial inicial es el primero que en la lista tabu no sea 1
257     for(i=1;i<=cantidad;i++){
```

```
258         if (listaTabu[i]!=1){
259             min=i;
260             posicion = i;
261             break;
262         }
263     }
264 
265     if(min!=0){   //garantiza que eln  el ciclo anterior se encontro un valor
266         // validar q aun haya clientes
267         menormenor = mDistancias[centroid][min];
268         posmm = min;
269         for(i=min;i<=cantidad;i++){
270             if (listaTabu[i]!=1 && i != centroid){
271                 if( mDistancias[centroid][i] < menormenor){
272                     menor = menormenor;
273                     posm = posmm;
274                     menormenor =  mDistancias[centroid][i];
275                     posmm = i;
276                 }
277             }
278         }
279         posicion = posmm;
280         if(acumulado + datos.dema[posmm] > datos.C)
281             posicion = posm;
282     }
283     return(posicion);
284 }
285 // ************************************************
286 int cliCercaAngulo(double izq, double der, double maxDist){
287     int min=0;      //primer minimo no marcado en lista tabu
288     double minimo = 20000;
289     int i=0;
290     int posicion=0;    // indice del  valor minimo encontrado
291     // el min es el minimo inicial inicial es el primero que en la lista tabu no sea 1
292     for(i=1;i<=cantidad;i++){
293         if (listaTabu[i]!=1){
294             min=i;
295             posicion = i;
296             break;
297         }
298     }
299 
300     if(min!=0){   //garantiza que eln  el ciclo anterior se encontro un valor
301         // validar q aun haya clientes
302         for(i=min;i<=cantidad;i++){
303             if (listaTabu[i] != 1){
304                 if(angulos[i] < izq && angulos[i] > der && maxDist >= mDistancias[0][i]){
305                     if(datos.dema[i] < minimo){
306                         minimo =  datos.dema[i];
307                         posicion = i;
308                     }
309                 }
310             }
311         }
312     }
313     return(posicion);
314 }
315 
316 void demTotal(){
```

```
317     int acu=0;
318     int i;
319     for(i=1;i<=cantidad;i++)
320         acu+= datos.dema[i];
321     //printf("La demanda total es  %d  ",acu) ;
322  }
323  /*****     ESCRIBE ARCHIVOS DE SALIDA     ******/
324  //los arreglos los comienzo en uno igual  los indices
325  //la posicion  cero es del deposito
326  void escribeArchivos(int vh,int index){
327     FILE *archSalida;
328     int i,j;
329     //char archivo[25];
330     //printf("Escriba el nombre  del archivo que quiere abrir ");
331     //scanf("%s",archivo);
332     if((archSalida = fopen("rutas.txt","w"))==NULL){
333         printf("No se puede abrir el archivo");
334     }else{
335         fprintf(archSalida,"%d\n", vh);
336        for (i=1,j=1; i<=index;i++){
337     //  fscanf(input,  "%d", &iValor); // &=direcci¢n memoria de var. iValor
338       // printf("%d\n", iValor*iValor);
339        //fprintf(output,"%d\n", iValor*iValor); // agrega una linea (enter)
340  //    for(j=0; j<m;j++){
341  //            fscanf(input,  "%d",&iValor );
342  //             x[i][j]=iValor;
343               //fprintf(output,"%d", iValor);
344               //fprintf(archSalida,"%d ",solucion.vectorSolucion[i]);
345                  if(solucion.vectorSolucion[i]==solucion.limiteSuperior[j])
346                  {//fprintf(archSalida,"%d ",0);
347                   //fprintf(archSalida,"\n");
348                   j++;
349                  }
350
351   // cierra los archivos
352         }
353         }
354    fclose(archSalida);
355   }
356
```

APENDICE B NUMEROS DE STIRLING DE SEGUNDA CLASE

(Nieto Said, 1996)

Una *partición* de un conjunto X es una colección $\{A_i : i \in I\}$ de subconjuntos no vacíos de X, disjuntos dos a dos y tales que su unión sea X. Cada partición de X permite definir una relación de equivalencia $\sim$ en X en la cual $x \sim y$ si y sólo si $x, y \in A_i$ para algún $i \in I$. Recíprocamente cada relación de equivalencia en X induce una partición de X, a saber aquella constituída por las clases de equivalencia. Es claro que esta correspondencia entre particiones de X y relaciones de equivalencia en X es biyectiva.

Cada función $f : X \to Y$ induce una relación de equivalencia $\sim$ en su dominio definiendo $a \sim b$ si y sólo si $f(a) = f(b)$. A la partición de X determinada por esta relación de equivalencia se le llama *núcleo* de la función f. Si f es sobreyectiva y el conjunto Y tiene k elementos $y_1, \ldots, y_k$ entonces el núcleo de f se compone también de k clases, a saber $f^{-1}(y_1), \ldots, f^{-1}(y_k)$. El número de particiones (o lo que es lo mismo, el número de relaciones de equivalencia) que admite un conjunto de n elementos se denomina *número de Bell* de orden n, y lo denotaremos B_n. Aceptaremos por convención que $B_0 = 1$, aunque esto puede también justificarse observando que el conjunto vacío admite exactamente una partición, a saber la partición vacía (es decir la partición sin miembro alguno). El número de particiones que admite un conjunto de n elementos con exactamente k clases se denomina *número de Stirling de segunda clase* con índices n y k y lo denotaremos mediante el símbolo $\left\{ {n \atop k} \right\}$. Es claro que si $n > 0$ entonces $\left\{ {n \atop 0} \right\} = 0$. También es obvio que si $n < k$ entonces $\left\{ {n \atop k} \right\} = 0$. Aceptaremos por convención, o en virtud del mismo razonamiento que hicimos más arriba para B_0, que $\left\{ {0 \atop 0} \right\} = 1$.

A continuación demostraremos varias propiedades de los números de Stirling de segunda clase.

Proposición 6.4.1. *Para todo $n > 0$ se cumple:*

$$\left\{ {n \atop 1} \right\} = \left\{ {n \atop n} \right\} = 1, \quad \left\{ {n \atop 2} \right\} = 2^{n-1} - 1, \quad \left\{ {n \atop n-1} \right\} = \binom{n}{2}$$

Demostración: Si $X = \{x_1, \ldots, x_n\}$ es un conjunto con $n > 0$ elementos entonces la única partición de X con una sola clase es obviamente $\{X\}$. A su vez, la única partición de X con n clases es $\{\{x_1\}, \ldots, \{x_n\}\}$ y así quedan probadas las dos primeras igualdades. En cuanto a la tercera hagamos corresponder a cada subconjunto A de X , no vacío y distinto del propio X, la

partición $\{A, X \setminus A\}$. De este modo se obtienen todas las particiones de X en dos clases, pero cada una de ellas aparece dos veces puesto que A y $X \setminus A$ determinan la misma partición. Como X tiene 2^n subconjuntos, descontando $\emptyset$ y X y dividiendo entre dos resulta $(2^n - 2)/2 = 2^{n-1} - 1$. Por último, para probar la cuarta igualdad basta observar que cualquier partición de X en $n-1$ clases debe constar de una clase de dos elementos y $n-2$ clases de un elemento cada una. Pero una partición de este tipo queda determinada una vez que sabemos cual es la clase con dos elementos, para lo cual hay precisamente $\binom{n}{2}$ posibilidades. □

Proposición 6.4.2. *Para todo $n > 0$ y $k > 0$ se cumple:*

$$\left\{ {n \atop k} \right\} = \left\{ {n-1 \atop k-1} \right\} + k \left\{ {n-1 \atop k} \right\}$$

Demostración: Sea $X = \{x_1, \ldots, x_n\}$. Las particiones de X en k clases pueden clasificarse en dos categorías: las que contienen la clase $\{x_n\}$ y las que no la contienen. Las primeras son $\left\{ {n-1 \atop k-1} \right\}$, puesto que si $\{x_n\}$ es una clase debe haber $k-1$ clases adicionales, cuya unión será $\{x_1, \ldots, x_{n-1}\}$. En las particiones de la segunda categoría el elemento x_n debe pertenecer a alguna de las k clases, que contendrá también algún otro elemento. Quitando x_n resultará una partición de $\{x_1, \ldots, x_{n-1}\}$ con k clases, de las cuales hay $\left\{ {n-1 \atop k} \right\}$. Sin embargo aplicando este procedimiento cada partición de $\{x_1, \ldots, x_{n-1}\}$ en k clases se obtiene k veces, pues x_n podría estar en cualquiera de las k clases. De este modo resulta que el número de particiones en la segunda categoría es $k \left\{ {n-1 \atop k} \right\}$. □

La relación que acabamos de demostrar, análoga a la fórmula de Stifel para los coeficientes binomiales y a la Proposición (6.2.2) para los números de Stirling de primera clase, permite calcular los números de Stirling de segunda clase recursivamente y construir una tabla semejante al triángulo de Pascal:

TABLA DE NUMEROS DE STIRLING DE SEGUNDA CLASE

	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$	$S_{2,4}$	$S_{2,5}$	$S_{2,6}$	$S_{2,7}$	$S_{2,8}$	$S_{2,9}$	$S_{2,10}$
$S^{(1}$	1									
$S_2^{(2}$	1	1								
$S_2^{(3}$	1	3	1							
$S_2^{(4}$	1	7	6	1						
$S_2^{(5}$	1	15	25	10	1					
$S_2^{(6}$	1	31	90	65	15	1				
$S_2^{(7}$	1	63	301	350	140	21	1			
$S_2^{(8}$	1	127	966	1701	1050	266	28	1		
$S_2^{(9}$	1	255	3025	7770	6951	2646	462	36	1	
$S_2^{(10)}$	1	511	9330	34105	42525	22.827	5880	750	45	1

En este triángulo, luego de llenar los lados con ceros y unos, se obtiene cada número sumando al que está en la fila superior y un lugar a la izquierda, el que está encima multiplicado por su número de columna. En la fila 5 y columna 3 se tiene por ejemplo $\left\{ {5 \atop 3} \right\} = 7 + 6 \times 3 = 25$.

APENDICE C COMPLEJIDAD ALGORITMICA

En el desarrollo de un programa computacional resulta necesario definir criterios para medir su rendimiento o comportamiento. Estos criterios se centran principalmente en su simplicidad y en el uso eficiente de los recursos.

Respecto al uso eficiente de los recursos, éste suele medirse en función de dos parámetros: el espacio, es decir, memoria que utiliza, y el tiempo, lo que tarda en ejecutarse. Ambos representan los costes que supone encontrar la solución al problema planteado mediante un algoritmo. Dichos parámetros van a servir además para comparar algoritmos entre sí, permitiendo determinar el más adecuado de entre varios que solucionan un mismo problema.

El análisis de la eficiencia temporal de los algoritmos proporciona una medida teórica, que consiste en obtener una función que acote (por arriba o por abajo) el tiempo de ejecución del algoritmo para unos valores de entrada dados. Esta medida ofrece estimaciones del comportamiento de los algoritmos de forma independiente del ordenador en donde serán implementados y sin necesidad de ejecutarlos. La unidad de tiempo a la que debe hacer referencia esta medida de eficiencia temporal no puede ser expresada en segundos o en otra unidad de tiempo concreta, pues no existe un ordenador estándar al que puedan hacer referencia todas las medidas. Esta medida de eficiencia temporal se define como una función del tamaño o talla de la entrada.

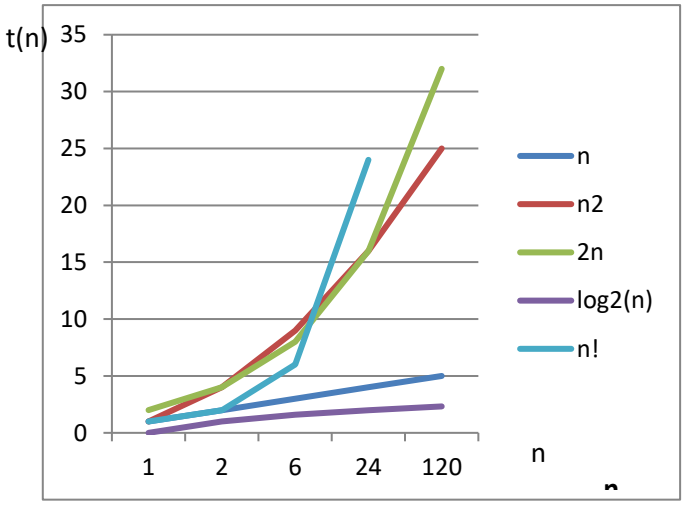
Denotaremos por T(n) el tiempo de ejecución de un algoritmo para una entrada de tamaño n, donde T(n) indica el número de instrucciones ejecutadas por un ordenador idealizado.

El comportamiento de un algoritmo puede cambiar notablemente para diferentes entradas pues para muchos programas el tiempo de ejecución es en realidad una función de la entrada específica, y no sólo del tamaño de ésta. Así suelen estudiarse tres casos para un mismo algoritmo: caso peor, caso mejor y caso medio.

La notación matemática utilizada para representar la complejidad cuando el tamaño de problema (n) es muy grande ($n \to \infty$) usaremos O (O grande), tal que:

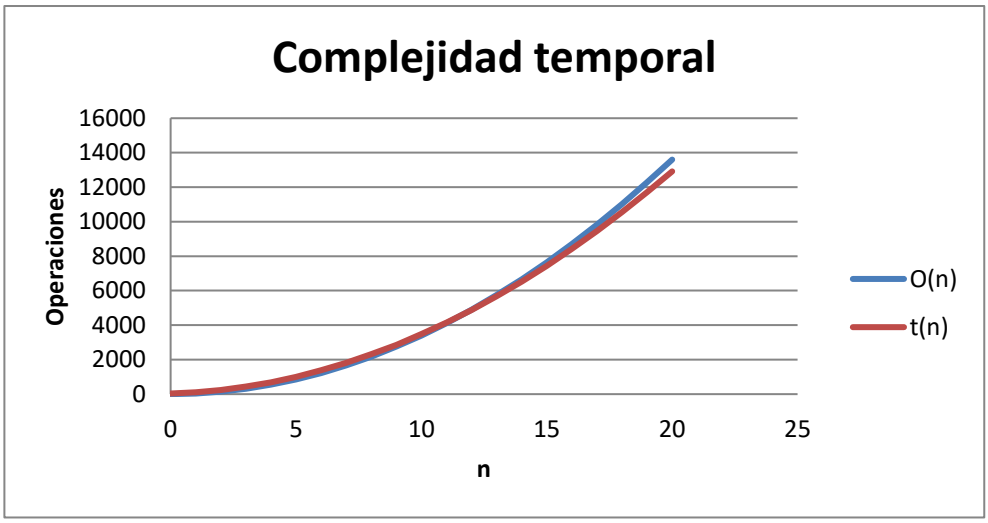
O(1) orden constante
O(log n) orden logarítmico
O(n) orden lineal
O(n2) orden cuadrático
O(na) orden polinomial (a > 2)
O(an) orden exponencial (a > 1)
O(n!) orden factorial

En la siguiente gráfica se muestran algus de la complejidades mencionadas.



Un algoritmo diremos que su orden de complejidad es O(f) si su tiempo de ejecución para el peor caso es de orden O(f), es decir, T(n) es de orden O(f), en la siguiente gráfica se muestra la O(f) calculada del algoritmo de agrupamiento propuesto en esta tesis como O(K n^2) calculado de su complejidad temporal T(N)= 7+ 4N + 9 N^2 + 8 + 7 N + 1 + k (2N + 6 + 10N + 7 +1 +10N^2 + 17N + 11N ^{2}–N -10 +1 +8)+ 5N + 7= 30 N^2 + 44N + 36. En la siguiente gráfica se muestra el crecimiento de los valores de la función O grande con una constante calculada igual a 4 y un n_0 de 10.

.



Es importante señalar que para este algoritmo el cálculo del T(n) es afectado por los datos de las instancias utilizadas, por ello, la cantidad de instrucciones ejecutadas depende de las rutas formadas,las cuales están determinadas por la capacidad de los vehículos y las demandas de los clientes.

	Instrucción	# de ejecuciones	Conjuntos de ejecuciones
leerArchivo();	F1		7+ 4N
distancias();	F2		9 N^2 + 8
aleatorio();	F3		7 N + 1
do{	C0	K	C0 K
do{	C1	N	C1-3 2N
finPila--;	C2	1	
centroid = pilaAleatoria[finPila];	C3	1	
}while(listaTabu[centroid]==1);			
listaTabu[centroid]=1;	C5	1	C5-10 6
solucion.vectorSolucion[k]=centroid;	C6	1	
k++;	C7	1	
solucion.limiteInferior[z]=centroid;	C8	1	
index++;	C9	1	
sumDem=0;	C10	1	
sumDem+=datos.dema[centroid];	F4		10N + 7
cliente=cliCerca(centroid, 0);			
	C11	1	
if(cliente != 0){	C12	N	1
while((sumDem + datos.dema[cliente]) < datos.C && cliente != 0){	C13	1	C13-22 (10+10N+
listaTabu[cliente]=1;	C14	1	7) *
	C15	1	N
solucion.vectorSolucion[k]=cliente;	C16	1	
k++;	C17	1	
auxCliente=cliente;	C18	1	
if(angIzq < angulos[cliente])	C19	1	
angIzq = angulos[cliente];	C20	1	
if(angDer > angulos[cliente])	C21	1	
angDer = angulos[cliente];	C22	1	
index++;			
sumDem += datos.dema[cliente];			
cliente=cliCerca(centroid, sumDem);	F4	10N + 7	
}			
cliente = cliCercaAngulo(angIzq, angDer);	F5	11N + 2	
while(cliente != 0){	C23	N-1	C24-31
if(sumDem + datos.dema[cliente] < datos.C){	C24	1	8+11N +
	C25	1	2 *(N-1)
listaTabu[cliente]=1;	C26	1	
solucion.vectorSolucion[k] = cliente;	C27	1	
	C28	1	
k++;	C29	1	
auxCliente = cliente;	C30	1	
index++;			

sumDem += datos.dema[cliente]; cliente = cliCercaAngulo(angIzq, angDer);	F5	11N + 2	

}
else
cliente = 0;
	C31	1	

}			1

}
 	
else{			

auxCliente = centroid;	C32	1	C33-40 8

}
 	
vh++;
	C33	1	

solucion.limiteSuperior[z]=auxCliente;	C34	1	
solucion.demandaRuta[z]=sumDem;	C35	1	
if(sumDem < bajo)	C36	1	
bajo = sumDem;	C37	1	
if(alto < sumDem)	C38	1	
alto = sumDem;	C39	1	

z++;
}while(index < cantidad);
 	
escribeArchivos(vh,index);
}
	C40
	
	F6
		1
		
		5N + 7
			5N + 7

////////////////FUNCIONES//////////////////// //////////////////			
/***** LEE ARCHIVOS DE ENTRADA ******/ void leeArchivos(){			
FILE *archDatos;	C41	1	C41-45
scanf("%s",archivo);	C42	1	5
if((archDatos = fopen(archivo,"rt"))==NULL){	C43 C44	1 1	
printf("No existe el archivo de datos"); }else{	C45 C46	1 N	
fscanf(archDatos," %d\t ",&datos.C);	C47 C48	1 1	C46-50 4 N
do{ fscanf(archDatos," %d\t ",&datos.coordX[cantidad]);	C49 C50	1 1	
fscanf(archDatos," %d\t ",&datos.coordY[cantidad]);	C51 C52	1 1	
fscanf(archDatos," %d\t ",&datos.dema[cantidad]);			C51-52 2
cantidad++; }while(!feof(archDatos)); cantidad--;			

fclose(archDatos);
}
}
			7 + 4 N

/**************** MATRIZ DE DISTANCIAS
*************/
void distancias(){ //cálculo de las
distancias euclidianas
 	
for(i=0;i<=cantidad;i++){
for(j=i+1;j<=cantidad;j++){
xx=(double)(datos.coordX[i]-
datos.coordX[j])*(datos.coordX[i]-
datos.coordX[j]);
yy=(double)(datos.coordY[i]-
datos.coordY[j])*(datos.coordY[i]-
datos.coordY[j]);
mDistancias[j][i] = mDistancias[i][j]
= sqrt(xx + yy);
}
if(i != 0){
angulos[i] =
(180/3.14159)*asin((datos.coordY[i]-
datos.coordY[0])/mDistancias[0][i]);
if(angulos[i] < 0)
angulos[i] += 360;
}
}
}
	C53
	C54
	C55
	
	C56
	
	C57
	
	C58
	C59
	
	C60
	C61
		N
		N
		3
		
		3
		
		3
		
		1
		5
		
		1
		1
			C55-57
			9 N^2
			
			C58-61
			8
			
			9 N^2 + 8

/**************** ALEATORIO *************/
void aleatorio(){
srand(time(NULL));
while(total < cantidad){
actual = (1+(rand()%cantidad));
if(visitados[actual] == 0){
pilaAleatoria[finPila] = actual;
// guardamos
finPila++; // aumentamos la pila
visitados[actual] = 1;
total++;
}
}
}
	C62
	C63
	C64
	C65
	C66
	C67
	C68
	C69
		1
		N
		2
		1
		1
		1
		1
		1
			1
			C64-69
			7 N+1
			
			7 N + 1

/*****************BUSCA CLIENTE MAS CERCANO
*************/
int cliCerca(int centroid, int acumulado){
 	
for(i=1;i<=cantidad;i++){
if (listaTabu[i]!=1){
min=i;
posicion = i;
break;
}
}
	C70
	C71
	C72
	C73
	C74
	
	C75
		N
		1
		1
		1
		1
		
		1
			C71-74
			4N

if(min!=0){	C76	1	
menormenor =	C77	1	C75-77
mDistancias[centroid][min];	C78	N	3
posmm = min;	C79	1	
for(i=min;i<=cantidad;i++){	C80	1	
if (listaTabu[i]!=1 && i !=	C81	1	C79-84
centroid){	C82	1	6N
if(mDistancias[centroid][i] <	C83	1	
menormenor){	C84	1	
menor = menormenor;			
posm = posmm;			
menormenor =			
mDistancias[centroid][i];	C85	1	
posmm = i;	C86	1	
}	C87	1	
}			
}	C88	1	
posicion = posmm;			C85-88
if(acumulado + datos.dema[posmm] >			4
datos.C)			
posicion = posm;			
}			10N + 7
return(posicion);			
}			
// ********** cliCercaAngulo			
**********			
int cliCercaAngulo(double izq, double der,			
double maxDist){			
for(i=1;i<=cantidad;i++){	C89	N	C89-93
if (listaTabu[i]!=1){	C90	1	4N
min=i;	C91	1	
posicion = i;	C92	1	
break;	C93	1	
}			
}			
if(min!=0){	C94	1	
for(i=min;i<=cantidad;i++){	C95	N	1
if (listaTabu[i] != 1){	C96	1	C96-100
if(angulos[i] < izq &&	C97	3	7N
angulos[i] > der && maxDist >=			
mDistancias[0][i]){	C98	1	
if(datos.dema[i] <	C99	1	
minimo){	C100	1	
minimo =			
datos.dema[i];			
posicion = i;			
}			
}			
}	C101	1	
}			
}			1
return(posicion);			
}			
			11N + 2
/***** ESCRIBE ARCHIVOS DE SALIDA			

******/			
void escribeArchivos(int vh,int index){			
FILE *archSalida;			
int i,j;			
if((archSalida =	C102	1	C102-107
fopen("rutas.txt","w"))==NULL){	C103	1	6
printf("No se puede abrir el	C104	1	
archivo");	C105	1	
}else{	C106	1	
fprintf(archSalida,"%d\n", vh);	C107	1	
for (i=1,j=1; i<=index;i++){	C108	N	
fprintf(archSalida,"%d	C109	1	C109-113
",solucion.vectorSolucion[i]);	C110	1	5 N
	C111	1	
if(solucion.vectorSolucion[i]==solucion.limite	C112	1	
Superior[j])	C113	1	
{fprintf(archSalida,"%d ",0);			
fprintf(archSalida,"\n");			
j++;			
}	C114	1	
}			
}			1
fclose(archSalida);			
}			
			5N + 7

$T(N)= 7+ 4N + 9\ N^2 + 8 + 7\ N + 1 + k\ (2N + 6 + 10N + 7 +1\ +10N^2 + 17N + 11N^2 -N -10 +1 +8\ )+ 5N + 7$

$=9\ N^2 + 11\ N + 16 + K\ (21N^2 + 28N + 13)+ 5N + 7$

$= K\ (21N^2 + 28N + 13)+ 9\ N^2 + 16N + 23 \ = \ 30\ N^2 + 44N + 36$

$O(KN^2)$

La complejidad no puede crecer a n^3 en el caso de que K crezca a n, porque en dicho caso la funciones afectadas por K dejan de ser n2 y se convierten en n, en tal caso cada cliente seria atendido por un vehículo y los cálculos se reducen.